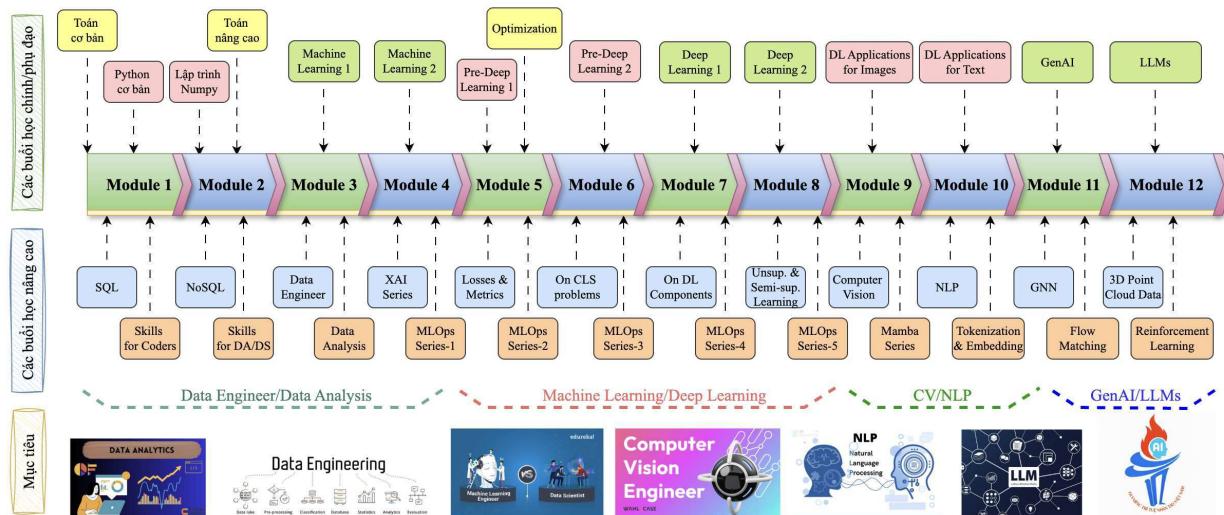




Sách bài tập Lớp AIO (Kèm bài giải; version 2025)



Nhóm tác giả

Dương Trường Bình
Trần Hoàng Duy
Nguyễn Thái Hà (PhD)
Nguyễn Thế Hào
Hồ Quang Hiển
Nguyễn Thọ Anh Khoa
Nguyễn Anh Khôi
Vũ Yến Linh
Trần Minh Nam
Nguyễn Đăng Nhã

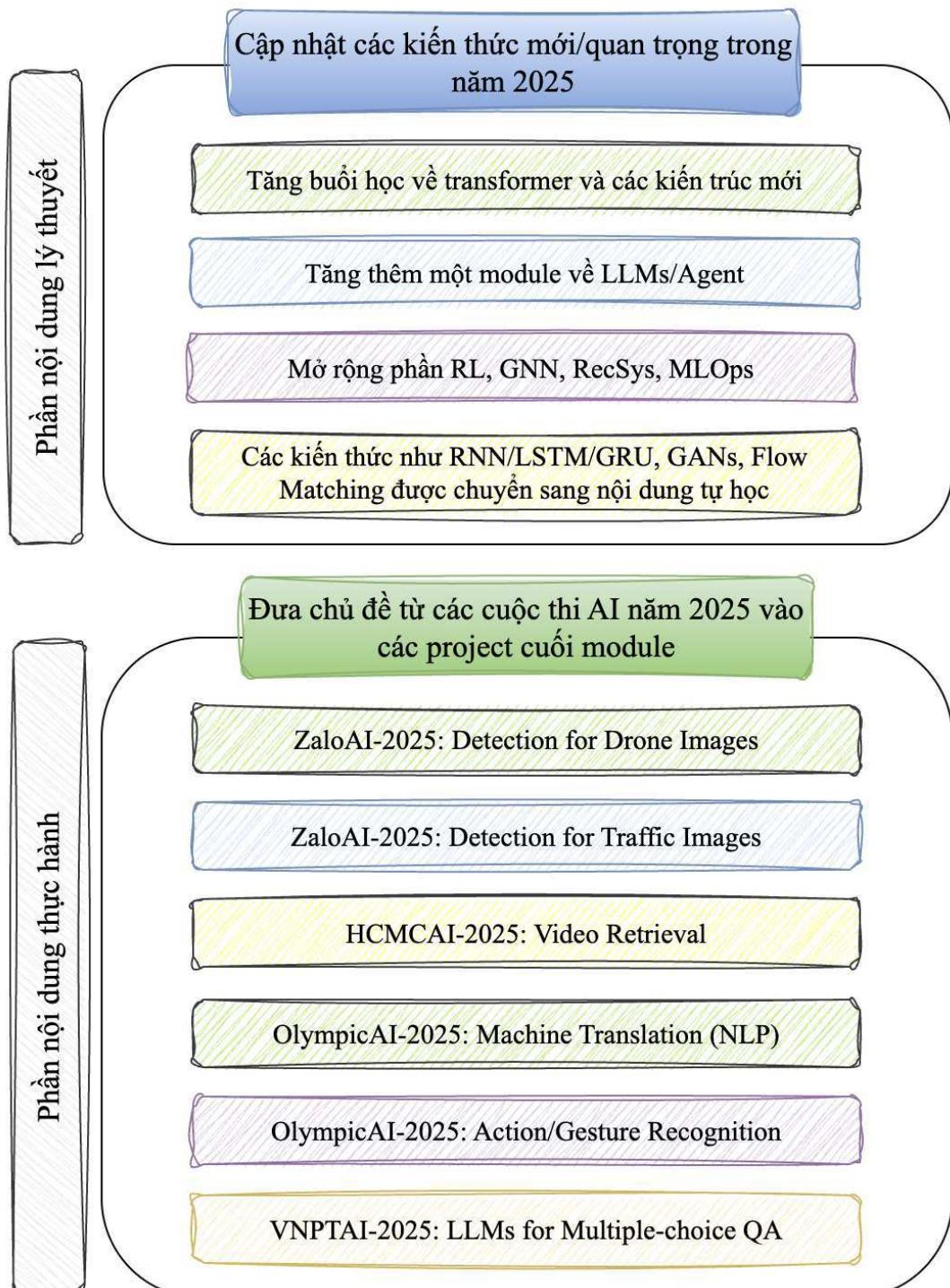
Trần Đại Nhân
Vũ Hoàng Nguyên
Lê Hữu Phước
Nguyễn Quốc Thái (MSc)
Dương Đình Thắng
Nguyễn Phúc Thịnh
Dương Nguyễn Thuận
Nguyễn Thị Ngọc Trúc
Nguyễn Đình Vinh (PhD)
Đinh Quang Vinh (PhD)

Nhóm Research và OpsTeam

Phạm Minh Châu
Tạ Hữu Anh Dương
Nguyễn Thành Huy (MSc)
Nguyễn Tiến Huy
Lê Tấn Anh Huy

Xin Quý Hùng
Nguyễn Ngọc Kim Như
Nguyễn Hồng Phúc (PhD)
Nguyễn Thành Tâm (MSc)
Đoàn Đức Tuấn

Kế hoạch cập nhật nội dung AIO cho năm 2026



Mục lục

I Module 1: Toán cơ bản và lập trình Python	25
1 Python cơ bản và các hàm activation	26
1.1 Giới thiệu	26
1.2 Câu hỏi tự luận	29
1.3 Câu hỏi trắc nghiệm	40
1.4 Phụ lục	49
2 Cấu trúc dữ liệu và bài toán word suggestion	50
2.1 Giới thiệu	50
2.2 Câu hỏi tự luận	53
2.2.1 Getting Max Over Kernel	53
2.2.2 Character Counting	53
2.2.3 Word Counting	54
2.2.4 Levenshtein Distance	56
2.3 Câu hỏi trắc nghiệm	61
3 Lập trình hướng đối tượng và cài đặt class trong PyTorch	74
3.1 Câu hỏi tự luận	74
3.1.1 Xây dựng class tính Sigmoid	74
3.1.2 Xây dựng Ward	75
3.1.3 Xây dựng class Stack	78
3.1.4 Xây dựng class Queue	80
3.2 Câu hỏi trắc nghiệm	82
4 Project 1: Triển khai hệ thống đơn giản với Streamlit	93
4.1 Giới thiệu	93
4.1.1 Lợi ích của Streamlit	94
4.1.2 Một số hàm thông dụng trong Streamlit	94

4.2	Thực hành Streamlit cơ bản	96
4.2.1	Giới thiệu ứng dụng	96
4.2.2	Pipeline tổng quan	96
4.2.3	Cài đặt môi trường và các thư viện cần thiết	96
4.2.4	Cài đặt thư viện cần thiết	97
4.2.5	Thiết Kế Kiến Trúc Ứng Dụng	97
4.3	Xây dựng ứng dụng tính hàm toán học giao thừa	107
4.3.1	Phân tích, thiết kế ứng dụng	107
4.3.2	Xây dựng ứng dụng	108
4.3.3	Xây dựng tính năng phân quyền và xác thực người dùng	111
4.4	Câu hỏi trắc nghiệm	116
5	Project 2: Xây dựng Chatbot cá nhân nhanh chóng với RAG	120
5.1	Giới thiệu	120
5.2	Xây dựng chương trình	122
5.2.1	Chương trình RAG	122
5.3	Xây dựng giao diện	131
5.3.1	Cài đặt môi trường conda	131
5.3.2	Cài đặt thư viện cần thiết	132
5.3.3	Khởi tạo Session State	133
5.3.4	Định nghĩa các hàm hỗ trợ	134
5.3.5	Xây dựng giao diện người dùng	137
5.3.6	Chạy ứng dụng và demo từng bước	139
5.3.7	Xây dựng khung chat	143
5.4	Câu hỏi trắc nghiệm	144
II	Module 2: Toán nâng cao và lập trình nâng cao	150
6	Numpy và đại số tuyến tính	151
6.1	Giới thiệu	151

6.2	Câu hỏi tự luận	152
6.2.1	Các phép toán trên vector và ma trận	152
6.2.2	Cosine Similarity	154
6.2.3	Background subtraction (tách nền)	155
6.3	Câu hỏi trắc nghiệm	158
6.3.1	Numpy	158
6.3.2	Xử lý ảnh	163
6.3.3	Phân tích dữ liệu dạng bảng	167
7	Xác suất và bài toán phân loại dùng Bayes	172
7.1	Giới thiệu	172
7.2	Câu hỏi tự luận	174
7.2.1	Hàm tạo training dataset	175
7.2.2	Hàm tính prior probabilities	175
7.2.3	Hàm tính conditional probabilities	176
7.2.4	Hàm lấy feature index	177
7.2.5	Hàm huấn luyện mô hình Naive Bayes	178
7.2.6	Hàm dự đoán cho một test sample	179
7.3	Câu hỏi trắc nghiệm	181
7.3.1	Binary Classification - Play Tennis	181
7.3.2	Multi-label Classification - Traffic Data	184
7.3.3	Iris Classification	187
8	Thống kê và các phép đo similarity	189
8.1	Mô tả	189
8.2	Câu hỏi trắc nghiệm	190
8.2.1	Basic Probability	190
8.2.2	Tabular Data Analysis	194
8.2.3	Text Retrieval	198
8.2.4	Plagiarism Checker	200
8.3	Phụ lục	205

9 Project 1: Vector database cho các ứng dụng AI	206
9.1 Giới thiệu	206
9.2 Các loại cơ sở dữ liệu truyền thống	207
9.2.1 SQL Database	207
9.2.2 NoSQL Database	208
9.3 Vector Database	210
9.3.1 Khái niệm	210
9.3.2 Embedding	211
9.3.3 Indexing trong Vector Database	212
9.4 Distance Metrics and Loss Functions	217
9.4.1 L2 Distance (Euclidean Distance)	218
9.4.2 L1 Distance (Manhattan Distance)	218
9.4.3 Cosine Similarity / Cosine Distance	219
9.4.4 Dot Product (Inner Product)	219
9.5 Face-Based Employee Check-in System	220
9.5.1 Chuẩn bị tập dữ liệu	221
9.5.2 Vectorize Image thành raw pixel	222
9.5.3 Xây dựng hàm tìm kiếm và hiển thị kết quả	224
9.5.4 Vectorize image thành feature maps	226
9.5.5 Cập nhật lại hàm tìm kiếm	228
9.6 Project 2: Cocktail Suggestion System	232
9.6.1 Thiết lập kết nối	234
9.6.2 Xử lý dữ liệu	237
9.6.3 Hệ thống Recommendation	244
9.6.4 Các phương thức tìm kiếm cocktail	247
9.6.5 Build Streamlit App	248
9.7 Câu hỏi trắc nghiệm	250
10 Project 2: Đánh giá cảm xúc khách hàng dùng Vector database	255
10.1 Giới thiệu	255

10.2 Xây dựng chương trình phân loại tin nhắn Spam với Naive Bayes	258
10.2.1 Tải và khám phá bộ dữ liệu	258
10.2.2 Cài đặt và Import thư viện	259
10.2.3 Đọc và tách dữ liệu	260
10.2.4 Tiền xử lý dữ liệu	261
10.2.5 Chia bộ dữ liệu	264
10.2.6 Huấn luyện mô hình	264
10.2.7 Đánh giá mô hình	265
10.2.8 Thực hiện dự đoán	266
10.3 Xây dựng chương trình phân loại tin nhắn Spam với Cơ sở dữ liệu Vector	267
10.3.1 Thiết lập môi trường và Tải dữ liệu	267
10.3.2 Đọc và Chuẩn bị Dữ liệu	269
10.3.3 Chuẩn bị Mô hình Embedding	269
10.3.4 Vector hóa Dữ liệu và Tạo Metadata	270
10.3.5 Xây dựng Cơ sở dữ liệu Vector và Chia Dữ liệu	271
10.3.6 Xây dựng Logic Phân loại và Đánh giá	272
10.3.7 Đánh giá Mô hình trên Tập Test	275
10.3.8 Xây dựng Pipeline Phân loại Hoàn chỉnh	277
10.3.9 Kiểm thử Pipeline	278
10.4 Câu hỏi trắc nghiệm	280
III Module 3: Máy học cơ bản	286
11 Trực quan hóa và phân tích dữ liệu dùng Pandas	287
11.1 Giới thiệu	287
11.2 Tìm hiểu về Pandas	290
11.2.1 Cài đặt và Import thư viện	290
11.2.2 Làm việc với dữ liệu cơ bản	291
11.2.3 Truy xuất và xử lý dữ liệu	297

11.2.4	Làm sạch dữ liệu	305
11.3	IMDB Movie Dataset	308
11.3.1	Read data	308
11.3.2	View data	309
11.3.3	Understand some basic information about the data . .	310
11.3.4	Data Selection – Indexing and Slicing data	312
11.3.5	Data Selection – Based on Conditional filtering	314
11.3.6	Groupby Operations	314
11.3.7	Sorting Operations	315
11.3.8	View missing values	316
11.3.9	Deal with missing values - Deleting	316
11.3.10	Dealing with missing values - Filling	317
11.3.11	apply() functions	317
11.4	Time Series Dataset	319
11.4.1	Read dataset	320
11.4.2	Time-based indexing	322
11.4.3	Visualizing time series data:	324
11.4.4	Seasonality	325
11.4.5	Frequencies	327
11.4.6	Resampling	329
11.4.7	Rolling windows	333
11.4.8	Trends	334
11.5	Câu hỏi trắc nghiệm	337
12	K-Nearest Neighbors và K-Means	342
12.1	Giới thiệu	342
12.2	Thuật toán KNN	344
12.2.1	KNN for Diabetes Regression	346
12.2.2	KNN for IRIS Classification	348
12.2.3	KNN for IMDB Classification	351
12.3	Thuật toán phân cụm K-Means	355

12.3.1	Cài đặt thư viện	356
12.3.2	Tải về bộ dữ liệu	356
12.3.3	Phân tích dữ liệu	357
12.3.4	Chuẩn hoá dữ liệu	357
12.3.5	Huấn luyện và đánh giá mô hình	358
12.3.6	Tối ưu lựa chọn tham số cụm	359
12.4	Câu hỏi trắc nghiệm	360
13	Decision tree cho bài toán phân loại và hồi quy	364
13.1	Giới thiệu	364
13.2	Cây quyết định ứng dụng cho bài toán phân loại	366
13.2.1	Cài đặt và Import thư viện	368
13.2.2	Đọc và khám phá bộ dữ liệu	369
13.2.3	Trực quan hóa dữ liệu	369
13.2.4	Chia tập dữ liệu và chuẩn hóa	373
13.2.5	Huấn luyện và đánh giá mô hình	373
13.3	Cây quyết định ứng dụng cho bài toán hồi quy	375
13.3.1	Cài đặt và Import thư viện	376
13.3.2	Đọc và khám phá bộ dữ liệu	376
13.3.3	Trực quan hóa dữ liệu	377
13.3.4	Chia tập dữ liệu và chuẩn hóa	378
13.3.5	Huấn luyện và đánh giá mô hình	379
13.4	Câu hỏi trắc nghiệm	381
14	Project 1: Phân loại chủ đề của một bài báo (dùng KNN, Decision Tree, và NBC)	387
14.1	Giới thiệu	387
14.2	Xây dựng chương trình phân loại topic của publication abstract	389
14.2.1	Cài đặt và Import thư viện	389
14.2.2	Đọc và khám phá bộ dữ liệu	390
14.2.3	Tiền xử lý dữ liệu	394

14.2.4	Mã hóa văn bản	384
14.2.5	Huấn luyện và đánh giá mô hình phân loại	391
14.3	Câu hỏi trắc nghiệm	432
15	Project 2: Phân loại khả năng mắc bệnh tim dựa vào các triệu chứng (dùng KNN, Decision Tree, và NBC)	437
15.1	Dẫn nhập	437
15.2	Cài đặt chương trình	440
15.2.1	Tổng quan về bộ dữ liệu	440
15.2.2	Kỹ thuật xử lý đặc trưng	443
15.2.3	Naive Bayes Classifier	453
15.2.4	K-Nearest Neighbors	458
15.2.5	Decision Tree	461
15.2.6	K-means	465
15.2.7	Cải tiến: Ensemble	467
15.3	Câu hỏi trắc nghiệm	472
IV	Module 4: Máy học (tập trung cho dữ liệu bảng và time-series)	476
16	Random Forest và AdaBoost	477
16.1	Giới thiệu	477
16.2	Kỹ thuật Ensemble learning	479
16.2.1	Tổng quan về phương pháp	479
16.2.2	Rừng ngẫu nhiên (Random Forest)	481
16.2.3	Mô hình AdaBoost	484
16.3	Huấn luyện mô hình trên bộ dữ liệu Housing	486
16.3.1	Chuẩn bị thư viện	486
16.3.2	Chuẩn bị dữ liệu	487
16.3.3	Mã hóa thuộc tính văn bản	487
16.3.4	Huấn luyện và đánh giá các mô hình	489

16.4 Câu hỏi trắc nghiệm	495
17 Gradient Boosting và XGBoost	501
17.1 Giới thiệu	501
17.2 Mô hình Gradient Boosting	503
17.2.1 Tổng quan	503
17.2.2 Thuật toán Gradient Boosting	503
17.2.3 Huấn luyện mô hình trên bộ dữ liệu Housing	506
17.3 Mô hình XGBoost cho Classification	510
17.3.1 Tổng quan	510
17.3.2 Các bước trong XGBoost cho Classification	511
17.3.3 Huấn luyện mô hình cho bộ dữ liệu phân loại chất lượng rượu vang	515
17.4 Câu hỏi trắc nghiệm	519
18 XGBoost và LightGBM	523
18.1 Giới thiệu	523
18.2 Mô hình XGBoost cho bài toán hồi quy	525
18.2.1 Tổng quan	525
18.2.2 Các bước trong XGBoost cho bài toán hồi quy	526
18.2.3 Huấn luyện mô hình cho bộ dữ liệu dự đoán giá thuê phòng Airbnb	529
18.3 Mô hình LightGBM	536
18.3.1 Tổng quan	536
18.3.2 GOSS và EFB	536
18.3.3 Huấn luyện mô hình LightGBM cho bài toán hồi quy - Bộ dữ liệu Airbnb	539
18.3.4 Huấn luyện mô hình cho bài toán phân loại - bộ dữ liệu phân loại chất lượng rượu vang	543
18.4 Câu hỏi trắc nghiệm	548
19 Project 1: Phân loại chủ đề của một bài báo (dùng các giải	

thuật đã học)	552
19.1 Giới thiệu	552
19.2 Xây dựng chương trình phân loại topic của publication abstract	554
19.2.1 Cài đặt và Import thư viện	554
19.2.2 Đọc và khám phá bộ dữ liệu	555
19.2.3 Tiền xử lý dữ liệu	559
19.2.4 Mã hóa văn bản	562
19.2.5 Huấn luyện và đánh giá mô hình phân loại	569
19.3 Câu hỏi trắc nghiệm	588
20 Project 2: Phân loại khả năng mắc bệnh tim dựa vào các triệu chứng (dùng các giải thuật đã học)	594
20.1 Giới thiệu	594
20.2 Cài đặt chương trình	596
20.2.1 Bộ dữ liệu	596
20.2.2 Cải thiện đặc trưng	598
20.2.3 Random Forest	609
20.2.4 AdaBoost	618
20.2.5 Gradient Boosting	623
20.2.6 XGBoost	628
20.2.7 Tổng hợp kết quả	633
20.2.8 Cài đặt UI và Deploy	634
20.3 Câu hỏi trắc nghiệm	642
V Module 5: Nền tảng cho deep learning (1)	651
21 Linear Regression (trình bày theo cách cơ bản)	652
21.1 Giới thiệu	652
21.2 Các bước trong Linear Regression cho bài toán hồi quy	654
21.3 Chuẩn bị dữ liệu	655
21.4 Huấn luyện mô hình	657

21.5 Chuẩn hoá đặc trưng	679
21.6 Câu hỏi trắc nghiệm	683
22 Linear Regression (ứng dụng Numpy và Đại số tuyến tính) 688	
22.1 Giới thiệu	688
22.2 Mô hình Linear Regression	691
22.2.1 Tổng quan	691
22.2.2 Linear Regression áp dụng từng mẫu dữ liệu	692
22.2.3 Linear Regression cho batch (mini-batch hoặc full batch) 693	
22.3 Xây dựng mô hình Linear Regression cho bài toán advertising	694
22.4 Xây dựng mô hình Linear Regression cho bài toán dự đoán Bitcoin	702
22.5 Câu hỏi trắc nghiệm	714
23 Tính ngẫu nhiên và Giải thuật di truyền 719	
23.1 Giới thiệu	719
23.2 Thực hành	723
23.2.1 Bài tập 1: OneMax và Trap Fitness	723
23.2.2 Bài tập 2: Linear Regression	727
23.2.3 Bài tập 3: Text Classification	730
23.3 Câu hỏi trắc nghiệm	734
24 Project 1: Đánh giá các thuật toán trên bài toán xâm nhập mặn (Linear and non-linear regressions) 741	
24.1 Giới thiệu	741
24.2 Mô hình Polynomial Regression	744
24.3 Mô hình Ridge Regression (L2)	744
24.4 Mô hình Lasso Regression (L1)	745
24.5 Xây dựng mô hình hồi quy nâng cao cho bài toán dự đoán giá nhà	747
24.6 Câu hỏi trắc nghiệm	760

25 Project 2: Xây dựng công cụ tối ưu hóa quy trình phân bổ hàng hóa cho chuỗi cửa hàng thời trang	766
25.1 Giới thiệu	766
25.2 Lý thuyết	771
25.2.1 Tại sao cần tối ưu hóa phân bổ hàng hóa?	771
25.2.2 Định nghĩa bài toán	772
25.3 Xây dựng hệ thống	776
25.3.1 Đọc dữ liệu	776
25.3.2 Phương pháp dựa trên các luật	778
25.3.3 Phương pháp thuật toán di truyền	781
25.3.4 So sánh hiệu năng và lựa chọn thuật toán	786
25.4 Hướng cải tiến	788
25.4.1 Tích hợp dữ liệu thực và mô hình dự báo	788
25.4.2 Cải tiến và tối ưu hóa giải thuật di truyền	790
25.4.3 Áp dụng thuật toán đa mục tiêu (NSGA-II)	791
25.5 Câu hỏi trắc nghiệm	793
VI Module 6: Nền tảng cho deep learning (2)	797
26 Logistic Regression	798
26.1 Giới thiệu	798
26.2 Thực hành	801
26.2.1 Bài tập 1: Titanic Survival Prediction	801
26.2.2 Bài tập 2: Twitter Sentiment Analysis	805
26.2.3 Bài tập 3: MNIST Classification (One-vs-All)	809
26.3 Câu hỏi trắc nghiệm	814
27 Softmax Regresion	820
27.1 Giới thiệu	820
27.2 Chi tiết về Softmax Regression	821
27.3 Softmax với tham số temperature	824

27.4 Credit Card Fraud Detection	827
27.5 Twitter Sentiment Analysis	839
27.6 Câu hỏi trắc nghiệm	850
28 Multi-layer Perceptron	854
28.1 Giới thiệu	854
28.2 Thực hành	857
28.2.1 Bài tập 1: Dự đoán hiệu suất xe hơi	857
28.2.2 Bài tập 2: Phân loại với dữ liệu phi tuyến	861
28.2.3 Bài tập 3: Phân loại cảm xúc trên ảnh	867
28.3 Câu hỏi trắc nghiệm	873
29 Project 1: Dự đoán chính xác cho dữ liệu time-series với mô hình DLinear và NLinear	879
29.1 Giới thiệu	879
29.2 Kiến thức cơ bản về dự báo chuỗi thời gian	886
29.2.1 Bài toán dự báo chuỗi thời gian là gì?	886
29.2.2 Dữ liệu trông như thế nào?	887
29.2.3 Chúng ta muốn dự đoán cái gì?	888
29.2.4 Liên hệ tới các mô hình trong project này	889
29.3 Mô hình LTSF-Linear: Linear	889
29.4 Mô hình LTSF-Linear: NLinear	891
29.5 Mô hình LTSF-Linear: DLinear	893
29.6 Xây dựng các mô hình LTSF-Linear từ đầu (from scratch)	896
29.7 Bài toán dự đoán giá cổ phiếu sử dụng LTSF-Linear Models	902
29.8 Hướng cải tiến và mở rộng	923
29.8.1 Mở rộng sang bài toán đa biến	923
29.8.2 Hybrid model: kết hợp phân rã và transformer	924
29.9 Câu hỏi trắc nghiệm	926
30 Project 2: Tách phân khúc khách hàng cho chiến lược marketing dùng Unsupervised Learning	931

30.1	Phân khúc khách hàng	931
30.2	Phân tích và làm sạch dữ liệu	935
30.2.1	Tải và kiểm tra dữ liệu ban đầu	935
30.2.2	Làm sạch dữ liệu	936
30.2.3	EDA	937
30.3	Tạo đặc trưng	943
30.3.1	Tại sao cần mở rộng Features?	943
30.3.2	Xây dựng bộ đặc trưng	944
30.3.3	Chuẩn hóa và biến đổi Dữ liệu	945
30.4	Mô hình hóa và phân cụm	949
30.4.1	Giới thiệu về K-Means clustering	949
30.4.2	Xác định số cụm tối ưu	950
30.4.3	Phân tích thành phần chính (PCA)	950
30.4.4	So sánh phân cụm K=3 và K=4	951
30.4.5	Phân tích chuyên sâu biểu đồ Radar	952
30.4.6	Quy trình định danh khách hàng với ChatGPT	955
30.5	Các hướng cải tiến	957
30.6	Câu hỏi trắc nghiệm	959

VII Module 7: Hiểu rõ cách hoạt động của Deep MLP 963

31	Bước Initializer và Optimizer cho mô hình deep learning	964
31.1	Mô Tả Lý Thuyết	964
31.2	Bài Tập và Gợi Ý	968
32	Project 1: Vấn đề gradient vanishing và các giải pháp	986
32.1	Giới thiệu	986
32.2	Cài đặt chương trình	991
32.3	Câu hỏi trắc nghiệm	1018
32.4	Phụ lục	1021

33 Project 2: Điều khiển thiết bị IoT qua việc nhận dạng các cử chỉ tay	1023
33.1 Giới Thiệu	1023
33.2 Thực Hành	1027
33.3 Phụ Lục	1048
 VIII Module 8: Những kiến trúc deep learning	
34 Convolution Neural Network	1056
34.1 Giới thiệu	1056
34.2 Thực hành	1060
34.3 Câu hỏi trắc nghiệm	1086
34.4 Phụ lục	1090
35 Khảo sát và đánh giá tất cả các mô hình CNN nổi bật	1091
35.1 Giới thiệu	1091
35.2 Bài tập	1093
35.2.1 Phần lập trình	1093
35.2.2 Phần trắc nghiệm	1115
35.3 Phụ lục	1118
36 Recurrent Neural Networks	1120
36.1 Giới thiệu	1120
36.2 Bài tập	1124
36.2.1 Phần lập trình	1124
36.2.2 Phần trắc nghiệm	1148
36.3 Phụ Lục	1151
37 Mô hình Transformer	1152
37.1 Transformer	1152
37.1.1 Kiến trúc Transformer	1152

37.1.2 Text Classification	1160
37.2 Text classification using BERT	1169
37.3 Vision Transformer	1174
37.4 Câu hỏi trắc nghiệm	1182
37.5 Phụ lục	1184
38 Project 1: OCR (Yolov9+CNN) để trích xuất thông tin ID Card	1186
38.1 Giới thiệu	1186
38.2 Thiết lập chương trình	1188
38.3 Câu hỏi trắc nghiệm	1225
38.4 Phụ lục	1228
39 Project 2: Visual Question Answering (luyện tập dùng 2 kiểu dữ liệu text và ảnh)	1229
39.1 Giới thiệu	1229
39.2 Cài đặt chương trình	1232
39.3 Câu hỏi trắc nghiệm	1264
39.4 Phụ lục	1268
IX Module 9: Deep learning cho dữ liệu ảnh	1269
40 Transfer learning và các bài toán ở dạng domain conversion (segmentation và super-resolution)	1270
40.1 Lý Thuyết	1270
40.2 Bài Tập và Gợi Ý	1274
40.3 Trắc Nghiệm	1292
40.4 Phụ Lục	1294
41 Object Detection 1: Thảo luận YOLO v1 đến v3	1296
41.1 Giới thiệu	1296
41.2 Nội dung chính	1298

41.2.1	Bài toán 1: Classification (Phân loại một object trong hình ảnh)	1298
41.2.2	Bài toán 2: Classification + Bounding Box Regression .	1305
41.2.3	Bài toán 3: Classification (> 2 objects in an image) + Bounding Box Regression	1314
41.2.4	Bài toán 4: YOLOv1	1329
41.3	Câu hỏi trắc nghiệm	1360
41.4	Phụ lục	1363
42	Object Detection 2: Thảo luận YOLO v4 đến v10	1364
42.1	Giới thiệu	1364
42.2	Nội dung chính	1366
42.2.1	Bài toán 1: Object Tracking	1366
42.2.2	Bài toán 2: Object Counting	1375
42.2.3	Bài toán 3: Open Vocab Detection	1380
42.3	Câu hỏi trắc nghiệm	1384
42.4	Phụ lục	1387
X	Module 10: Deep learning cho dữ liệu text	1389
43	Bài toán POS và Medical NER	1390
43.1	Giới thiệu	1390
43.2	Part-of-Speech Tagging	1392
43.3	Medical NER	1398
43.4	Câu hỏi trắc nghiệm	1409
43.5	Phụ lục	1412
44	Project 1: Aspect-Based Sentiment Analysis (Text Classification + NER)	1413
44.1	Giới thiệu	1413
44.2	Aspect Term Extraction	1415

44.3 Aspect Term Sentiment Classification	1420
44.4 Câu hỏi trắc nghiệm	1425
44.5 Phụ lục	1428
45 Project 2: Xây dựng hệ thống end-to-end Question Answering	429
45.1 Giới thiệu	1429
45.2 Cài đặt chương trình	1431
45.2.1 Dataset SQuAD2.0	1431
45.2.2 Reader: DistilBERT	1431
45.2.3 Retriever: Faiss	1444
45.2.4 Áp dụng mô hình hỏi-đáp để trả lời câu hỏi	1449
45.3 Câu hỏi trắc nghiệm	1453
45.4 Phụ lục	1457
46 Text Generation và Machine Translation	1458
46.1 Giới thiệu	1458
46.2 Transformer-based Translation	1460
46.3 Câu hỏi trắc nghiệm	1471
46.4 Phụ lục	1474
47 Project 1: Dịch ngôn ngữ Anh-Việt với điều kiện tài nguyên ít	1475
47.1 Giới thiệu	1475
47.2 Fine-tuning mBART50	1477
47.3 Câu hỏi trắc nghiệm	1486
47.4 Phụ lục	1488
48 Project 2: Xây dựng chương trình sinh thơ	1489
48.1 Giới thiệu	1489
48.2 Cài đặt chương trình	1493
48.2.1 Thu thập dữ liệu	1493
48.2.2 Xây dựng mô hình sinh thơ	1504

48.3 Câu hỏi trắc nghiệm	1514
48.4 Phụ lục	1516
XI Module 11: Những mô hình tạo sinh	1517
49 Style Transfer	1518
49.1 Style Transfer	1518
49.2 Bài Tập và Gợi Ý	1522
49.3 Trắc Nghiệm	1537
49.4 Phụ Lục	1542
50 GAN và DCGAN	1544
50.1 Giới thiệu	1544
50.2 Text To Image Synthesis using DCGAN-BERTs	1546
50.3 Câu hỏi trắc nghiệm	1559
50.4 Phụ lục	1562
51 Project 1: Xây dựng hệ thống sinh nhạc dùng VAE	1563
51.1 Giới thiệu	1563
51.1.1 Project pipeline	1564
51.1.2 Tổng quan về VAE	1566
51.2 Cài đặt chương trình	1571
51.2.1 Thu thập bộ dữ liệu	1571
51.2.2 Xây dựng mô hình	1581
51.3 Câu hỏi trắc nghiệm	1598
51.4 Phụ lục	1601
52 Diffusion Models	1602
52.1 Giới thiệu	1602
52.2 Image Inpainting using Denoising Diffusion Probabilistic Model	1604
52.3 Câu hỏi trắc nghiệm	1619

52.4 Phụ lục	1622
53 Flow Matching	1623
53.1 Flow Matching	1623
53.2 Conditional Flow Matching	1627
53.3 Image Inpainting using Conditional Flow Matching	1629
53.4 Câu hỏi trắc nghiệm	1635
53.5 Phụ lục	1637
54 Project 2: Áp dụng Stable Diffusion cho bài toán tô màu ảnh	1638
54.1 Giới thiệu	1638
54.1.1 Mô tả bài toán	1639
54.1.2 Project pipeline	1639
54.1.3 Tổng quan về Stable Diffusion Model	1642
54.2 Cài đặt chương trình	1647
54.2.1 Thu thập bộ dữ liệu	1647
54.2.2 Xây dựng mô hình Stable Diffusion	1660
54.2.3 Triển khai mô hình	1679
54.3 Câu hỏi trắc nghiệm	1681
54.4 Phụ lục	1684
55 Project 3: Sinh ảnh từ gợi ý (text) dùng Flow Matching	1685
55.1 Giới thiệu	1685
55.2 Text-to-Image using Conditional Flow Matching	1688
55.3 Text-Guided Image Generation using Conditional Flow Matching	1694
55.4 Câu hỏi trắc nghiệm	1701
55.5 Phụ lục	1703
XII Module 12: LLMs và VLMs	1704
56 Pretraining GPT	1705

56.1	Giới thiệu	1705
56.2	Câu hỏi trắc nghiệm	1708
57	RAG, Prompting và LangChain	1710
57.1	Giới thiệu	1710
57.2	Cài đặt chương trình	1712
57.2.1	Tổ chức thư mục code	1712
57.2.2	Cập nhật file requirements.txt	1713
57.2.3	Cập nhật file data_source/generative_ai/download-.py	1714
57.2.4	Cập nhật file src/base/llm_model.py	1716
57.2.5	Cập nhật file src/rag/file_loader.py	1717
57.2.6	Cập nhật file src/rag/vectorstore.py	1719
57.2.7	Cập nhật file src/rag/offline_rag.py	1721
57.2.8	Cập nhật file src/rag/utils.py	1722
57.2.9	Cập nhật file src/rag/main.py	1722
57.2.10	Cập nhật file src/app.py	1723
57.3	Câu hỏi trắc nghiệm	1727
57.4	Phụ lục	1731
58	Instruction Tuning	1732
58.1	Giới thiệu	1732
58.2	Cài đặt chương trình	1735
58.2.1	Cài đặt và import các thư viện cần thiết	1735
58.2.2	Tải pre-trained LLM	1736
58.2.3	Cài đặt QLoRA	1737
58.2.4	Chuẩn bị bộ dữ liệu huấn luyện	1738
58.2.5	Thực hiện instruction tuning LLM	1743
58.2.6	Save Checkpoints	1745
58.2.7	Run Inference	1746
58.3	Câu hỏi trắc nghiệm	1749

58.4 Phụ lục	1752
59 Project 1: Áp dụng PEFT cho bài toán hỏi đáp các câu hỏi trắc nghiệm	1753
59.1 Giới thiệu	1753
59.2 PEFT for Multiple-choice QA	1758
59.3 Câu hỏi trắc nghiệm	1767
59.4 Phụ lục	1770
60 LLM Reasoning	1771
60.1 Giới thiệu	1771
60.2 Cài đặt chương trình	1775
60.2.1 Cài đặt và import các thư viện cần thiết	1775
60.2.2 Load mô hình lớn và setup cài đặt LoRA	1775
60.2.3 Tải bộ dữ liệu	1776
60.2.4 Cài đặt format prompt cho reasoning	1777
60.2.5 Định nghĩa các hàm reward cho học tăng cường	1779
60.2.6 Huấn luyện mô hình	1782
60.2.7 Lưu trữ và inference	1784
60.3 Câu hỏi trắc nghiệm	1787
60.4 Phụ lục	1790
61 Preference Tuning dùng RLHF và DPO	1791
61.1 Giới thiệu	1791
61.2 DPO for Vietnamese Custom Preference Dataset	1794
61.3 Câu hỏi trắc nghiệm	1802
61.4 Phụ lục	1805
62 ReAct Agent	1806
62.1 Giới thiệu	1806
62.2 ReAct Agent	1808
62.3 Plan-and-Execute Agent	1817

62.4 Câu hỏi trắc nghiệm	1826
62.5 Phụ lục	1828
63 Project 2: Agentic AI Using Vision Language Model	1829
63.1 Giới thiệu	1829
63.2 Visual Agentic AI	1832
63.3 Câu hỏi trắc nghiệm	1857
63.4 Phụ lục	1859
64 Giao thức MCP	1860
64.1 Giới thiệu	1860
64.2 Model Context Protocol	1862
64.2.1 Tổng quan	1862
64.2.2 Kiến trúc của MCP	1873
64.2.3 MCP Client Features	1879
64.2.4 MCP Server Features	1888
64.2.5 Các cơ Chế Truyền Tải trong MCP	1914
64.2.6 Chạy Library MCP Server và thiết kế MCP Client để kiểm tra	1923
64.3 Cài đặt Personal Database MCP Server	1933
64.3.1 Tạo thư mục project và cài thư viện	1933
64.3.2 Chuẩn bị dữ liệu và văn bản	1934
64.3.3 Xây dựng Vector Store	1937
64.3.4 Xây dựng Retriever	1943
64.3.5 Xây dựng MCP Server	1950
64.3.6 Chạy MCP Server và tích hợp vào Claude Desktop	1966
64.3.7 Sử dụng Prompts, Resources và Tools trên Claude Desktop	1970
64.4 Câu hỏi trắc nghiệm	1974
64.4.1 Câu hỏi	1974
64.4.2 Đáp án và giải thích	1976
65 Giao thức A2A	1978

65.1	Giới thiệu	1978
65.2	Các khái niệm cốt lõi của A2A	1981
65.2.1	Agent Card	1981
65.2.2	Giao thức JSON-RPC 2.0	1984
65.2.3	A2A Server	1985
65.2.4	A2A Client	1987
65.2.5	Task	1988
65.2.6	Message	1990
65.2.7	Part	1990
65.2.8	Artifact	1991
65.2.9	Mô hình tương tác giữa 2 Agent	1991
65.2.10	Mô hình phối hợp nhiều Agent	1993
65.3	Xây dựng mô hình áp dụng A2A và MCP	1995
65.3.1	Tổng quan về mô hình	1995
65.3.2	Xây dựng các A2A Server	1996
65.3.3	Xây dựng Host Agent	2012
65.3.4	Cloning GitHub Repo và chạy Demo	2024

Phần I

Module 1: Toán cơ bản và lập trình Python

Chương 1

Python cơ bản và các hàm activation

1.1 Giới thiệu

1. Giới thiệu về Python

Python là một ngôn ngữ lập trình cấp cao, thông dịch, đa mục đích và có cú pháp rõ ràng, dễ học. Nó hỗ trợ nhiều mô hình lập trình như hướng thủ tục, hướng đối tượng và hàm. Python được sử dụng rộng rãi trong khoa học dữ liệu, trí tuệ nhân tạo, phát triển web, tự động hóa và nhiều lĩnh vực khác.

2. Biến và Kiểu Dữ Liệu

Biến là vùng nhớ được đặt tên dùng để lưu trữ giá trị. Python là ngôn ngữ động nên không cần khai báo kiểu trước.

Một số kiểu dữ liệu cơ bản:

- **int**: số nguyên (*e.g.* 1, -5)
- **float**: số thực (*e.g.* 3.14, -0.5)
- **str**: chuỗi ký tự (*e.g.* "hello")
- **bool**: giá trị đúng/sai (True/False)
- **list, tuple, set, dict**: cấu trúc dữ liệu phức tạp

Ví dụ khai báo:

```
1 x = 10
2 name = "An"
3 flag = True
```

3. Câu lệnh điều kiện và vòng lặp

if, elif, else Cho phép kiểm tra điều kiện để điều khiển luồng chương trình:

```

1 if x > 0:
2     print("Số dương")
3 elif x == 0:
4     print("Số không")
5 else:
6     print("Số âm")

```

for Vòng lặp duyệt qua dãy số hoặc phần tử:

```

1 for i in range(5):
2     print(i)

```

while Lặp lại khi điều kiện còn đúng:

```

1 x = 0
2 while x < 5:
3     print(x)
4     x += 1

```

4. Hàm trong Python

Hàm là khối mã thực thi khi được gọi. Có thể truyền tham số và nhận giá trị trả về.

Khai báo và sử dụng hàm:

```

1 def greet(name):
2     return "Hello " + name
3
4 print(greet("An"))

```

Tham số mặc định và không giới hạn:

```

1 def say_hello(name="bạn"):
2     print("Chào", name)
3

```

```
4 def sum_all(*args):  
5     return sum(args)
```

5. Một số hàm dụng sẵn

- `type(x)`: kiểm tra kiểu
- `len(s)`: độ dài chuỗi/danh sách
- `int()`, `str()`, `float()`: chuyển đổi kiểu
- `range(start, stop)`: tạo cách dang sách số từ start đến stop-1
- `import math`, `import random`

1.2 Câu hỏi tự luận

1. Viết hàm tính độ đo F1-Score thường được sử dụng để đánh giá mô hình phân loại.

- Precision = $\frac{TP}{TP + FP}$
- Recall = $\frac{TP}{TP + FN}$
- F1-score = $2 * \frac{Precision * Recall}{Precision + Recall}$
- Input: hàm nhận 3 giá trị tp, fp, fn
- Output: trả về và in ra kết quả của Precision, Recall, và F1-score

+ giải thích đề

⇒ BT này có mục đích gì

⇒ BT này có ý nghĩa gì

Chú ý: Đề bài yêu cầu các điều kiện sau

- Phải kiểm tra giá trị nhận vào tp, fp, fn là kiểu dữ liệu int, nếu là kiểu dữ liệu khác khác thì in ra thông báo cho người dùng ví dụ kiểm tra fn là float, in ra thông báo "fn must be int" và thoát hàm hoặc dừng chương trình.
- Yêu cầu tp, fp, fn phải đều lớn hơn hoặc bằng 0, nếu khác thì in "tp and fp and fn must be greater than or equal zero" và thoát hàm hoặc dừng chương trình. Cần kiểm tra trường hợp phép tính chỉ 0.

```

1 # Examples
2 evaluate_f1_components(tp=2, fp=3, fn=4)
3 >>> precision is 0.4
4 recall is 0.3333333333333333
5 f1-score is 0.3636363636363636

6
7 evaluate_f1_components(tp="a", fp=3, fn=4)
8 >>> tp must be int
9
10
11 evaluate_f1_components(tp=2, fp="a", fn=4)
12 >>> fp must be int
13

```

```
14 evaluate_f1_components(tp=2, fp=3, fn="a")
15 >>> tp must be int
16
17 evaluate_f1_components(tp=2.1, fp=3, fn=0)
18 >>> tp must be int
```

- Giải thích dc F₁-score, Precision, Recall là gì?
- Ý nghĩa F₁-score trong thưc tế
- F₁-score và Accuracy có mâu thuẫn?
- Lỗi code

2. Viết hàm tính giá trị cho các hàm số kích hoạt.

- **Sigmoid Function:** Sigmoid Function, hay còn được gọi là logistic function, là một trong những hàm kích hoạt cơ bản nhất trong machine learning và neural networks. Hình dạng của nó giống như chữ "S". Sigmoid chuyển đổi mọi giá trị đầu vào thành một giá trị đầu ra nằm trong khoảng 0 và 1.

Ứng Dụng: Sigmoid Function thường được sử dụng trong các bài toán phân loại nhị phân, nơi mà mục tiêu là phân loại đầu vào thành một trong hai lớp.

Ưu Điểm:

- **Dễ hiểu và triển khai:** Do tính chất đơn giản và phổ biến, Sigmoid được triển khai trong nhiều loại mạng neuron.
- **Đầu ra nằm trong khoảng [0,1]:** Giá trị đầu ra luôn nằm trong khoảng từ 0 đến 1, giúp dễ dàng diễn giải như là xác suất.

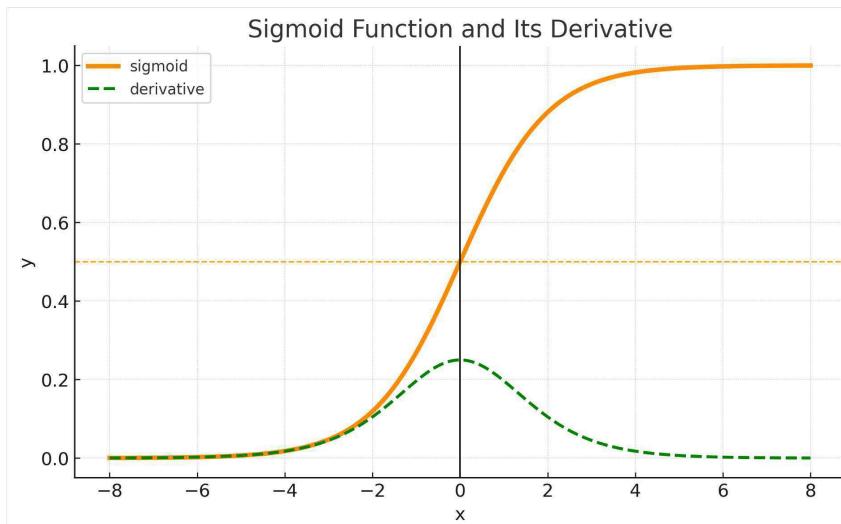
Nhược điểm:

- **Vanishing gradient problem:** Khi đầu vào có giá trị lớn hoặc nhỏ, đạo hàm của Sigmoid tiệm cận đến 0, dẫn đến vấn đề vanishing gradient, làm chậm quá trình học của mạng.
- **Tâm đối xứng không tại 0:** Tâm đối xứng không nằm tại điểm 0, điều này có thể gây ra vấn đề trong việc điều chỉnh trọng số trong quá trình học.

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

ReLU Function: ReLU, viết tắt của "Rectified Linear Unit", là một hàm kích hoạt rất phổ biến trong neural networks. Được đánh giá cao vì tính đơn giản nhưng hiệu quả, ReLU được sử dụng rộng rãi để giúp neural networks học được những đặc trưng phức tạp mà không gặp phải vấn đề biến mất gradient. ReLU hoạt động dựa trên nguyên tắc rất đơn giản: nếu đầu vào là số âm, hàm sẽ trả về giá trị 0, còn nếu đầu vào là số dương, hàm sẽ trả về chính giá trị đó.

Ứng dụng: ReLU phổ biến trong các ứng dụng như nhận dạng hình ảnh và xử lý ngôn ngữ tự nhiên, nơi ReLU giúp cải thiện tốc



Hình 1.1: Hàm Sigmoid và đạo hàm

độ học và giảm thiểu vấn đề biến mất gradient. ReLU cũng rất quan trọng trong học tăng cường và các tác vụ phân loại, cung cấp một phương pháp hiệu quả để xử lý thông tin phi tuyến tính.

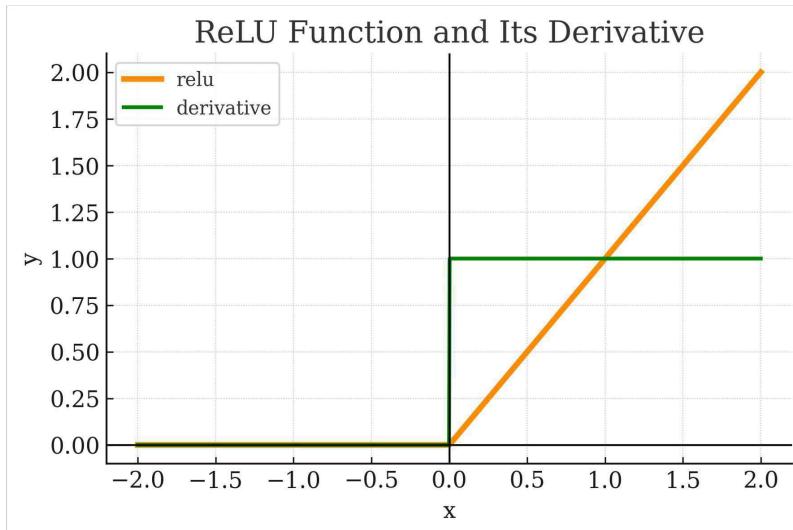
Ưu điểm:

- **Tính toán đơn giản:** Do cấu trúc đơn giản, ReLU nhanh và hiệu quả hơn trong việc tính toán so với các hàm kích hoạt phi tuyến tính khác.
- **Giảm mất mát gradient:** ReLU giúp giảm thiểu vấn đề biến mất gradient, một điểm mạnh quan trọng trong quá trình huấn luyện neural networks.

Nhược điểm:

- **Vấn đề dying ReLU:** Đôi khi neuron có thể chỉ trả về giá trị 0 cho tất cả các đầu vào, dẫn đến hiện tượng "dying ReLU", khi đó neuron trở nên không hoạt động.
- **Không đổi xứng tại zero:** Do ReLU không phải là hàm có giá trị trung bình bằng 0 nên có thể gây ra vấn đề trong quá trình tối ưu hóa mạng.

$$\text{relu}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0. \end{cases}$$



Hình 1.2: Hàm ReLU và đạo hàm

- **ELU Function:** ELU, viết tắt của Exponential Linear Unit, là một loại activation function được sử dụng trong neural network. Được đề xuất bởi Djork-Arné Clevert và các cộng sự vào năm 2015, ELU nhằm mục đích giải quyết một số vấn đề của các activation functions trước đây như ReLU. ELU giảm thiểu vấn đề vanishing gradient ở các giá trị âm, đồng thời vẫn duy trì tính phi tuyến cần thiết cho quá trình học sâu.

Ứng Dụng: ELU chủ yếu được sử dụng trong các mạng neuron sâu, đặc biệt là những nơi cần giải quyết vấn đề vanishing gradient. ELU thường được áp dụng trong các mô hình học sâu phức tạp như convolutional neural networks (CNNs) và recurrent neural networks (RNNs) để cải thiện tốc độ học và hiệu suất của mô hình.

Ưu Điểm:

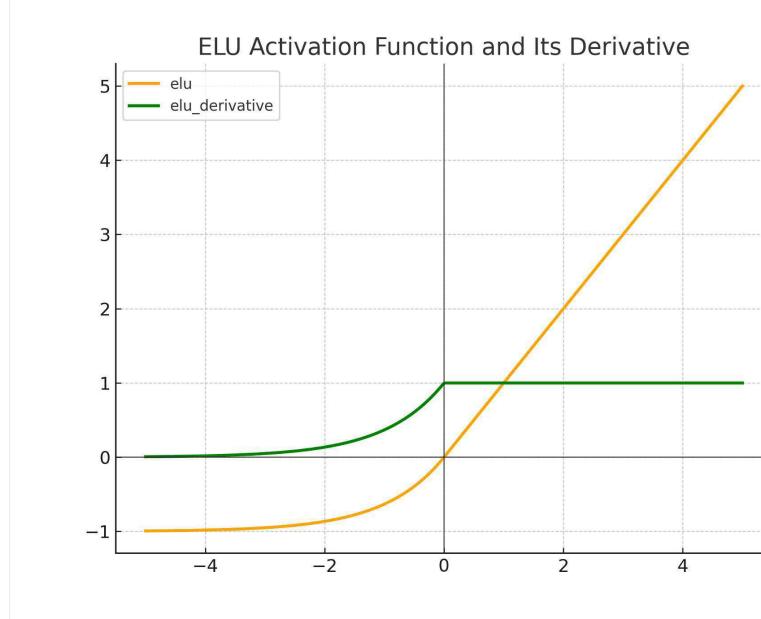
- **Hiệu suất cao:** Trong một số trường hợp, ELU cho thấy hiệu suất tốt hơn so với các activation functions khác như ReLU và Leaky ReLU, đặc biệt trong các mạng sâu.

- **Có đầu ra âm:** Việc này giúp duy trì một phân phối đầu ra cân bằng hơn, có thể cải thiện khả năng học của mô hình.

Nhược Điểm:

- **Tính toán phức tạp hơn:** Do cấu trúc phức tạp của công thức, ELU đòi hỏi nhiều chi phí tính toán hơn so với ReLU.
- **Lựa chọn α :** Việc lựa chọn giá trị của α có thể ảnh hưởng đáng kể đến hiệu suất của mô hình và nó không có một quy tắc cụ thể nào, đòi hỏi phải thử nghiệm để tìm ra giá trị phù hợp.

$$ELU(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$



Hình 1.3: Hàm ELU và đạo hàm

- Input:
 - Người dùng nhập giá trị x
 - Người dùng nhập tên **activation function** chỉ có 3 loại (**sigmoid, relu, elu**)

- Output: Kết quả $f(x)$ (x khi đi qua activation function tương ứng theo activation function name). Ví dụ **nhập $x=3$, activation_function = "relu"**. Output: trả về kết quả và in ra "relu: $f(3)=3$ "

Chú ý: Lưu ý các điều kiện sau:

- Dùng hàm **is_number** được cung cấp sẵn để kiểm tra x có hợp lệ hay không (vd: $x=10$, **is_number(x)** sẽ trả về True ngược lại là False). Nếu không hợp lệ in ra "x must be a number" và dừng chương trình.
- Kiểm tra tên hàm kích hoạt có hợp lệ hay không nằm trong 3 tên (sigmoid, relu, elu). Nếu không print "ten_function_user is not supported" (vd người dùng nhập "belu" thì in ra "belu is not supported")
- Ép kiểu dữ liệu của x sang **float**
- Thực hiện theo công thức với activation name tương ứng. In ra kết quả
- Dùng **math.e** để lấy số e
- $\alpha = 0.01$

```

1 # Given
2 def is_number(n):
3     try:
4         float(n)    # Type-casting the string to 'float'.
5         # If string is not a valid 'float',
6         # it'll raise 'ValueError' exception
7     except ValueError:
8         return False
9     return True

```

```

1 interactive_activation_function()
2 >> Input x = 1.5
3 Input activation Function (sigmoid|relu|elu): sigmoid
4 sigmoid: f(1.5) = 0.8175744761936437
5
6 interactive_activation_function()

```

```
7 >> Input x = abc
8 x must be a number
9
10 interactive_activation_function()
11 >> Input x = 1.5
12 Input activation Function (sigmoid|relu|elu): belu
13 belu is not supported
```

3. Viết hàm lựa chọn hàm mất mát và tính giá trị hàm mất mát (loss functions)

- $MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$
- $MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$
- $RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$
- **n** chính là số lượng samples (`num_samples`), với **i** là mỗi sample cụ thể. Ở đây các bạn có thể hiểu là cứ mỗi **i** thì sẽ có 1 cặp y_i là target và \hat{y}_i là predict.
- Input:
 - Người dùng nhập số lượng sample (`num_samples`) được tạo ra (chỉ nhận integer numbers)
 - Người dùng nhập loss name (MAE, MSE, RMSE-(optional)) chỉ cần MAE và MSE, bạn nào muốn làm thêm RMSE đều được.
- Output: Print ra **loss name, sample, predict, target, loss**
 - **loss name:** là loss mà người dùng chọn
 - **sample:** là thứ tự sample được tạo ra (ví dụ `num_samples=5`, thì sẽ có 5 samples và mỗi sample là sample-0, sample-1, sample-2, sample-3, sample-4)
 - **predict:** là số mà model dự đoán (chỉ cần dùng random tạo random một số trong range [0,10])
 - **target:** là số target mà momg muốn mode dự đoán đúng (chỉ cần dùng random tạo random một số trong range [0,10])
 - **loss:** là kết quả khi đưa predict và target vào hàm loss
 - **note:** ví dụ `num_sample=5` thì sẽ có 5 cặp predict và target.

Chú ý: Các bạn lưu ý

- Dùng `.isnumeric()` method để kiểm tra `num_samples` có hợp lệ hay không (vd: `x=10, num_samples.isnumeric()` sẽ trả về True ngược lại là False). Nếu không hợp lệ print "number of samples must be an integer number" và dừng chương trình.

- Dùng vòng lặp for, lặp lại num_samples lần. Mỗi lần dùng random modules tạo một con số ngẫu nhiên trong range [0.0, 10.0) cho predict và target. Sau đó đưa predict và target vào loss function và print ra kết quả mỗi lần lặp.
- Dùng random.uniform(0,10) để tạo ra một số ngẫu nhiên trong range [0,10)
- Giả sử người dùng luôn nhập đúng loss name MSE, MAE, và RMSE (đơn giản bước này để các bạn không cần kiểm tra tên hợp lệ)
- Dùng abs() để tính trị tuyệt đối ví dụ abs(-3) sẽ trả về 3
- Dùng math.sqrt() để tính căn bậc 2

```
1 cal_activation_function()
2 >> Input number of samples (integer number) which are
   generated: 3
3 Input loss name: MAE
4 loss_name: MAE, sample: 0: pred: 5.92 target: 6.96 loss: 1.05
5 loss_name: MAE, sample: 1: pred: 0.1 target: 5.74 loss: 5.64
6 loss_name: MAE, sample: 2: pred: 0.94 target: 2.1 loss: 1.16
7 final MAE: 2.6156580640656397
8
9 cal_activation_function()
10 >> Input number of samples (integer number) which are
    generated: aa
11 number of samples must be an integer number
```

4. Viết 4 hàm để ước lượng các hàm số sau.

$$\sin(x) \approx \sum_{n=0}^{\infty} (-1)^n \frac{x^{(2n+1)}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \dots$$

$$\cos(x) \approx \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \dots$$

$$\sinh(x) \approx \sum_{n=0}^{\infty} \frac{x^{(2n+1)}}{(2n+1)!} = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \frac{x^9}{9!} + \dots$$

$$\cosh(x) \approx \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \frac{x^8}{8!} + \frac{x^{10}}{10!} + \dots$$

- Input: x (số muốn tính toán) và n (số lần lặp muốn xấp xỉ)
- Output: Kết quả ước lượng hàm tương ứng với x. Ví dụ hàm $\cos(x=0)$ thì output = 1

Chú ý: Các bạn chú ý các điều kiện sau

- x là radian
- n là số nguyên dương > 0
- các bạn nên viết một hàm tính giai thừa riêng

```

1 approx_sin(x=3.14, n=10)
2 >> 0.0015926529267151343
3
4 approx_cos(x=3.14, n=10)
5 >> -0.9999987316527259
6
7 approx_sinh(x=3.14, n=10)
8 >> 11.53029203039954
9
10 approx_cosh(x=3.14, n=10)
11 >> 11.573574828234543

```

1.3 Câu hỏi trắc nghiệm

Câu hỏi 1 : Viết hàm thực hiện đánh giá mô hình phân loại bằng F1-Score. Hàm nhận vào 3 giá trị **tp**, **fp**, **fn** và trả về F1-score

- Precision = $\frac{TP}{TP + FP}$
- Recall = $\frac{TP}{TP + FN}$
- F1-score = $2 * \frac{Precision * Recall}{Precision + Recall}$

Đầu ra của chương trình sau đây là gì?

```

1 import math
2 def calc_f1_score(tp, fp, fn):
3     """
4         Tính điểm F1 Score từ các giá trị:
5             - tp: Số lượng true positives
6             - fp: Số lượng false positives
7             - fn: Số lượng false negatives
8
9         Công thức:
10            precision = tp / (tp + fp)
11            recall = tp / (tp + fn)
12            f1_score = 2 * (precision * recall) / (precision + recall)
13
14         Trả về:
15            f1_score: Điểm F1 Score dưới dạng float
16            """
17         # Your code here
18
19         # End your code
20
21 assert round(calc_f1_score(tp=2, fp=3, fn=5), 2) == 0.33
22 print(round(calc_f1_score(tp=2, fp=4, fn=5), 2))

```

- a) 0.33
- b) 0.35

- c) 0.31
- d) Raise an Error

Câu hỏi 2 : Viết hàm `is_number` nhận input có thể là string hoặc một số kiểm tra n (một số) có hợp lệ hay không (vd: n=10, `is_number(n)` sẽ trả về True ngược lại là False). Đầu ra của chương trình sau đây là gì?

```

1 import math
2 def is_number(n):
3     # Your code here
4
5     # End your code
6 assert is_number(3) == 1.0
7 assert is_number("-2a") == 0.0
8 print(is_number(1))
9 print(is_number("n"))

```

- a) n
- b)
- True
- False
- c) 1
- d) Raise an Error

Câu hỏi 3 : Đoạn code dưới đây đang thực hiện activation function nào?

```

1 x = -2.0
2 if x<=0:
3     y = 0.0
4 else:
5     y = x
6 print(y)
7 >> 0.0

```

- a) Elu
- b) Sigmoid

- c) ReLU
- d) Activation function khác

Câu hỏi 4 : Viết function thực hiện Sigmoid Function $f(x) = \frac{1}{1 + e^{-x}}$. Nhận input là x và return kết quả tương ứng trong Sigmoid Function. Đầu ra của chương trình sau đây là gì?

```

1 import math
2 def calc_sig(x):
3     """
4     Tính hàm sigmoid
5     """
6     # Your code here
7
8     # End your code
9
10 assert round(calc_sig(3), 2)==0.95
11 print(round(calc_sig(2), 2))

```

- a) 0.88
- b) 2
- c) 0.9907970779778823
- d) Raise an Error

Câu hỏi 5 : Viết function thực hiện Elu Function $f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$.

Nhận input là x và return kết quả tương ứng trong Elu Function. Đầu ra của chương trình sau đây là gì khi $\alpha = 0.01$?

```

1 import math
2 def calc_elu(x):
3     """
4     Tính hàm ELU (Exponential Linear Unit):
5     ELU(x) = x             nếu x >= 0
6             = alpha * (e^x - 1) nếu x < 0
7     """

```

```

8     # Your code here
9
10    # End your code
11
12 assert round(calc_elu(1))==1
13 print(round(calc_elu(-1), 2))

```

- a) -1
- b) -0.01
- c) -0.106321205588285576
- d) Raise an Error

Câu hỏi 6 : Viết function nhận 2 giá trị x , và tên của activation function act_name activation function chỉ có 3 loại (sigmoid, relu, elu), thực hiện tính toán activation function tương ứng với name nhận được trên giá trị của x và trả kết quả. Đầu ra của chương trình sau đây là gì?

```

1 import math
2
3 def calc_activation_func(x, act_name):
4     """
5     Tính hàm kích hoạt cho x dựa trên act_name:
6     "relu", "sigmoid", hoặc "elu".
7     """
8     # Your code here
9
10    # End your code
11 assert calc_activation_func(x = 1, act_name="relu") == 1
12 print(round(calc_activation_func(x = 3, act_name="sigmoid"), 2))

```

- a) 0.95
- b) 4
- c) 1
- d) Raise an Error

Câu hỏi 7 : Viết function tính absolute error = $|y - \hat{y}|$. Nhận input là y và

\hat{y} , return về kết quả absolute error tương ứng. Đầu ra của chương trình sau đây là gì?

```

1 def calc_ae(y, y_hat):
2     """
3         Tính sai số tuyệt đối (Absolute Error)
4         giữa giá trị thực tế và giá trị dự đoán.
5
6         Tham số:
7             y (float hoặc int): Giá trị thực tế (ground truth).
8             y_hat (float hoặc int): Giá trị dự đoán.
9
10        Trả về:
11            float: Giá trị tuyệt đối của hiệu giữa y và y_hat.
12        """
13        # Your code here
14
15        # End your code
16
17 y = 1
18 y_hat = 6
19 assert calc_ae(y, y_hat)==5
20 y = 2
21 y_hat = 9
22 print(calc_ae(y, y_hat))

```

- a) 7
- b) 8
- c) 9
- d) Raise an Error

Câu hỏi 8 : Viết function tính squared error = $(y - \hat{y})^2$. Nhận input là y và \hat{y} , return về kết quả squared error tương ứng. Đầu ra của chương trình sau đây là gì?

```

1 def calc_se(y, y_hat):
2     """
3         Tính sai số bình phương (Squared Error)
4         giữa giá trị thực tế và giá trị dự đoán.

```

```

5      Tham số:
6      y (float hoặc int): Giá trị thực tế (ground truth).
7      y_hat (float hoặc int): Giá trị dự đoán.
8
9      Trả về:
10     float: Bình phương của hiệu giữa y và y_hat.
11
12
13     # Your code here
14
15     # End your code
16 y = 4
17 y_hat = 2
18 assert calc_se(y, y_hat) == 4
19 print(calc_se(2, 1))

```

- a) 1
- b) 2
- c) 3
- d) Raise an Error

Câu hỏi 9 : Dựa vào công thức xấp xỉ sin và điều kiện được giới thiệu trong phần tự luận. Viết function xấp xỉ sin khi nhận x là giá trị muốn tính $\sin(x)$ và n là số lần lặp muôn xấp xỉ. Return về kết quả $\sin(x)$ với bậc xấp xỉ tương ứng. Đầu ra của chương trình sau đây là gì?

```

1 def approx_sin(x, n):
2     """
3         Approximate the sine of x using the Taylor series expansion.
4
5     Parameters:
6         x (float): The input angle in radians.
7         n (int): Number of terms in the Taylor series expansion.
8
9     Returns:
10        float: Approximate value of sin(x) using n+1 terms.
11
12    # Your code here
13

```

```

14     # End your code
15
16 assert round(approx_sin(x=1, n=10), 4)==0.8415
17 print(round(approx_sin(x=3.14, n=10), 4))

```

- a) 0.0016
- b) 0.0017
- c) 0.0018
- d) Raise an Error

Câu hỏi 10 : Dựa vào công thức xấp xỉ cos và điều kiện được giới thiệu trong phần tự luận. Viết function xấp xỉ cos khi nhận x là giá trị muốn tính $\cos(x)$ và n là số lần lặp muốn xấp xỉ. Trả về kết quả $\cos(x)$ với bậc xấp xỉ tương ứng. Đầu ra của chương trình sau đây là gì?

```

1 def approx_cos(x, n):
2     """
3     Approximate the cosine of x using the Taylor series expansion.
4     Parameters:
5         x (float): The input angle in radians.
6         n (int): Number of terms in the Taylor series expansion.
7     Returns:
8         float: Approximate value of cos(x) using n+1 terms.
9     """
10    # Your code here
11
12    # End your code
13
14 assert round(approx_cos(x=1, n=10), 2)==0.54
15 print(round(approx_cos(x=3.14, n=10), 2))

```

- a) 2
- b) 1
- c) -1.0
- d) Raise an Error

Câu hỏi 11 : Dựa vào công thức xấp xỉ sinh và điều kiện được giới thiệu trong phần tự luận. Viết function xấp xỉ sinh khi nhận x là giá trị muốn tính $\sinh(x)$ và n là số lần lặp muốn xấp xỉ. Trả về kết quả $\sinh(x)$ với bậc xấp xỉ tương ứng. Đầu ra của chương trình sau đây là gì?

```

1 def approx_sinh(x, n):
2     """
3     Approximate the sinh of x using the Taylor series expansion.
4     Parameters:
5         x (float): The input angle in radians.
6         n (int): Number of terms in the Taylor series expansion.
7     Returns:
8         float: Approximate value of sinh(x) using n+1 terms.
9         """
10    # Your code here
11
12    # End your code
13
14 assert round(approx_sinh(x=1, n=10), 2)==1.18
15 print(round(approx_sinh(x=3.14, n=10), 2))

```

- a) 11.53
- b) 12.53
- c) 13.53
- d) Raise an Error

Câu hỏi 12 : Dựa vào công thức xấp xỉ cosh và điều kiện được giới thiệu trong phần tự luận. Viết function xấp xỉ cosh khi nhận x là giá trị muốn tính $\cosh(x)$ và n là số lần lặp muốn xấp xỉ. Trả về kết quả $\cosh(x)$ với bậc xấp xỉ tương ứng. Đầu ra của chương trình sau đây là gì?

```

1 def approx_cosh(x, n):
2     """
3     Approximate the hyperbolic cosine of x using the Taylor series.
4     Parameters:
5         x (float): The input value.
6         n (int): Number of terms in the Taylor series expansion.
7     Returns:

```

```
8     float: Approximate value of cosh(x) using n+1 terms.  
9     """  
10    # Your code here  
11  
12    # End your code  
13 assert round(approx_cosh(x=1, n=10), 2)==1.54  
14 print(round(approx_cosh(x=3.14, n=10), 2))
```

- a) 11.57
- b) 11.58
- c) 11.59
- d) Raise an Error

1.4 Phụ lục

1. **Hint:** Dựa vào file tải về [Exercise: Activation Functions](#) để hoàn thiện các đoạn code.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng khi hết deadline phần project, admin mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

Basic Python – Exercise Rubric		
Câu	Kiến Thức	Đánh Giá
1	- Hiểu được cách sử dụng Python để tính toán F1-Score và giới thiệu cơ bản đánh giá mô hình bằng F1-Score.	- Có khả năng viết function và xử lý lỗi khi code Python tính F1-Score, thể hiện sự hiểu biết về thực hiện code theo một công thức cho trước.
2	- Nắm được cách thức cài đặt các hàm kích hoạt như sigmoid, relu và elu trong Python.	- Thể hiện khả năng thực hiện các hàm này trong môi trường lập trình Python, phù hợp cho các ứng dụng AI.
3	- Hiểu rõ cách thực hiện các hàm regression loss phổ biến trong Python để tính toán loss cho các bài toán regression.	- Thể hiện khả năng cài đặt các hàm loss đa dạng trong môi trường lập trình Python, phù hợp để tính toán loss trong các bài toán regression
4	- Tìm hiểu về các phương pháp lập trình Python để ước lượng giá trị hàm số.	- Có khả năng lập trình Python để mô phỏng các hàm số, áp dụng hiệu quả trong các bài toán tính toán.
5	- Hiểu được cách xây dựng và sử dụng các chỉ số đánh giá mô hình nâng cao trong Python.	Thực hiện thành thạo các tính toán error phức tạp trong việc sử dụng Python để cải thiện hiệu suất mô hình.

- *Hết* -

Chương 2

Cấu trúc dữ liệu và bài toán word suggestion

2.1 Giới thiệu

Trong ngôn ngữ lập trình Python, có bốn kiểu cấu trúc dữ liệu cơ bản thường được sử dụng để lưu trữ và thao tác với tập hợp dữ liệu: **List**, **Tuple**, **Set**, và **Dictionary**. Mỗi loại có đặc điểm và cách sử dụng riêng, phù hợp với từng ngữ cảnh cụ thể trong lập trình.

List (Danh sách)

List là một dãy các phần tử có thứ tự và có thể thay đổi (mutable). Đây là một trong những kiểu dữ liệu linh hoạt và phổ biến nhất trong Python.

- Được khai báo bằng dấu ngoặc vuông [].
- Các phần tử có thể thuộc nhiều kiểu dữ liệu khác nhau.
- Cho phép các phần tử trùng lặp.
- Hỗ trợ các thao tác như thêm, xoá, cập nhật, duyệt lặp.

Ví dụ sử dụng list:

```
1 fruits = ["apple", "banana", "cherry"]
2 print(fruits[1])      # In ra phần tử thứ 2: banana
3 fruits.append("mango") # Thêm phần tử vào cuối
4 fruits[0] = "grape"   # Thay đổi phần tử đầu tiên
5 print(fruits)
```

Tuple (Bộ giá trị bất biến)

Tuple là một dạng dãy phần tử giống như list, nhưng không thể thay đổi sau khi đã khởi tạo. Điều này giúp tuple hoạt động hiệu quả hơn và an toàn hơn trong một số tình huống.

- Được khai báo bằng dấu ngoặc tròn () .
- Không thể thay đổi nội dung sau khi tạo.
- Có thể chứa nhiều kiểu dữ liệu và trùng lặp phần tử.
- Thường dùng cho dữ liệu cố định như tọa độ, cấu hình.

Ví dụ sử dụng tuple:

```

1 coordinates = (10, 20)
2 print(coordinates[0])    # In ra: 10
3 # coordinates[0] = 30   # Lỗi: không thể gán lại giá trị

```

Set (Tập hợp)

Set là một tập các phần tử không có thứ tự và không chứa phần tử trùng lặp. Đây là lựa chọn tốt khi muốn lưu trữ các giá trị duy nhất.

- Khai báo bằng dấu ngoặc nhọn {} hoặc hàm set().
- Tự động loại bỏ phần tử trùng.
- Không thể truy cập bằng chỉ số.
- Hỗ trợ các phép toán tập hợp: hợp, giao, hiệu,...

Ví dụ sử dụng set:

```

1 numbers = {1, 2, 3, 3, 4}
2 print(numbers)      # Kết quả: {1, 2, 3, 4}
3 numbers.add(5)     # Thêm phần tử mới
4 numbers.remove(2)  # Xoá phần tử

```

Dictionary (Từ điển)

Dictionary là một cấu trúc ánh xạ giữa các **key** và **value**. Đây là kiểu dữ liệu rất mạnh trong Python khi cần truy cập nhanh theo khoá định danh.

- Khai báo bằng dấu ngoặc nhọn {} với các cặp key: value.
- Key là duy nhất và không thể thay đổi (immutable).
- Value có thể là bất kỳ kiểu dữ liệu nào.
- Hỗ trợ thêm, cập nhật, xoá, truy cập theo khoá.

Ví dụ sử dụng dictionary:

```

1 person = {
2     "name": "Alice",
3     "age": 25,
4     "city": "Hanoi"
5 }
6 print(person["name"])      # Truy cập theo khoá
7 person["age"] = 26          # Cập nhật giá trị
8 person["email"] = "a@gmail.com" # Thêm mới

```

Tóm tắt:

- **List:** Có thứ tự, thay đổi được, cho phép trùng lặp.
- **Tuple:** Có thứ tự, không thay đổi, cho phép trùng lặp.
- **Set:** Không thứ tự, không trùng lặp, không chỉ mục.
- **Dictionary:** Ánh xạ khóa – giá trị, key duy nhất.

Các cấu trúc dữ liệu này là nền tảng quan trọng giúp lập trình hiệu quả với dữ liệu trong Python. Việc hiểu rõ đặc điểm và cách sử dụng của từng kiểu sẽ giúp bạn thiết kế thuật toán tối ưu và dễ bảo trì hơn.

Nội dung bài tập sẽ tập trung vào làm việc với một số cấu trúc dữ liệu và giải thuật đơn giản.

2.2 Câu hỏi tự luận

2.2.1 Getting Max Over Kernel

Tìm giá trị lớn nhất trong cửa sổ trượt trên danh sách cho trước

Bài toán Cho một list các số nguyên num_list và một sliding window có kích thước size k di chuyển từ trái sang phải. Mỗi lần dịch chuyển 1 vị trí sang phải có thể nhìn thấy được k số trong num_list và tìm số lớn nhất trong k số này sau mỗi lần trượt k phải lớn hơn hoặc bằng 1.

- Input: num_list = [3, 4, 5, 1, -44, 5, 10, 12, 33, 1] với k=3
- Output: [5, 5, 5, 10, 12, 33, 33]
- Ví dụ:

```
[3, 4, 5], 1, -44, 5, 10, 12, 33, 1 => max 5
3, [4, 5, 1], -44, 5, 10, 12, 33, 1 => max 5
3, 4, [5, 1, -44], 5, 10, 12, 33, 1 => max 5
3, 4, 5, [1, -44, 5], 10, 12, 33, 1 => max 5
3, 4, 5, 1, [-44, 5, 10], 12, 33, 1 => max 10
3, 4, 5, 1, -44, [5, 10, 12], 33, 1 => max 12
3, 4, 5, 1, -44, 5, [10, 12, 33], 1 => max 33
3, 4, 5, 1, -44, 5, 10, [12, 33, 1] => max 33
```

2.2.2 Character Counting

Đếm tần suất xuất hiện của các ký tự

Bài toán Viết hàm trả về một dictionary đếm số lượng chữ xuất hiện trong một từ, với key là chữ cái và value là số lần xuất hiện

- **Input:** một từ
- **Output:** dictionary đếm số lần các chữ xuất hiện
- **Note:** Giả sử các từ nhập vào đều có các chữ cái thuộc [a-z] hoặc [A-Z]

```

1 string = "Happiness"
2 count_character(string)
3 >> {"H": 1, "a": 1, "e": 1, "i": 1, "n": 1, "p": 2, "s": 2}
4
5 string = "smiles"
6 count_character(string)
7 >> {"e": 1, "i": 1, "l": 1, "m": 1, "s": 2}

```

2.2.3 Word Counting

Đếm tần suất xuất hiện của các từ trong đoạn văn bản

Bài toán Viết hàm đọc các câu trong một file txt, đếm số lượng các từ xuất hiện và trả về một dictionary với key là từ và value là số lần từ đó xuất hiện.

- **Input:** Đường dẫn đến file txt
- **Output:** dictionary đếm số lần các từ xuất hiện
- **Note:**
 - Giả sử các từ trong file txt đều có các chữ cái thuộc [a-z] hoặc [A-Z]
 - Không cần các thao tác xử lý string phức tạp **nhưng cần xử lý các từ đều là viết thường**
 - Các bạn dùng lệnh này để download
!gdown <https://drive.google.com/uc?id=1IBScGdW2xlNsc9v5zSAya548kNgiOrko>

```

1 # Tải file từ Google Drive
2 !gdown https://drive.google.com/uc?id=
           1IBScGdW2xlNsc9v5zSAya548kNgiOrko
3
4 # Đường dẫn file
5 file_path = "./P1_data.txt"
6
7 # Gọi hàm xử lý
8 count_word(file_path)

```

```
9  
10 >> {  
11     "a": 7,  
12     "again": 1,  
13     "and": 1,  
14     "are": 1,  
15     "at": 1,  
16     "be": 1,  
17     "become": 2,  
18     ...  
19 }
```

2.2.4 Levenshtein Distance

Khoảng cách chỉnh sửa văn bản Levenshtein

Bài toán: Viết chương trình tính khoảng cách chỉnh sửa tối thiểu Levenshtein. Khoảng cách Levenshtein thể hiện khoảng cách khác biệt giữa 2 chuỗi ký tự. Khoảng cách Levenshtein giữa chuỗi S và chuỗi T là số bước ít nhất biến chuỗi S thành chuỗi T thông qua 3 phép biến đổi là:

- Xoá một ký tự
- Thêm một ký tự
- Thay thế ký tự này bằng ký tự khác

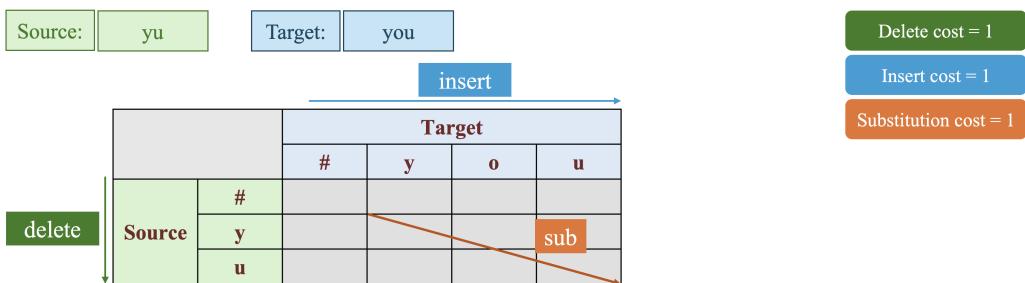
Khoảng cách này được sử dụng trong việc tính toán sự giống và khác nhau giữa 2 chuỗi, như chương trình kiểm tra lỗi chính tả của winword spellchecker. Ví dụ: Khoảng cách Levenshtein giữa 2 chuỗi “kitten” và “sitting” là 3, vì phải dùng ít nhất 3 lần biến đổi. Trong đó:

- kitten -> sitten (thay “k” bằng “s”)
- sitten -> sittin (thay “e” bằng “i”)
- sittin -> sitting (thêm ký tự “g”)

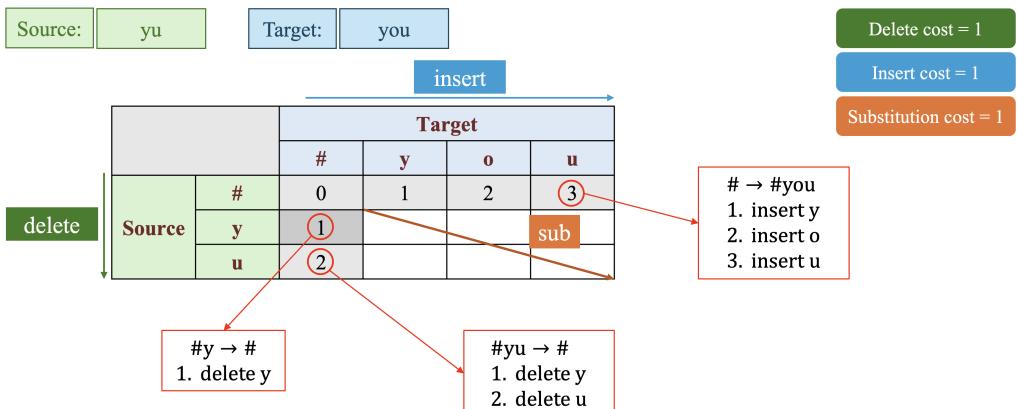
Để hiểu rõ về thuật toán, chúng ta lấy ví dụ, khoảng cách cần tính giữa hai từ source: “yu” và target: “you”. Chi phí thực hiện cho các phép biến đổi bao gồm: xoá một ký tự, thêm một ký tự và thay thế ký tự này thành ký tự khác đều là 1 (Nếu 2 ký tự giống nhau thì chi phí thực hiện là 0).

Các bước thực hiện như sau:

- Bước 1: Xây dựng ma trận lưu trữ có số hàng là M và số cột là N. Trong đó M là số lượng các ký tự trong từ source + 1, N là số lượng các ký tự trong từ target + 1. Vì vậy với ví dụ “yu” và “you”, ta có ma trận được biểu diễn như hình 1. Ký hiệu “#” đại diện cho chuỗi rỗng. Gọi là ma trận D.

Hình 2.1: Khởi tạo ma trận D .

- Bước 2: Hoàn thiện hàng và cột đầu tiên. Với hàng đầu tiên, các giá trị đại diện cho chuỗi bắt đầu là chuỗi “#” và phép biến đổi là thêm (insert) từ chuỗi “#” thành “#”, “#y”, “#yo”, “#you” lần lượt là 0, 1, 2, 3 tương ứng với ô $D[0, 0], D[0, 1], D[0, 2], D[0, 3]$. Với cột đầu tiên, các giá trị đại diện cho chuỗi “#”, “#y”, “#yu” và phép biến đổi là xoá (delete) để thu được chuỗi “#” lần lượt là: 0, 1, 2 tương ứng với ô $D[0, 0], D[1, 0], D[2, 0]$. Ta được hình 2.



Hình 2.2: Số phép biến đổi cho hàng đầu tiên (thêm) và cột đầu tiên (xoá).

- Bước 3. Tính toán các giá trị với các ô còn lại trong ma trận. Bắt đầu từ $D[1, 1]$ được tính dựa vào 3 ô phía trước là $D[0, 1], D[1, 0], D[0, 0]$

nhiều sau:

$$D[1, 1] = \min \begin{cases} D[0, 1] + \text{del_cost}(\text{source}[1]) = 1 + 1 = 2 \\ D[1, 0] + \text{ins_cost}(\text{target}[1]) = 1 + 1 = 2 \\ D[0, 0] + \text{sub_cost}(\text{source}[1], \text{target}[1]) = 0 + 0 = 0 \end{cases} \quad (2.1)$$

Vì vậy $D[1, 1] = 0$ ta được ma trận D như sau:

Source:	yu	Target:	you		
		Target			
		#	y	o	u
Source	#	0	1	2	3
	y	1	0		
	u	2			

Hình 2.3: Giá trị tại $D[1, 1]$.

Tiếp theo chúng ta tính $D[2, 1], D[1, 2]$:

$$D[2, 1] = \min \begin{cases} D[1, 1] + \text{del_cost}(\text{source}[2]) = 0 + 1 = 1 \\ D[2, 0] + \text{ins_cost}(\text{target}[1]) = 2 + 1 = 3 \\ D[1, 0] + \text{sub_cost}(\text{source}[2], \text{target}[1]) = 1 + 1 = 2 \end{cases} \quad (2.2)$$

$$D[1, 2] = \min \begin{cases} D[0, 2] + \text{del_cost}(\text{source}[1]) = 2 + 1 = 3 \\ D[1, 1] + \text{ins_cost}(\text{target}[2]) = 0 + 1 = 1 \\ D[0, 1] + \text{sub_cost}(\text{source}[1], \text{target}[2]) = 1 + 1 = 2 \end{cases} \quad (2.3)$$

Vì vậy $D[2, 1] = 1, D[1, 2] = 1$ ta được ma trận D như sau:

Cuối cùng, chúng ta tính $D[1, 3], D[2, 2], D[2, 3]$:

$$D[1, 3] = \min \begin{cases} D[0, 3] + \text{del_cost}(\text{source}[1]) = 3 + 1 = 4 \\ D[1, 2] + \text{ins_cost}(\text{target}[3]) = 1 + 1 = 2 \\ D[0, 2] + \text{sub_cost}(\text{source}[1], \text{target}[3]) = 2 + 1 = 3 \end{cases} \quad (2.4)$$

Source:	yu	Target:	you	
		insert		
		Target		
		#	y	o
		0	1	2
Source	#	0	1	3
	y	1	0	1
	u	2	1	

Delete cost = 1
Insert cost = 1
Substitution cost = 1

Hình 2.4: Giá trị tại $D[2, 1], D[1, 2]$.

Source:	yu	Target:	you	
		insert		
		Target		
		#	y	o
		0	1	2
Source	#	0	1	3
	y	1	0	1
	u	2	1	1

Delete cost = 1
Insert cost = 1
Substitution cost = 1

Hình 2.5: Giá trị tại $D[1, 3], D[2, 2], D[2, 3]$.

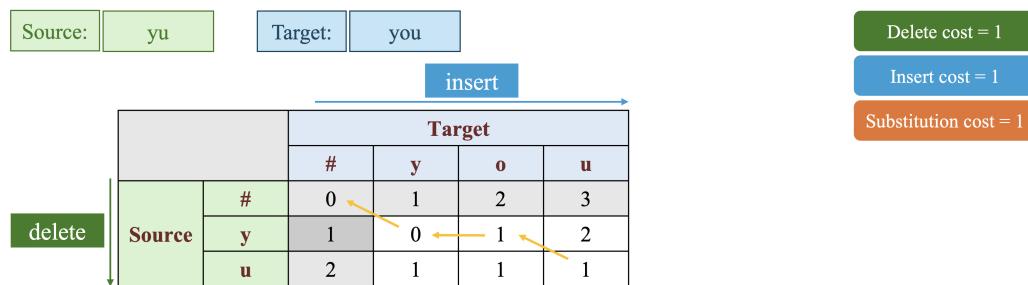
$$D[2, 2] = \min \begin{cases} D[1, 2] + \text{del_cost}(\text{source}[2]) = 1 + 1 = 2 \\ D[2, 1] + \text{ins_cost}(\text{target}[2]) = 1 + 1 = 2 \\ D[1, 1] + \text{sub_cost}(\text{source}[2], \text{target}[2]) = 0 + 1 = 1 \end{cases} \quad (2.5)$$

$$D[2, 3] = \min \begin{cases} D[1, 3] + \text{del_cost}(\text{source}[2]) = 2 + 1 = 3 \\ D[2, 2] + \text{ins_cost}(\text{target}[3]) = 1 + 1 = 2 \\ D[1, 2] + \text{sub_cost}(\text{source}[2], \text{target}[3]) = 1 + 0 = 1 \end{cases} \quad (2.6)$$

Vì vậy $D[1, 3] = 2, D[2, 2] = 1, D[2, 3] = 1$ ta được ma trận như sau:

- Bước 4: Sau khi hoàn thành ma trận, chúng ta đi tìm đường đi từ ô cuối cùng $D[2, 3]$ có giá trị là 1. Vì vậy khoảng cách chỉnh sửa từ source: “yu” sang thành target: “you” là 1. Đầu tiên ký tự “y” giữ nguyên sau đó thực hiện 1 phép thêm ký tự “o” vào sau ký tự “y” và cuối cùng ký

tự “u” được giữ nguyên. Minh họa các bước quay lui để tìm đường đi ngắn nhất tương ứng mũi tên vàng trong hình sau:



Hình 2.6: Quay lui, tìm các bước thực hiện chỉnh sửa từ source “yu” sang target: “you”.

2.3 Câu hỏi trắc nghiệm

Phản trắc nghiệm dựa trên các nội dung của câu hỏi tự luận, vì vậy học viên cần xem kỹ các yêu cầu mô tả, hoặc code ví dụ trong phần tư luận 2.2 để hoàn thiện tốt nhất.

1. Hoàn thành chương trình sau với mô tả bài toán tìm giá trị lớn nhất trong cửa sổ trượt 2.2.1 và cho biết đầu ra của chương trình dưới đây là gì?

```
1 def max_kernel(num_list, k):
2     """
3         Trả về danh sách các giá trị lớn nhất trong mỗi cửa sổ
4             con (window) kích thước k
5             chạy trượt trên danh sách num_list.
6
7     Parameters:
8         num_list (list of numbers): Danh sách các số đầu vào.
9         k (int): Kích thước của cửa sổ trượt.
10
11    Returns:
12        list: Danh sách các giá trị lớn nhất trong từng cửa sổ
13            ở con liên tiếp.
14        """
15
16    result = []
17    # Your Code Here
18
19    # End Code Here
20
21    return result
22
23
24
25 assert max_kernel([3, 4, 5, 1, -44], 3) == [5, 5, 5]
26 num_list = [3, 4, 5, 1, -44, 5, 10, 12, 33, 1]
27 k = 3
28 print(max_kernel(num_list, k))
```

- (a) [5, 5, 5, 5, 10, 12, 33, 33]
 (b) [2, 5, 3, 4, 1, 10, 3, 3]
 (c) [0, 9, 5, 1, 0, 12, 3, 33]

- (d) Raise an Error
2. Hoàn thành chương trình sau với mô tả bài toán đếm tần suất xuất hiện các ký tự 2.2.2 và cho biết đầu ra của chương trình dưới đây là gì?

```

1 def count_character(word):
2     """
3         Đếm số lần xuất hiện của từng ký tự trong chuỗi đầu vào.
4
5     Parameters:
6         word (str): Chuỗi ký tự cần thống kê.
7
8     Returns:
9         dict: Từ điển với mỗi ký tự là một khóa, giá trị là số lần xuất hiện.
10    """
11    character_statistic = {}
12
13    # Your Code Here
14
15    # End Code Here
16    return character_statistic
17
18 assert count_character("Baby") == {"B": 1, "a": 1, "b": 1, "y": 1}
19 print(count_character("smiles"))
20

```

- (a) "s": 2, "m": 1, "i": 1, "l": 1, "e": 1
 (b) "s": 0, "m": 1, "i": 1, "l": 1, "e": 8
 (c) "s": 4, "m": 1, "i": 2, "l": 1, "e": 1
 (d) Raise an Error
3. Hoàn thành chương trình sau với mô tả bài toán đếm tần suất xuất hiện các từ 2.2.3 và cho biết đầu ra của chương trình dưới đây là gì?

```

1 !gdown https://drive.google.com/uc?id=
  1IBScGdW2xlNsc9v5zSAYa548kNgiOrko

```

```

2
3     def count_word(file_path):
4         """
5             Đếm số lần xuất hiện của từng từ trong văn bản đầu vào
6             sau khi tiền xử lý.
7
8             Quá trình tiền xử lý gồm:
9             - Chuyển văn bản thành chữ thường
10            - Loại bỏ dấu chấm(.) và dấu phẩy(,)
11            - Tách văn bản thành danh sách từ
12
13            Parameters:
14                file_path (str): Đường dẫn đến file.
15
16            Returns:
17                dict: Từ điển đếm số lần xuất hiện của từng từ.
18                """
19            counter = []
20
21            # Your Code Here
22
23
24            return counter
25            file_path = "./P1_data.txt"
26            result = count_word(file_path)
27            assert result["who"] == 3
28            print(result["man"])
29

```

- (a) 4
 (b) 5
 (c) 6
 (d) 9

4. Hoàn thành chương trình sau với mô tả bài toán tính khoảng cách Levenshtein 2.2.4 và cho biết đầu ra của chương trình dưới đây là gì?

```

1     def levenshtein_distance(token1, token2):
2         """

```

```
3     Tính khoảng cách Levenshtein (edit distance) giữa hai chuỗi.
4
5     Khoảng cách Levenshtein là số lần chỉnh sửa nhỏ nhất (chèn, xóa, thay thế)
6     để biến chuỗi token1 thành token2.
7
8     Parameters:
9         token1 (str): Chuỗi thứ nhất.
10        token2 (str): Chuỗi thứ hai.
11
12    Returns:
13        int: Khoảng cách Levenshtein giữa hai chuỗi.
14        """
15        # Your Code Here
16
17        # End Code Here
18
19        return distance
20
21    assert levenshtein_distance("hi", "hello") == 4
22    print(levenshtein_distance("hola", "hello"))
23
```

- (a) 1
- (b) 2
- (c) 3
- (d) 4

5. Hoàn thành chương trình sau. Đầu ra của chương trình dưới đây là gì?

```

1  def check_the_number(N):
2      list_of_numbers = []
3      result = ""
4      for i in range(1, 5):
5          #Your code here
6          # Su dung append them i vao trong list_of_number
7          if N in list_of_numbers:
8              results = "True"
9          if N not in list_of_numbers:
10             results = "False"
11     return results
12
13 N = 7
14 assert check_the_number(N) == "False"
15
16 N = 2
17 results = check_the_number(N)
18 print(results)
19

```

- (a) True
- (b) False
- (c) None
- (d) Raise an Error

6. Hoàn thành chương trình sau. Đầu ra của chương trình dưới đây là gì?

```

1  def my_function(data, max_value, min_value):
2      result = []
3      for i in data:
4          # Your code here
5          # Neu i < min_value thi them min vao result
6          elif i > max_value:
7              result.append(max_value)
8          else:
9              result.append(i)
10     return result
11

```

```
12     # test
13     my_list = [5, 2, 5, 0, 1]
14     max_value = 1
15     min_value = 0
16     assert my_function(my_list, max_value, min_value) == [1, 1, 1
17                                     , 0, 1]
18     my_list = [10, 2, 5, 0, 1]
19     max = 2
20     min = 1
21     print(my_function(my_list, max_value, min_value))
```

- (a) [10, 2, 5, 1, 1]
(b) [0, 2, 2, 0, 0]
(c) [2, 2, 2, 1, 1]
(d) Raise an Error
7. Hoàn thành chương trình sau tìm giá trị bé nhất. Đầu ra của chương trình dưới đây là gì?

```
1 def my_function(n):
2     # Your code here
3     # Tìm giá trị nhỏ nhất của một list
4
5     my_list = [1, 22, 93, -100]
6     assert my_function(my_list) == -100
7
8     my_list = [1, 2, 3, -1]
9     print(my_function(my_list))
10
```

- (a) None
(b) Raise an Error
(c) -1
(d) 3

8. Hoàn thành chương trình sau tìm giá trị lớn nhất. Đầu ra của chương trình dưới đây là gì?

```
1 def my_function(n):
2     #Your code here
3     # Tìm giá trị lớn nhất của một list
4
5     my_list = [1001, 9, 100, 0]
6     assert my_function(my_list) == 1001
7
8     my_list = [1, 9, 9, 0]
9     print(my_function(my_list))
10
```

- (a) None
- (b) Raise an Error
- (c) 0
- (d) 9

9. Hoàn thành chương trình sau. Đầu ra của chương trình dưới đây là gì?

```

1 def my_function(integers, number = 1):
2     return any(#Your code here: Thuc hien duyet tung phan tu
3             #trong integers,
4             #so sanh tung phan tu voi number, neu bang
5             #nhau tra ve True,
6             #khac nhau tra ve False
7             #vi du: integers = [1, 2, 3], number = 2,
8             #ban se tao ra list [False, True, False])
9
10 my_list = [1, 3, 9, 4]
11 assert my_function(my_list, -1) == False
12
13 my_list = [1, 2, 3, 4]
14 print(my_function(my_list, 2))

```

- (a) 1
 (b) 4
 (c) True
 (d) False
10. Hoàn thành chương trình sau tính giá trị trung bình. Đầu ra của chương trình dưới đây là gì?

```

1 def my_function(list_nums = [0, 1, 2]):
2     var = 0
3     for i in list_nums:
4         var += i
5     return #Your code here: Tra ve gia tri trung binh cua
6             #list
7             #bang cach chia var cho so luong phan tu trong
8             #list_nums
9
10 assert my_function([4, 6, 8]) == 6
11 print(my_function())

```

- (a) 1.0
(b) 2.0
(c) Raise an Error
(d) A and C
11. Hoàn thành chương trình sau tìm các số chia hết cho 3. Đầu ra của chương trình dưới đây là gì?

```
1 def my_function(data):
2     var = []
3     for i in data:
4         #Your code here
5         #Neu i chia het cho 3 thi them i vao list var
6         return var
7
8 assert my_function([3, 9, 4, 5]) == [3, 9]
9 print(my_function([1, 2, 3, 5, 6]))
10
```

- (a) [3, 6]
(b) [1, 2, 3, 5, 6]
(c) [1, 3, 6]
(d) [5, 1]
12. Hoàn thành chương trình sau đây thực hiện tính giai thừa của 1 số. Đầu ra của chương trình dưới đây là gì?

```
1 def my_function(y):
2     var = 1
3     while(y > 1):
4         #Your code here
5         return var
6
7 assert my_function(8) == 40320
8 print(my_function(4))
9
```

- (a) 0
(b) 20
(c) 24
(d) Raise an Error
13. Hoàn thành chương trình đảo ngược chuỗi dưới đây. Đầu ra của chương trình là gì?
- ```
1 def my_function(x):
2 #your code here
3
4 x = "I can do it"
5 assert my_function(x) == "ti od nac I"
6
7 x = "apricot"
8 print(my_function(x))
9
```
- (a) apricot  
(b) tocirpa  
(c) Raise an Error  
(d) None
14. Hoàn thành chương trình dưới đây. Đầu ra của chương trình là gì?

```
1 def function_helper(x):
2 #Your code here
3 #Neu x > 0 tra ve "T", nguoc lai tra ve "N"
4
5 def my_function(data):
6 res = [function_helper(x) for x in data]
7 return res
8
9 data = [10, 0, -10, -1]
10 assert my_function(data) == ["T", "N", "N", "N"]
11
12 data = [2, 3, 5, -1]
13 print(my_function(data))
```

14

- (a) ["N", "T", "T", "N"]
- (b) ["T", "N", "T", "N"]
- (c) ["T", "T", "T", "N"]
- (d) Raise an Error

15. Hoàn thành chương trình dưới đây để loại bỏ những phần tử trùng nhau. Đầu ra của chương trình là gì?

```
1 def function_helper(x, data):
2 for i in data:
3 #Your code here
4 #Neu x == i thi return 0
5 return 1
6
7 def my_function(data):
8 res = []
9 for i in data:
10 if function_helper(i, res):
11 res.append(i)
12 return res
13
14 lst = [10, 10, 9, 7, 7]
15 assert my_function(lst) == [10, 9, 7]
16
17 lst = [9, 9, 8, 1, 1]
18 print(my_function(lst))
19
```

- (a) [9, 8, 1]
- (b) [1, 1, 1]
- (c) [9, 9, 8, 1, 1]
- (d) Raise an Error

1. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

| Word Suggestion - Rubric |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Câu                      | Kiến Thức                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Đánh Giá                                                                                                                                                                                                                                                                                                                                                                                                 |
| 1                        | <ul style="list-style-type: none"> <li>- Sử dụng dictionary data để đếm số lần chữ xuất hiện trong 1 từ</li> <li>- Sử dụng dictionary data để đếm số lần từ xuất hiện trong 1 các câu của 1 file</li> <li>- Việc đếm số từ và chữ là 1 trong những bước cơ bản để học NLP</li> <li>- Đọc file và lấy từng line trong file</li> <li>- Xử lý string cơ bản viết thường, tách từ</li> <li>- Sử dụng dictionary kiểm tra key có tồn tại hay không, và cách cập nhật value của 1 key khi đã tồn tại trong dictionarry</li> </ul> | <ul style="list-style-type: none"> <li>- Bước đầu biết kết hợp xử lý string và dictionary để lưu data trong dictionary cho task NLP</li> <li>- Biết thao tác với dictionary cơ bản và quan trọng nhất, kiểm tra key và cập nhật value khi key tồn tại</li> <li>- Biết thao tác với file và lấy nội dung từng line trong file - Làm quen với xử lý string tách từ, và convert sang viết thường</li> </ul> |
| 2                        | <ul style="list-style-type: none"> <li>- Sử dụng vòng lặp for để duyệt qua list.</li> <li>- Sử dụng một số phương thức của List trong Python: len(), append().</li> <li>- Sử dụng hàm có sẵn max() trong Python để tìm phần tử lớn nhất.</li> <li>- Kỹ thuật slicing trong List.</li> <li>- Lý thuyết cơ bản về sliding window</li> </ul>                                                                                                                                                                                   | <ul style="list-style-type: none"> <li>- Vận dụng được các hàm, phương thức có sẵn liên quan đến List trong Python để giải quyết bài toán.</li> </ul>                                                                                                                                                                                                                                                    |
| 3                        | <ul style="list-style-type: none"> <li>- Sử dụng các từ khóa điều kiện in, not trong Python để tạo câu điều kiện với List.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                       | Biết sử dụng các toán tử điều kiện trong Python để tương tác với List.                                                                                                                                                                                                                                                                                                                                   |
| 4                        | <ul style="list-style-type: none"> <li>- Kỹ thuật list comprehension. Sử dụng các câu điều kiện để tạo điều kiện ràng buộc cho người dùng.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                       | <ul style="list-style-type: none"> <li>- Vận dụng kỹ thuật list comprehension hoặc vòng lặp for để tạo một List mới dựa trên List ban đầu.</li> </ul>                                                                                                                                                                                                                                                    |

# Chương 3

## Lập trình hướng đối tượng và cài đặt class trong PyTorch

### 3.1 Câu hỏi tự luận

#### 3.1.1 Xây dựng class tính Sigmoid

**Sigmoid Function:** Sigmoid Function, hay còn được gọi là logistic function, là một trong những activation function cơ bản nhất trong machine learning và neural networks. Hình dạng của nó giống như chữ "S" nằm ngang. Sigmoid chuyển đổi mọi giá trị đầu vào thành một giá trị đầu ra nằm giữa 0 và 1, với một sự chuyển đổi mượt mà tại giá trị 0.

**Ứng Dụng:** Sigmoid Function thường được sử dụng trong các bài toán phân loại nhị phân, nơi mà mục tiêu là phân loại đầu vào thành một trong hai lớp.

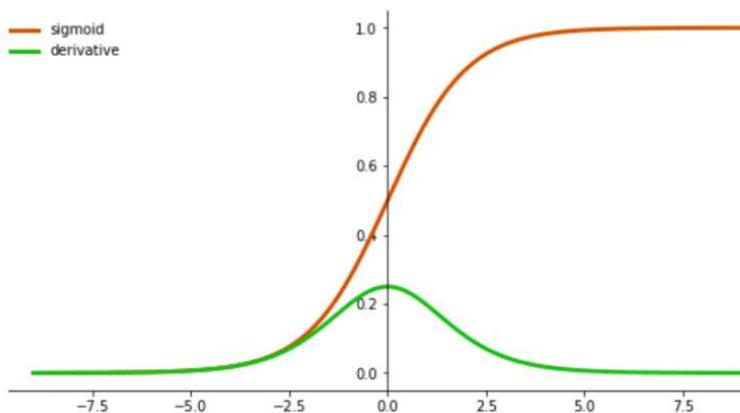
##### Ưu Điểm:

- **Dễ hiểu và triển khai:** Do tính chất đơn giản và phổ biến, Sigmoid rất được triển khai trong nhiều loại mạng neuron.
- **Đầu ra nằm trong khoảng (0,1):** Giá trị đầu ra luôn nằm trong khoảng từ 0 đến 1, giúp dễ dàng diễn giải như là xác suất.

##### Nhược điểm:

- **Vanishing gradient problem:** Khi đầu vào có giá trị lớn hoặc nhỏ, đạo hàm của Sigmoid tiệm cận đến 0, dẫn đến vấn đề vanishing gradient, làm chậm quá trình học của mạng.
- **Tâm đối xứng không nằm tại 0:** Tâm đối xứng không nằm tại điểm 0, điều này có thể gây ra vấn đề trong việc điều chỉnh trọng số trong quá trình học.

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$



```
data = [1, 5, -4, 3, -2]
```

```
data_a = sigmoid(data)
```

```
data_a = [0.731, 0.993, 0.017, 0.95, 0.119]
```

Hình 3.1: Hàm Sigmoid và đạo hàm.

```

1 # Examples 1
2 import torch
3
4 # input data
5 x = torch.tensor([1.0, 5.0, -4.0])
6
7 # sigmoid function
8 output = torch.sigmoid(x)
9 print(output)
10 >> tensor([0.7311, 0.9933, 0.0180])

```

### 3.1.2 Xâu dựng Ward

Một Ward gồm có name (string) và danh sách của mọi người trong Ward. Một người person có thể là student, doctor, hoặc teacher. Một student gồm có name, yob (int) (năm sinh), và grade (string). Một teacher gồm có name,

yob, và subject (string). Một doctor gồm có name, yob, và specialist (string). Lưu ý cần sử dụng a list để chứa danh sách của mọi người trong Ward.

1. Thực hiện các class student, doctor, và teacher theo mô tả trên. Thực hiện describe() method để print ra tất cả thông tin của các objects.
2. Viết addPerson(person) method trong Ward class để add thêm một người mới với nghề nghiệp bất kỳ (student, teacher, doctor) vào danh sách người của ward. Tạo ra một ward object, và thêm vào 1 student, 2 teacher, và 2 doctor. Thực hiện describe() method để in ra tên ward (name) và toàn bộ thông tin của từng người trong ward.
3. Viết countDoctor() method để đếm số lượng doctor trong ward.
4. Viết sortAge() method để sort mọi người trong ward theo tuổi của họ với thứ tự tăng dần (Gợi ý: Có thể sử dụng sort của list hoặc viết thêm function đều được).
5. Viết aveTeacherYearOfBirth() method để tính trung bình năm sinh của các teachers trong ward.

```

1 # Examples
2 # 2(a)
3 student1 = Student(name="studentA", yob=2010, grade="7")
4 student1.describe()
5 #output
6 >> Student - Name: studentA - YoB: 2010 - Grade: 7
7
8 teacher1 = Teacher(name="teacherA", yob=1969, subject="Math")
9 teacher1.describe()
10 #output
11 >> Teacher - Name: teacherA - YoB: 1969 - Subject: Math
12
13 doctor1 = Doctor(name="doctorA", yob=1945, specialist="
14 Endocrinologists")
15 doctor1.describe()
16 #output
17 >> Doctor - Name: doctorA - YoB: 1945 - Specialist: Endocrinologists
18
19 # 2(b)
20 print()

```

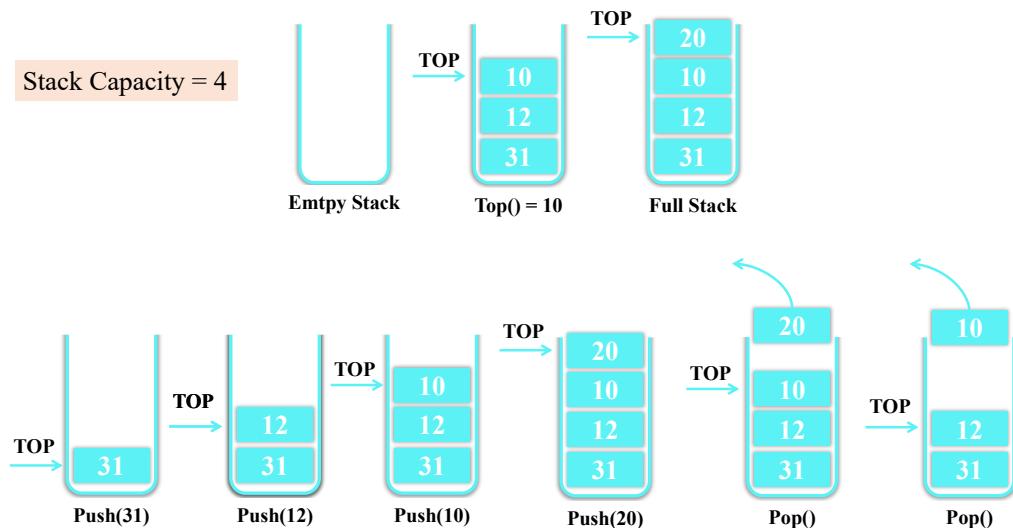
```
21 teacher2 = Teacher(name="teacherB", yob=1995, subject="History")
22 doctor2 = Doctor(name="doctorB", yob=1975, specialist="Cardiologists
23 ")
24 ward1 = Ward(name="Ward1")
25 ward1.addPerson(student1)
26 ward1.addPerson(teacher1)
27 ward1.addPerson(teacher2)
28 ward1.addPerson(doctor1)
29 ward1.addPerson(doctor2)
30 ward1.describe()
31
32 #output
33 >> Ward Name: Ward1
34 Student - Name: studentA - YoB: 2010 - Grade: 7
35 Teacher - Name: teacherA - YoB: 1969 - Subject: Math
36 Teacher - Name: teacherB - YoB: 1995 - Subject: History
37 Doctor - Name: doctorA - YoB: 1945 - Specialist: Endocrinologists
38 Doctor - Name: doctorB - YoB: 1975 - Specialist: Cardiologists
39
40 # 2(c)
41 print(f"\nNumber of doctors: {ward1.countDoctor()}")
42
43 #output
44 >> Number of doctors: 2
45
46 # 2(d)
47 print("\nAfter sorting Age of Ward1 people")
48 ward1.sortAge()
49 ward1.describe()
50
51 #output
52 >> After sorting Age of Ward1 people
53 Ward Name: Ward1
54 Student - Name: studentA - YoB: 2010 - Grade: 7
55 Teacher - Name: teacherB - YoB: 1995 - Subject: History
56 Doctor - Name: doctorB - YoB: 1975 - Specialist: Cardiologists
57 Teacher - Name: teacherA - YoB: 1969 - Subject: Math
58 Doctor - Name: doctorA - YoB: 1945 - Specialist: Endocrinologists
59
60 # 2(e)
61 print(f"\nAverage year of birth (teachers): {ward1.
62 aveTeacherYearOfBirth()}")
63
64 #output
```

63 |>> Average year of birth (teachers): 1982.0

### 3.1.3 Xây dựng class Stack

Thực hiện xây dựng class Stack với các chức năng (method) sau đây:

- **initialization** method nhận một input “capacity”: dùng để khởi tạo stack với capacity là số lượng element mà stack có thể chứa.
- **.isEmpty()**: kiểm tra stack có đang rỗng.
- **.isFull()**: kiểm tra stack đã full chưa.
- **.pop()**: loại bỏ top element và trả về giá trị đó.
- **.push(value)** add thêm value vào trong stack.
- **.top()** lấy giá trị top element hiện tại của stack, nhưng không loại bỏ giá trị đó.
- **Lưu ý:** Không cần thiết phải thực hiện với pointer như trong hình minh họa số 3.2.



Hình 3.2: Hình ảnh minh họa về các Stack và một số phương thức trên Stack.

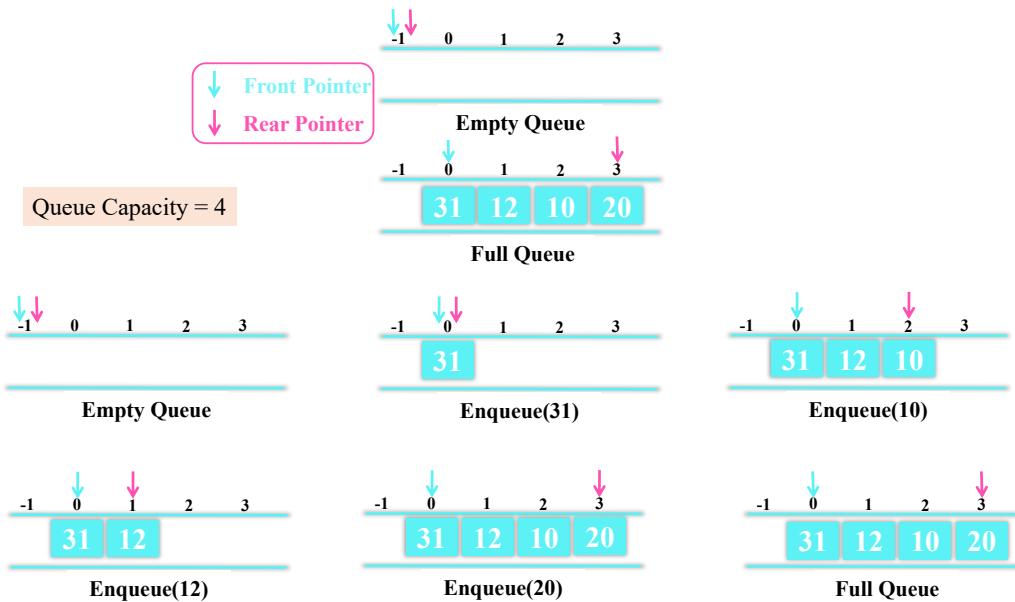
```
1 stack1 = MyStack(capacity=5)
2
3 stack1.push(1)
4
5 stack1.push(2)
6
7 print(stack1.isFull())
8 >> False
9
10 print(stack1.top())
11 >>2
12
13 print(stack1.pop())
14 >> 2
15
16 print(stack1.top())
17 >> 1
18
19 print(stack1.pop())
20 >> 1
21
22 print(stack1.isEmpty())
23 >> True
```

### 3.1.4 Xây dựng class Queue

Thực hiện xây dựng class Queue với các chức năng (method) sau đây:

- **initialization** method nhận một input "capacity": dùng để khởi tạo queue với capacity là số lượng element mà queue có thể chứa
- **.isEmpty()**: kiểm tra queue có đang rỗng
- **.isFull()**: kiểm tra queue đã full chưa
- **.dequeue()**: loại bỏ first element và trả về giá trị đó
- **.enqueue(value)** add thêm value vào trong queue
- **.front()** lấy giá trị first element hiện tại của queue, nhưng không loại bỏ giá trị đó
- **Không cần thiết phải thực hiện với các pointers như trong hình minh họa**

```
1 queue1 = MyQueue(capacity=5)
2
3 queue1.enqueue(1)
4
5 queue1.enqueue(2)
6
7 print(queue1.isFull())
8 >> False
9
10 print(queue1.front())
11 >> 1
12
13 print(queue1.dequeue())
14 >> 1
15
16 print(queue1.front())
17 >> 2
18
19 print(queue1.dequeue())
20 >> 2
21
22 print(queue1.isEmpty())
```



Hình 3.3: Hình ảnh minh họa về các Queue và một số phương thức trên Queue.

### 3.2 Câu hỏi trắc nghiệm

1. (Repeat PyTorch code) Kết quả của đoạn code dưới đây là gì?

```
1 import torch
2
3 # input data
4 x = torch.tensor([5.0, 3.0])
5
6 # sigmoid function
7 output = torch.sigmoid(x)
8 print(output)
```

- (a) [0.0900, 0.9526].  
(b) [0.9933, 0.6652].  
(c) [0.9933, 0.9526].  
(d) [0.1900, 0.2447].
2. Hoàn thành đoạn code sau đây theo công thức tính sigmoid và cho biết kết quả in ra màn hình là gì?

```
1 import torch
2 import torch.nn as nn
3
4 class MySigmoid(nn.Module):
5 def __init__(self):
6 super().__init__()
7
8 def forward(self, x):
9 ### Your Code Here
10
11 ### End Code Here
12
13 data = torch.Tensor([3.0, -2.0])
14 my_sigmoid = MySigmoid()
15 output = my_sigmoid(data)
16 output
```

- (a) [0.1192, 0.9526].  
 (b) [0.9526, 0.1192].  
 (c) [0.1192, 0.2447].  
 (d) [0.7054, 0.1192].
3. Một người (person) có thể là student, doctor, hoặc teacher. Một student gồm có name (string), yob (int) (năm sinh), và grade (string). Các bạn thực hiện viết class Student theo mô tả trên (Các bạn sẽ viết thêm describe() method để print ra tất cả thông tin của object). Theo đó, kết quả đầu ra khi thực hiện lời gọi method describe() là gì?

```

1 from abc import ABC, abstractmethod
2
3 class Person(ABC):
4 def __init__(self, name:str, yob:int):
5 self._name = name
6 self._yob = yob
7
8 def getYoB(self):
9 return self._yob
10
11 @abstractmethod
12 def describe(self):
13 pass
14
15
16 class Student(Person):
17 def __init__(self, name:str, yob:int, grade:str):
18 ### Your Code Here
19
20 ### End Code Here
21
22 def describe(self):
23 ### Your Code Here
24
25 ### End Code Here
26
27 student1 = Student(name="studentZ2023", yob=2011, grade="6")
 student1.describe()

```

- (a) Student - Name: studentZ2023 - YoB: 2011 - Grade: 6.

- (b) Student - Name: studentZ2023 - YoB: 6 - Grade: 2011.
- (c) Student - Name: 6 - YoB: studentZ2023 - Grade: 2011.
- (d) Tất cả đều sai.
4. Một người (person) có thể là student, doctor, hoặc teacher. Một teacher gồm có name (string), yob (int), và subject (string). Các bạn thực hiện viết class Teacher theo mô tả trên (Các bạn sẽ viết thêm describe() method để in ra màn hình tất cả thông tin của object). Theo đó, kết quả đầu ra khi thực hiện lời gọi method describe() là gì?

```

1 from abc import ABC, abstractmethod
2
3 class Person(ABC):
4 def __init__(self, name:str, yob:int):
5 self._name = name
6 self._yob = yob
7
8 def getYoB(self):
9 return self._yob
10
11 @abstractmethod
12 def describe(self):
13 pass
14
15
16 class Teacher(Person):
17 def __init__(self, name:str, yob:int, subject:str):
18 ### Your Code Here
19
20 ### End Code Here
21
22 def describe(self):
23 ### Your Code Here
24
25 ### End Code Here
26
27 teacher1 = Teacher(name="teacherZ2023", yob=1991, subject="History") teacher1.describe()

```

- (a) Teacher - Name: 1991 - YoB: teacherZ2023 - Subject: History.

- (b) Teacher - Name: teacherZ2023 - YoB: 1991 - Subject: History.
- (c) Teacher - Name: History - YoB: teacherZ2023 - Subject: 1991.
- (d) Tất cả đều sai.
5. Một người (person) có thể là student, doctor, hoặc teacher. Một doctor gồm có name (string), yob (string), và specialist (string). Các bạn thực hiện viết class Teacher theo mô tả trên (Các bạn sẽ viết thêm describe() method để print ra tất cả thông tin của object). Theo đó, kết quả đầu ra khi thực hiện lời gọi method describe() là gì?

```

1 from abc import ABC, abstractmethod
2
3 class Person(ABC):
4 def __init__(self, name:str, yob:int):
5 self._name = name
6 self._yob = yob
7
8 def getYoB(self):
9 return self._yob
10
11 @abstractmethod
12 def describe(self):
13 pass
14
15
16 class Doctor(Person):
17 def __init__(self, name:str, yob:int, specialist:str):
18 ### Your Code Here
19
20 ### End Code Here
21
22 def describe(self):
23 ### Your Code Here
24
25 ### End Code Here
26
27 doctor1 = Doctor(name="doctorZ2023", yob=1981, specialist="Endocrinologists") doctor1.
 describe()

```

- (a) Doctor - Name: doctorZ2023 - YoB: 1981 - Specialist: Endocrinologists.

- (b) Doctor - Name: 1981 - YoB: doctorZ2023 - Specialist: Endocrinologists.  
 (c) Teacher - Name: History - YoB: teacherZ2023 - Subject: 1991.  
 (d) Tất cả đều sai.
6. Một Ward gồm có name (string) và danh sách của mọi người trong Ward. Một người person có thể là student, doctor, hoặc teacher và cần sử dụng một list để chứa danh sách của mọi người trong Ward. Viết addPerson(person) method trong Ward class để add thêm một người mới với nghề nghiệp bất kỳ (student, teacher, doctor) vào danh sách người của ward. Tạo ra một ward object, và thêm vào 1 student, 2 teacher, và 2 doctor. Thực hiện describe() method để in ra tên ward (name) và toàn bộ thông tin của từng người trong ward. Như vậy, đáp án đúng nhất cho hàm đếm số lượng doctor là gì?

```

1 class Ward:
2 def __init__(self, name:str):
3 self.__name = name
4 self.__listPeople = list()
5
6 def addPerson(self, person:Person):
7 self.__listPeople.append(person)
8
9 def describe(self):
10 print(f"Ward Name: {self.__name}")
11 for p in self.__listPeople:
12 p.describe()
13
14 def countDoctor(self):
15 ### Your Code Here
16
17 ### End Code Here
18
19 student1 = Student(name="studentA", yob=2010, grade="7")
20 teacher1 = Teacher(name="teacherA", yob=1969, subject="Math")
21 teacher2 = Teacher(name="teacherB", yob=1995, subject="History")
22 doctor1 = Doctor(name="doctorA", yob=1945, specialist="Endocrinologists")
23 doctor2 = Doctor(name="doctorB", yob=1975, specialist="Cardiologists")
24 ward1 = Ward(name="Ward1")
25 ward1.addPerson(student1)

```

```

26 ward1.addPerson(teacher1)
27 ward1.addPerson(teacher2)
28 ward1.addPerson(doctor1)
29 ward1.addPerson(doctor2)
30 ward1.countDoctor()

```

- (a) 4.  
 (b) 3.  
 (c) 2.  
 (d) 1.

7. Thực hiện xây dựng class Stack với các chức năng (method) sau đây:

- init() method nhận một input “capacity”: dùng để khởi tạo stack với capacity là số lượng element mà stack có thể chứa.
- isFull(): kiểm tra stack đã đầy chưa.
- push(value): thêm value vào trong stack.

Kết quả đầu ra là gì?

```

1 class MyStack:
2 def __init__(self, capacity):
3 self.__capacity = capacity
4 self.__stack = []
5
6 def isFull(self):
7 return len(self.__stack) == self.__capacity
8
9 def push(self, value):
10 ### Your Code Here
11
12 ### End Code Here
13
14 stack1 = MyStack(capacity=5)
15 stack1.push(1)
16 stack1.push(2)
17 print(stack1.isFull())

```

- (a) True.  
(b) False.  
(c) None.  
(d) Raise an error.
8. Thực hiện xây dựng class Stack với các chức năng (method) sau đây:
- init() method nhận một input “capacity”: dùng để khởi tạo stack với capacity là số lượng element mà stack có thể chứa.
  - isEmpty(): kiểm tra stack có đang rỗng.
  - isFull(): kiểm tra stack đã đầy chưa.
  - push(value): thêm value vào trong stack.
  - top(): lấy giá trị top element hiện tại của stack, nhưng không loại bỏ giá trị đó.

Kết quả đầu ra là gì?

```
1 class MyStack:
2 def __init__(self, capacity):
3 self.__capacity = capacity
4 self.__stack = []
5
6 def isFull(self):
7 return len(self.__stack) == self.__capacity
8
9 def push(self, value):
10 ### Your Code Here
11
12 ### End Code Here
13
14 def top(self):
15 ### Your Code Here
16
17 # End Code Here
18
19 stack1 = MyStack(capacity=5)
20 stack1.push(1)
21 stack1.push(2)
22 print(stack1.top())
```

- (a) 1.  
(b) 2.  
(c) None.  
(d) Raise an error.
9. Thực hiện xây dựng class Queue với các chức năng (method) sau đây
- init() method nhận một input “capacity”: dùng để khởi tạo queue với capacity là số lượng element mà queue có thể chứa.
  - isFull(): kiểm tra queue đã đầy chưa.
  - enqueue(value): thêm value vào trong queue.
- Kết quả đầu ra là gì?

```
1 class MyQueue:
2 def __init__(self, capacity):
3 self.__capacity = capacity
4 self.__queue = []
5
6 def isFull(self):
7 return len(self.__queue) == self.__capacity
8
9 def push(self, value):
10 ### Your Code Here
11
12 ### End Code Here
13
14 queue1 = MyQueue(capacity=5)
15 queue1.push(1)
16 queue1.push(2)
17 print(queue1.isFull())
```

- (a) False.  
(b) True.  
(c) None.  
(d) Raise an error.

10. Thực hiện xây dựng class Queue với các chức năng (method) sau đây
- init() method nhận một input “capacity”: dùng để khởi tạo queue với capacity là số lượng element mà queue có thể chứa.
  - isFull(): kiểm tra queue đã đầy chưa.
  - push(value): thêm value vào trong queue.
  - front(): lấy giá trị first element hiện tại của queue, nhưng không loại bỏ giá trị đó.

Kết quả đầu ra là gì?

```
1 class MyQueue:
2 def __init__(self, capacity):
3 self.__capacity = capacity
4 self.__queue = []
5
6 def isEmpty(self):
7 return len(self.__queue) == 0
8
9 def isFull(self):
10 return len(self.__queue) == self.__capacity
11
12 def dequeue(self):
13
14 def enqueue(self, value):
15
16 def front(self):
17 ### Your Code Here
18
19queue1 = MyQueue(capacity=5)
20queue1.enqueue(1)
21assert queue1.is_full() == False
22queue1.enqueue(2)
23print(queue1.front())
```

- (a) 4.
- (b) 3.
- (c) 2.
- (d) 1.

1. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần bài tập, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

### 3. Rubric:

| Mục | Kiến Thức                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Đánh Giá                                                                                                                                                                                                                                                                                                                                                                                                |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.  | <ul style="list-style-type: none"> <li>- Làm quen với Pytorch.</li> <li>- Hiểu cơ bản về nn.Module trong Pytorch.</li> <li>- Tạo Class mới từ Module.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                    | <ul style="list-style-type: none"> <li>- Khai báo các tham số.</li> <li>- Ghi đè phương thức forward.</li> <li>- Áp dụng cho hàm Sigmoid.</li> </ul>                                                                                                                                                                                                                                                    |
| 2.  | <ul style="list-style-type: none"> <li>- Khái niệm Abstract Class trong Python.</li> <li>- Khái niệm Inheritance trong Python.</li> <li>- Khái niệm Polymorphism trong Python.</li> <li>- Khái niệm Encapsulation trong Python.</li> <li>- Python Access Modifiers: public, protected, và private members của class.</li> <li>- Sử dụng nhiều method với các yêu cầu khác nhau trong Python.</li> <li>- Dểm các object cùng class trong 1 list các object.</li> <li>- Sort 1 list các object.</li> <li>- Lọc các object cùng loại và tính toán dựa trên các data member.</li> </ul> | <ul style="list-style-type: none"> <li>- Hiểu và thực hiện được các khái niệm Abstract class, Inheritance, Polymorphism, Encapsulation.</li> <li>- Sử dụng được các Python Access Modifiers.</li> <li>- Khởi tạo được các method khác nhau trong class theo yêu cầu.</li> <li>- Biết cách lọc và đếm các object cùng class.</li> <li>- Thực hiện được xếp các object dựa trên các attribute.</li> </ul> |
| 3.  | <ul style="list-style-type: none"> <li>- Hiểu rõ về các thành phần và cơ chế hoạt động của stack (kiểm tra full hay empty, pop, push, lấy top element của stack).</li> <li>- Sử dụng các kiểu data cần thiết và dùng class để tạo thành 1 stack class.</li> <li>- Hiểu được rằng stack có thể được thực hiện từ các kiểu dữ liệu khác nhau chỉ cần nắm rõ nguyên tắc hoạt động.</li> </ul>                                                                                                                                                                                          | <ul style="list-style-type: none"> <li>- Cơ bản biết kết hợp các kiểu data khác nhau để tạo ra một kiểu mà mình mong muốn.</li> <li>- Đã có khả năng thiết kế stack dựa trên các kiểu data và kết hợp với class để tạo stack class hoàn chỉnh.</li> <li>- Hiểu được cơ bản cách thiết kế 1 data structure (cần tìm hiểu nắm rõ được cơ chế hoạt động và các hành vi của data structure).</li> </ul>     |

|    |                                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                                                                                                                 |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 4. | <ul style="list-style-type: none"><li>- Hiểu rõ về các thành phần và cơ chế hoạt động của queue (kiểm tra full hay empty, push, pop, lấy first element của queue).</li><li>- Sử dụng các kiểu data cần thiết và dùng class để tạo thành 1 queue class.</li><li>- Hiểu được rằng queue có thể được thực hiện từ các kiểu dữ liệu khác nhau chỉ cần nắm rõ nguyên tắc hoạt động.</li></ul> | <ul style="list-style-type: none"><li>- Cơ bản biết kết hợp các kiểu data khác nhau để tạo ra một kiểu mà mình mong muốn.</li><li>- Đã có khả năng thiết kế queue dựa trên các kiểu data và kết hợp với class để tạo queue class hoàn chỉnh.</li><li>- Hiểu được cơ bản cách thiết kế 1 data structure (cần tìm hiểu nắm rõ được cơ chế hoạt động và các hành vi của data structure).</li></ul> |
|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## Chương 4

# Project 1: Triển khai hệ thống đơn giản với Streamlit

### 4.1 Giới thiệu



Hình 4.1: Streamlit

**Streamlit** là một thư viện mã nguồn mở của Python, được phát triển nhằm hỗ trợ việc xây dựng các ứng dụng web tương tác phục vụ cho khoa học dữ liệu và machine learning một cách nhanh chóng và thuận tiện. Framework này cho phép các lập trình viên chuyển đổi mã Python thông thường thành ứng dụng web chỉ với vài dòng lệnh đơn giản.

Với Streamlit, bạn có thể dễ dàng tạo dashboard, trực quan hóa dữ liệu qua các biểu đồ, cũng như triển khai các mô hình học máy, tất cả đều không yêu cầu kiến thức nền tảng về lập trình web.

#### 4.1.1 Lợi ích của Streamlit

Streamlit là một lựa chọn phù hợp cho các ứng dụng web đơn giản, đặc biệt trong các dự án nhỏ hoặc dùng để trình diễn mô hình. Một số lợi ích chính gồm:

- Phù hợp với các dự án quy mô nhỏ, ứng dụng nội bộ hoặc demo nhanh.
- Không yêu cầu cấu hình bảo mật phức tạp.
- Thiết kế theo hướng một trang duy nhất, không cần liên kết nhiều trang.
- Giao diện đơn giản, dễ sử dụng, ít cần tùy chỉnh về layout.
- Có thể tích hợp với Flask, Django hoặc các công cụ frontend khác khi cần mở rộng.

#### 4.1.2 Một số hàm thông dụng trong Streamlit

##### Các thành phần giao diện cơ bản trong Streamlit:

Các thành phần giao diện trong Streamlit là các yếu tố trực quan và tương tác, được sử dụng để xây dựng ứng dụng web. Mỗi thành phần cung cấp một chức năng cụ thể và có thể được tùy chỉnh linh hoạt để phù hợp với yêu cầu của ứng dụng. Dưới đây là một số thành phần cơ bản thường được sử dụng:

- `st.title("...")`: Dùng để tạo tiêu đề chính cho ứng dụng.
- `st.file_uploader("...", type=["..."])`: Tạo công cụ tải tệp lên ứng dụng. Tham số `type` cho phép giới hạn định dạng tệp được chấp nhận, ví dụ `["csv", "txt", "pdf"]`.
- `st.write("...")`: Dùng để hiển thị văn bản, bảng dữ liệu, biểu đồ hoặc các loại dữ liệu khác.
- `st.sidebar`: Tạo thanh điều hướng bên trái, giúp tổ chức các widget và điều khiển hợp lý hơn.
- `st.button("...")`: Tạo nút bấm cho phép thực thi hành động khi người dùng nhấn vào.

- `st.plotly_chart()`: Tạo biểu đồ tương tác sử dụng thư viện Plotly, hỗ trợ zoom, pan và các hành động tương tác khác.
- `st.line_chart()`, `st.bar_chart()`, `st.area_chart()`: Các hàm tích hợp sẵn để trực quan hóa dữ liệu dưới dạng biểu đồ đường, cột và vùng, thường sử dụng với DataFrame hoặc mảng NumPy.
- `st.map()`: Hiển thị dữ liệu địa lý trên bản đồ tương tác, áp dụng cho các tập dữ liệu có tọa độ kinh độ và vĩ độ.

Ngoài các thành phần trên, bạn có thể tham khảo thêm các widget và tính năng khác trong tài liệu chính thức của Streamlit để xây dựng các ứng dụng phong phú và tương tác hơn.

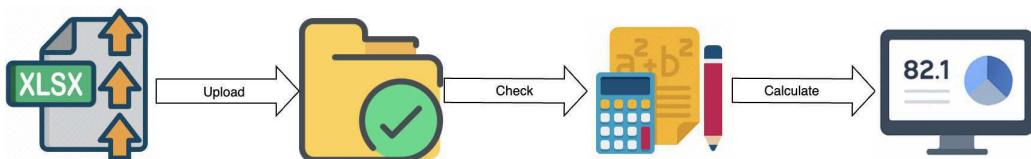
## 4.2 Thực hành Streamlit cơ bản

### 4.2.1 Giới thiệu ứng dụng

Trong phần thực hành này, ta sẽ xây dựng một ứng dụng hỗ trợ phân tích dữ liệu điểm số học sinh từ các file Excel. Các chức năng chính bao gồm: tính điểm trung bình, phân loại điểm theo từng khoảng, và trực quan hóa kết quả dưới dạng biểu đồ.

### 4.2.2 Pipeline tổng quan

Pipeline tổng quan của chương trình như sau: Người dùng upload file, hệ thống tự động tính toán và trả về kết quả phân tích cùng với biểu đồ trực quan.



Hình 4.2: Pipeline minh họa quá trình xử lý điểm số

### 4.2.3 Cài đặt môi trường và các thư viện cần thiết

Để xây dựng ứng dụng Streamlit một cách dễ dàng và tránh xung đột thư viện, ta nên tạo một môi trường ảo Python mới. Dưới đây là hai cách phổ biến để thực hiện: dùng `conda` hoặc `venv`.

#### Cách 1: Dùng `conda` (nếu bạn dùng Anaconda hoặc Miniconda)

- Tạo môi trường mới:

```
conda create -n streamlit_app -y
```

- `-n streamlit_app`: chỉ định tên môi trường mới là `streamlit_app`.
- `-y`: tự động xác nhận (yes) tất cả các bước cài đặt, không cần hỏi lại người dùng.

2. Kích hoạt môi trường:

```
conda activate streamlit_app
```

#### Cách 2: Dùng venv (nếu không dùng Anaconda)

1. Tạo môi trường ảo:

```
python3 -m venv streamlit_env
```

2. Kích hoạt môi trường:

```
source streamlit_env/bin/activate
```

#### 4.2.4 Cài đặt thư viện cần thiết

Sau khi kích hoạt môi trường, cài đặt các thư viện sau:

```
pip install streamlit pandas matplotlib openpyxl
```

- **streamlit**: streamlit xây dựng giao diện web tương tác cho ứng dụng Python.
- **pandas**: xử lý dữ liệu dạng bảng (DataFrame).
- **matplotlib**: vẽ biểu đồ, trực quan hóa dữ liệu.
- **openpyxl**: hỗ trợ đọc và ghi file Excel định dạng .xlsx.

#### 4.2.5 Thiết Kế Kiến Trúc Ứng Dụng

##### 1. Import các thư viện cần thiết

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import streamlit as st
```

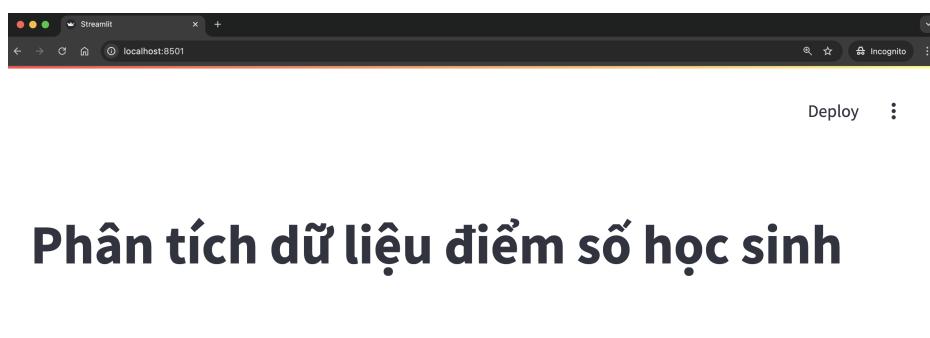
- **streamlit**: xây dựng giao diện web.
- **pandas**: đọc và xử lý dữ liệu từ file Excel.

- `matplotlib.pyplot`: vẽ biểu đồ (ở đây là biểu đồ tròn).

## 2. Tạo tiêu đề ứng dụng

```
st.title("Phân tích dữ liệu điểm số học sinh")
```

Sử dụng `st.title()` để tạo tiêu đề lớn trên giao diện web. Người dùng sẽ thấy dòng tiêu đề nổi bật khi mở ứng dụng.



Hình 4.3: Hiển thị tiêu đề

## 3. Người dùng tải file lên

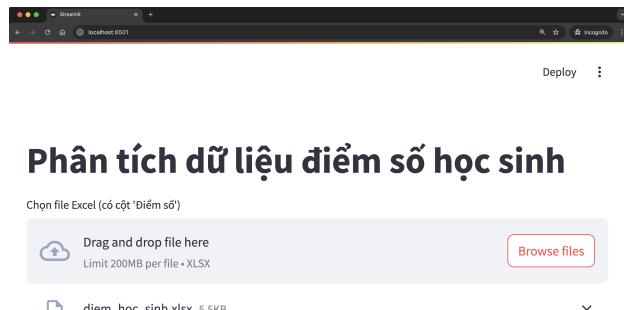
```
1 uploaded_file = st.file_uploader("Chọn file Excel (có cột 'Điểm số')
", type=["xlsx"])
```

`st.file_uploader()` tạo nút chọn file để người dùng tải file Excel từ máy tính.

Tải lên file .xlsx chứa cột Điểm số như hình: **Lưu ý:** File phải đúng định dạng .xlsx và chứa cột tên "Điểm số".

|   | A     | B       | C |
|---|-------|---------|---|
| 1 | Tên   | Điểm số |   |
| 2 | An    | 85      |   |
| 3 | Bình  | 78      |   |
| 4 | Chi   | 92      |   |
| 5 | Dung  | 67      |   |
| 6 | Em    | 59      |   |
| 7 | Phong | 91      |   |
| 8 | Huy   | 74      |   |

(a) Chuẩn bị File Excel như hình



(b) Giao diện chương trình sau khi tải file Excel lên

Hình 4.4: Upload file Excel lên giao diện

Kiểm tra nếu người dùng đã upload file chưa. Nếu người dùng đã tải file lên, ứng dụng sẽ tiếp tục xử lý các bước tiếp theo.

#### 4. Viết các hàm xử lý dữ liệu

##### a. Hàm tính điểm trung bình

```
1 def calculate_average(scores):
2 return sum(scores) / len(scores)
```

Hàm cộng tổng tất cả điểm rồi chia cho số lượng học sinh.

##### b. Hàm phân loại điểm số

```
1 def percentage_distribution(scores):
2 bins = {"90-100": 0, "80-89": 0, "70-79": 0, "60-69": 0, "<60": 0}
3
4 for score in scores:
5 if score >= 90:
6 bins["90-100"] += 1
7 elif score >= 80:
8 bins["80-89"] += 1
9 elif score >= 70:
10 bins["70-79"] += 1
```

```

10 elif score >= 60:
11 bins["60-69"] += 1
12 else:
13 bins["<60"] += 1
14
 return bins

```

Hàm chia điểm số thành 5 nhóm và đếm số lượng học sinh trong mỗi nhóm.

## 5. Đọc file Excel và hiển thị trung bình

```

1 # Đọc file
2 df = pd.read_excel(uploaded_file)
3
4 # Xử lý danh sách điểm số
5 scores = df["Điểm số"].dropna().astype(float).tolist()
6
7 # Hiển thị các chỉ số
8 st.write("Tổng số học sinh:", len(scores))
9 st.write("Điểm trung bình:", round(calculate_average(scores), 2))

```

## 6. Vẽ biểu đồ

Ta chuyển kết quả phân loại về dạng danh sách như sau:

```

1 labels = list(dist.keys())
2 values = list(dist.values())

```

Trong đó:

- **labels**: Danh sách tên của các nhóm điểm số (ví dụ: “90-100”, “80-89”, ...).
- **values** : Danh sách số lượng học sinh tương ứng với từng nhóm điểm số.

### Vẽ biểu đồ tròn

```

1 fig, ax = plt.subplots(figsize=(3, 3))
2 ax.pie(values, labels=labels, autopct="%1.1f%%", startangle=90)
3 ax.axis("equal")
4 st.pyplot(fig)

```

- **plt.subplots(figsize=(1, 1))**: Tạo biểu đồ với kích thước trung bình (1 inch × 1 inch).
- **ax.pie(values, labels=labels, autopct="%1.1f%", startangle=90)**: Vẽ biểu đồ tròn với:
  - **values**: Giá trị tương ứng với từng phần của biểu đồ.
  - **labels**: Tên nhóm tương ứng với từng phần của biểu đồ.
  - **autopct="%1.1f%"**: autopct là viết tắt của "automatic percentage", dùng để tự động hiển thị tỷ lệ phần trăm trên từng phần của biểu đồ. Định dạng "%1.1f%" chỉ định rằng phần trăm sẽ được hiển thị dưới dạng số thực với một chữ số sau dấu thập phân.
- **ax.axis("equal")**: Đảm bảo biểu đồ cân đối (tỉ lệ trực X và Y bằng nhau).
- **st.pyplot(fig)**: Hiển thị biểu đồ trên ứng dụng Streamlit.

### Hiển thị biểu đồ với độ phân giải cao

#### Cách 1: Vẽ trực tiếp bằng **st.pyplot(fig)**

```

1 st.pyplot(fig, use_container_width=False)
2 st.markdown("Biểu đồ phân bố điểm số.")

```

Cách này độ nét của hình ảnh phụ thuộc vào thông số **figsize** và độ phân giải mặc định. Điều này có thể dẫn đến hiện tượng hình ảnh bị mờ khi hiển thị trên giao diện Streamlit, như minh họa tại Hình 4.5a.

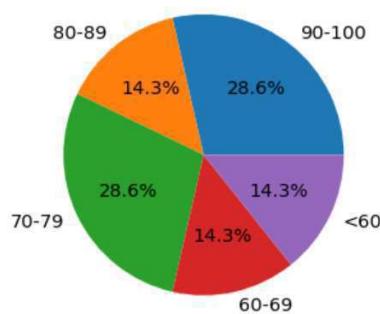
#### Cách 2: Lưu biểu đồ thành ảnh rồi hiển thị.

Phương pháp này lưu biểu đồ dưới định dạng PNG với độ phân giải cao (dpi=300 trước khi hiển thị bằng `st.image`). Điều này giúp đảm bảo hình ảnh rõ nét, đồng thời cho phép kiểm soát chính xác kích thước khi hiển thị (ví dụ, thiết lập `width=250px`). Do đó, cách 2 được khuyến khích sử dụng trong các ứng dụng yêu cầu chất lượng trình bày cao, như minh họa tại Hình 4.5b.

```

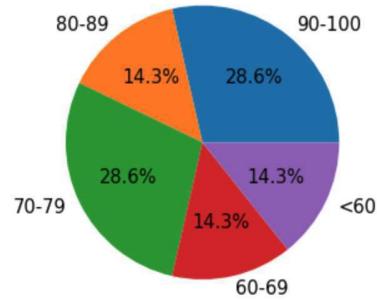
1 import io
2 from PIL import Image
3
4 buf = io.BytesIO()
5 fig.savefig(buf, format="png", dpi=300) # Lưu với dpi cao để ảnh sắc
 c nét
6 buf.seek(0)
7
8 st.markdown("Biểu đồ phân bố điểm số.")
9 img = Image.open(buf)
10 st.image(img, width=250) # Hiển thị ảnh với kích thước cố định

```



Biểu đồ phân bố điểm số.

(a) Cách 1



Biểu đồ phân bố điểm số.

(b) Cách 2 (khuyến khích sử dụng)

Hình 4.5: So sánh hình ảnh trước và sau xử lý.

**Thêm ghi chú nhỏ dưới biểu đồ**

```
1 st.markdown("Biểu đồ phân bố điểm số.")
```

### Đưa biểu đồ vào giữa giao diện:

Để biểu đồ hiển thị ở vị trí trung tâm thay vì căn trái mặc định, ta có thể sử dụng chức năng chia cột trong Streamlit.

Cụ thể, ta tạo ba cột với tỷ lệ [1, 2, 1], trong đó cột giữa có độ rộng lớn hơn để chứa biểu đồ. Sau đó, sử dụng `with col2:` để đặt biểu đồ vào cột giữa, đảm bảo nội dung được căn giữa giao diện.

```
1 col1, col2, col3 = st.columns([1, 2, 1]) # Tạo ba cột, cột giữa rộng hơn
2 with col2:
3 st.image(img, width=300) # Hiển thị biểu đồ ở cột giữa
4 st.markdown("Biểu đồ phân bố điểm số.")
```

### Hoàn thiện ứng dụng:

Sau khi hoàn thành các bước trên, ta thu được một tập tin app.py hoàn chỉnh, sẵn sàng để chạy trực tiếp trên nền tảng Streamlit.

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import streamlit as st
4 import io
5 from PIL import Image
6
7
8 # Tiêu đề ứng dụng
9 st.title("Phân tích dữ liệu điểm số học sinh")
10
11 # Upload file
12 uploaded_file = st.file_uploader("Chọn file Excel (có cột 'Điểm số')",
13 ", type=['xlsx']")
14
15 # Hàm tính điểm trung bình
16 def calculate_average(scores):
```

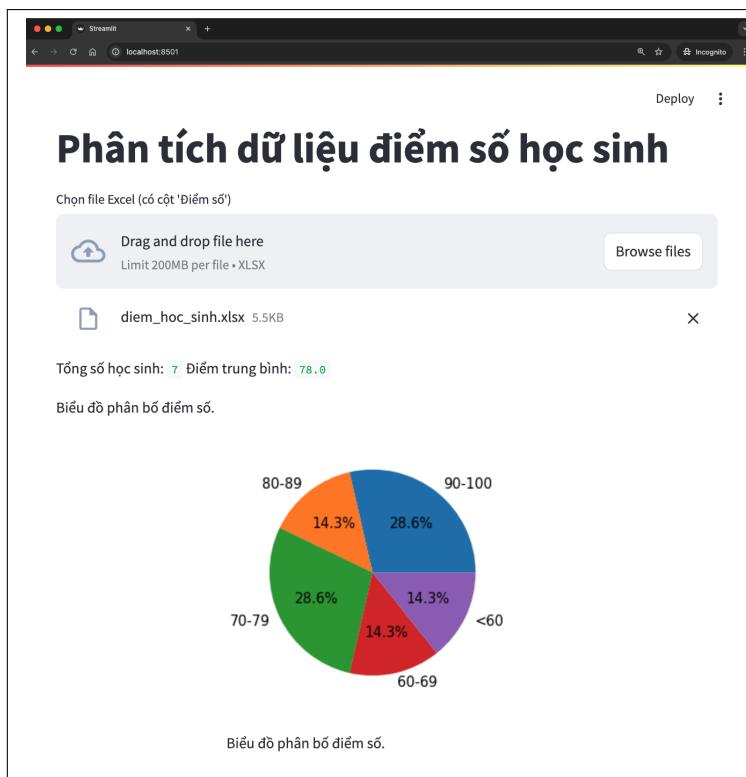
```
16 return sum(scores) / len(scores)
17
18 # Hàm phân loại điểm số
19 def percentage_distribution(scores):
20 bins = {"90-100": 0, "80-89": 0, "70-79": 0, "60-69": 0, "<60": 0}
21 for score in scores:
22 if score >= 90:
23 bins["90-100"] += 1
24 elif score >= 80:
25 bins["80-89"] += 1
26 elif score >= 70:
27 bins["70-79"] += 1
28 elif score >= 60:
29 bins["60-69"] += 1
30 else:
31 bins["<60"] += 1
32 return bins
33
34 # Khi có file
35 if uploaded_file:
36 df = pd.read_excel(uploaded_file)
37 scores = df["Điểm số"].dropna().astype(float).tolist()
38
39 if scores:
40 st.write("Tổng số học sinh:", len(scores), "Điểm trung bình:",
41 ", round(calculate_average(scores),
42 , 2))")
43 # Phân loại điểm
44 dist = percentage_distribution(scores)
45 labels = list(dist.keys())
46 values = list(dist.values())
47
48 fig, ax = plt.subplots(figsize=(1, 1))
49 ax.pie(
50 values,
51 labels=labels,
52 autopct="%1.1f%%",
53 textprops={"fontsize": 3.5},
54)
55 ax.axis("equal")
56 plt.tight_layout(pad=0.1)
57
58 buf = io.BytesIO()
```

```
57 fig.savefig(buf, format="png", dpi=300)
58 buf.seek(0)
59 st.markdown("Biểu đồ phân bố điểm số.")
60 img = Image.open(buf)
61
62 col1, col2, col3 = st.columns([1, 2, 1]) # Tạo ba cột, cột
63 # giữa rộng hơn
64 with col2:
65 st.image(img, width=300) # Hiển thị biểu đồ ở cột giữa
66 st.markdown("Biểu đồ phân bố điểm số.")
```

Sau khi viết xong code (ví dụ lưu thành file app.py), bạn chạy ứng dụng bằng

```
streamlit run app.py
```

Sau khi hoàn thành các bước xây dựng, giao diện người dùng (*UI*) của ứng dụng sẽ hiển thị như sau:



Hình 4.6: Giao diện ứng dụng Streamlit sau khi triển khai.

## 4.3 Xây dựng ứng dụng tính hàm toán học giai thừa

Trong phần này, chúng ta sẽ cùng nhau phân tích, thiết kế và triển khai ứng dụng tính giao thừa với streamlit.

### 4.3.1 Phân tích, thiết kế ứng dụng

Đầu tiên, cần phải hiểu rõ bài toán và yêu cầu của ứng dụng mà chúng ta sẽ xây dựng. Bước này tuy đơn giản nhưng rất quan trọng, càng mô tả chi tiết, sản phẩm chúng ta tạo ra sẽ càng tốt.

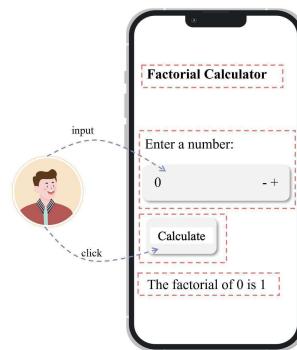
Chẳng hạn, mô tả bài toán có thể như sau: Chúng ta sẽ xây dựng một ứng dụng web để tính giao thừa của một số tự nhiên dành cho học sinh. Ứng dụng này sẽ có giao diện đơn giản, cho phép người dùng nhập số cần tính giao thừa. Sau đó, người dùng sẽ nhấn vào một nút tính toán và ứng dụng sẽ hiển thị kết quả là giao thừa của số tự nhiên mà người dùng đã nhập.

#### Phân tích yêu cầu bài toán

Với mô tả trên, chúng ta sẽ tiến hành phân tích yêu cầu bài toán này:

- Yêu cầu chức năng:
  - Nhập số tự nhiên: Người dùng có thể nhập vào một số tự nhiên để tính giao thừa.
  - Nút tính toán: Một nút bấm để kích hoạt chức năng tính giao thừa.
  - Hiển thị kết quả: Ứng dụng sẽ hiển thị kết quả giao thừa của số tự nhiên mà người dùng nhập vào.
- Yêu cầu phi chức năng:
  - Giao diện đơn giản và thân thiện: Ứng dụng cần có giao diện đơn giản để học sinh có thể dễ dàng sử dụng.
  - Hiệu suất: Ứng dụng phải tính toán và hiển thị kết quả một cách nhanh chóng.
  - Độ chính xác: Kết quả tính toán phải chính xác tuyệt đối.

## Thiết kế ứng dụng



Hình 4.7: Thiết kế giao diện ứng dụng

- Giao diện người dùng (UI):
  - Hiển thị tên ứng dụng
  - Nhập số: Một ô nhập văn bản (st.number\_input) để người dùng nhập số tự nhiên.
  - Nút tính toán: Một nút bấm (st.button) để bắt đầu tính toán giai thừa.
  - Hiển thị kết quả: Một vùng để hiển thị kết quả tính toán (st.write).
- Luồng hoạt động
  - Người dùng nhập số tự nhiên vào ô nhập văn bản.
  - Người dùng nhấn nút tính toán.
  - Ứng dụng tính giai thừa của số đã nhập.
  - Ứng dụng hiển thị kết quả lên màn hình.

### 4.3.2 Xây dựng ứng dụng

Đầu tiên, chúng ta sẽ tạo một dự án với cấu trúc như sau:

```

1 factorial-app
2 |---- app.py
3 |
4 |---- factorial.py
5 |
6 |---- requirements.txt

```

**factorial.py** là file chúng ta sẽ viết hàm tính giai thừa, file này là 1 module trong dự án.

```

1 # Factorial function
2 def fact(n):
3 if n == 0 or n == 1:
4 return 1
5 else:
6 return n * fact(n-1)

```

Trong chương trình trên, chúng ta tạo một hàm tên là `fact` với tham số `n`. Hàm này sẽ thực hiện tính giai thừa của số `n` với phương pháp đệ quy.

**app.py** là file chúng ta sẽ viết giao diện ứng dụng và xử lý tính toán tương tác với người dùng.

```

1 import streamlit as st
2 from factorial import fact
3
4 def main():
5 st.title("Factorial Calculator")
6 number = st.number_input("Enter a number:",
7 min_value=0,
8 max_value= 900)
9 if st.button("Calculate"):
10 result = fact(number)
11 st.write(f"The factorial of {number} is {result}")
12
13 if __name__ == "__main__":
14 main()

```

Trong chương trình trên, đầu tiên chúng ta khai báo các thư viện và

module. Ở đây chúng ta sử dụng thư viện streamlit và module factorial chúng ta xây dựng phía trên.

Tiếp theo chúng ta xây dựng giao diện ứng dụng, ta sử dụng st.title để đặt tiêu đề cho ứng dụng. Sau đó sử dụng st.number\_input tạo một ô nhập số với nhãn "Enter a number:", giá trị tối thiểu là 0 và giá trị tối đa là 900, vì với thuật toán tính giai thừa ở trên không tính được giai thừa của số lớn hơn 900, bạn có thể thay đổi thuật toán để tính được giai thừa cho các số lớn hơn.

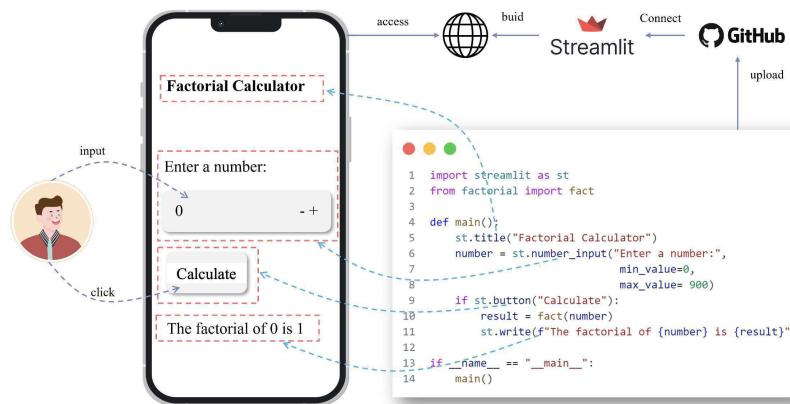
Cuối cùng chúng ta tạo một button calculate, khi sự kiện click vào button này là True thì sẽ thực hiện tính giai thừa cho số người dùng nhập vào và hiển thị kết quả ra màn hình với hàm st.write().

**requirements.txt** là file chứa các thư viện mà chúng ta cần cài đặt cho dự án.

Đến đây, ứng dụng có thể hoạt động trên máy của chúng ta. Để mở ứng dụng này chúng ta kích hoạt môi trường streamlit\_env đã tạo và sử dụng lệnh:

```
streamlit run app.py
```

## Triển khai ứng dụng



Hình 4.8: Triển khai ứng dụng

Trong phần này, chúng ta sẽ triển khai ứng dụng trên nền tảng github và streamlit cloud. Đầu tiên chúng ta truy cập vào github và đăng nhập, sau đó chúng ta tạo một repository mới có tên trùng với tên thư mục dự án là factorial-app.

Tiếp theo ta click vào upload an existing file để tải lên toàn bộ source code.

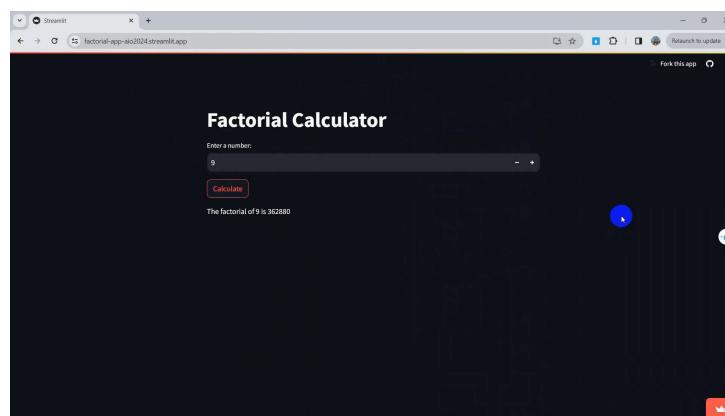
Sau khi upload thành công, chúng ta truy cập vào streamlit và tiến hành đăng nhập.

Tiếp theo chúng ta click vào create new app và thiết lập các thông tin để triển khai ứng dụng này.

Cuối cùng chúng ta click vào deploy và ứng dụng của chúng ta sẽ được triển khai thành công. Ứng dụng có thể truy cập bởi bất kỳ thiết bị nào có kết nối internet, bạn nhấn nút share và copy đường dẫn để chia sẻ ứng dụng của mình đến với bạn bè, người dùng của mình.

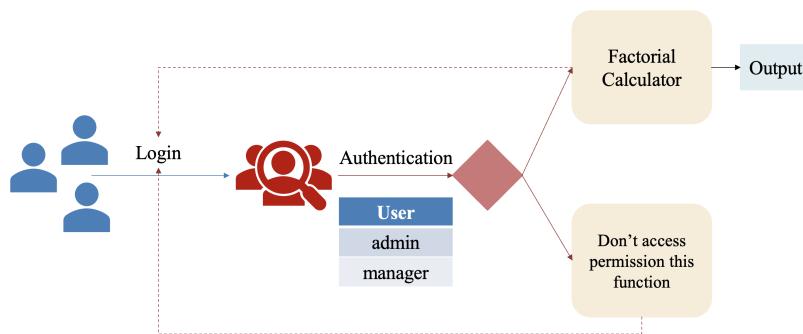
### 4.3.3 Xây dựng tính năng phân quyền và xác thực người dùng

Bên cạnh các tính năng tên, chúng ta có thể xây dựng chức năng đăng nhập, phân quyền:



Hình 4.9: Ứng dụng sau khi triển khai thành công

- Những người có quyền thuộc các tên có trong file user.txt sẽ có quyền truy cập tính năng tính giai thừa
- Những người không thuộc danh sách user thì không được phép truy cập



Hình 4.10: Xác thực và phân quyền người dùng

Đoạn code bổ sung tính năng sẽ thành:

```

1 import streamlit as st
2 from factorial import fact
3 import os
4

```

```
5 def load_users():
6 """Đọc danh sách user từ file user.txt"""
7 try:
8 if os.path.exists("user.txt"):
9 with open("user.txt", "r", encoding="utf-8") as f:
10 users = [line.strip() for line in f.readlines() if
11 line.strip()]
12 return users
13 else:
14 st.error("File user.txt không tồn tại!")
15 return []
16 except Exception as e:
17 st.error(f'Lỗi khi đọc file user.txt: {e}')
18 return []
19
20 def login_page():
21 """Trang đăng nhập"""
22 st.title("Đăng nhập")
23
24 # Input username
25 username = st.text_input("Nhập tên người dùng:")
26
27 if st.button("Đăng nhập"):
28 if username:
29 users = load_users()
30 if username in users:
31 st.session_state.logged_in = True
32 st.session_state.username = username
33 st.rerun()
34 else:
35 # Nếu user không hợp lệ, hiển thị trang chào hỏi
36 st.session_state.show_greeting = True
37 st.session_state.username = username
38 st.rerun()
39 else:
40 st.warning("Vui lòng nhập tên người dùng!")
41
42 def factorial_calculator():
43 """Trang tính giai thừa"""
44 st.title("Factorial Calculator")
45
46 # Hiển thị thông tin user đã đăng nhập
47 st.write(f"Xin chào, {st.session_state.username}!")
```

```
48 # Nút đăng xuất
49 if st.button("Đăng xuất"):
50 st.session_state.logged_in = False
51 st.session_state.username = ""
52 st.rerun()
53
54 st.divider()
55
56 # Chức năng tính giao thừa
57 number = st.number_input("Nhập vào một số:",
58 min_value=0,
59 max_value=900)
60
61 if st.button("Tính giao thừa"):
62 result = factorial(number)
63 st.write(f"Giai thừa của {number} là {result}")
64
65 def greeting_page():
66 """Trang chào hỏi cho user không hợp lệ"""
67 st.title("Xin chào!")
68 st.write(f"Xin chào {st.session_state.username}!")
69 st.write("Bạn không có quyền truy cập vào chức năng tính giao th
 ừa.")
70
71 if st.button("Quay lại đăng nhập"):
72 st.session_state.show_greeting = False
73 st.session_state.username = ""
74 st.rerun()
75
76 def main():
77 # Khởi tạo session state
78 if 'logged_in' not in st.session_state:
79 st.session_state.logged_in = False
80 if 'username' not in st.session_state:
81 st.session_state.username = ""
82 if 'show_greeting' not in st.session_state:
83 st.session_state.show_greeting = False
84
85 # Điều hướng trang dựa trên trạng thái đăng nhập
86 if st.session_state.logged_in:
87 factorial_calculator()
88 elif st.session_state.show_greeting:
89 greeting_page()
90 else:
```

```
91 login_page()
92
93 if __name__ == "__main__":
94 main()
```

Giao diện mới sẽ hiển thị như sau sau khi chạy lại app:

## Đăng nhập      Factorial Calculator

Nhập tên người dùng:

Đăng nhập

Xin chào, admin!

Đăng xuất

Nhập vào một số:

0

Tính giai thừa

Giai thừa của 0 là 1

Hình 4.11: Giao diện đăng nhập và tính năng

## 4.4 Câu hỏi trắc nghiệm

1. Hàm nào sau đây được sử dụng để hiển thị chuỗi văn bản trong streamlit.
  - (a) st.text(...)
  - (b) st.image(...)
  - (c) st.selectbox(...)
  - (d) st.slider(...)
2. Hàm nào sau đây trong streamlit sử dụng để người dùng nhập văn bản trên giao diện.
  - (a) st.text(...)
  - (b) st.multibox(...)
  - (c) st.audio(...)
  - (d) st.text\_input(...)
3. Khai báo nào sau đây là sai.
  - (a) st.image(image\_path, caption="A cat", width=100, channels="RGB")
  - (b) st.image(image\_path, caption="A cat", width=None, channels="BGR")
  - (c) st.image(image\_path, caption="A cat", width="RGB", channels="BGR")
  - (d) st.image(image\_path, caption="A cat", width=None, channels="RGB")
4. Hàm nào sau đây cho phép người dùng tải lên nhiều file.
  - (a) uploaded\_files = st.file\_uploader("Choose files", accept\_multiple\_files=True)
  - (b) uploaded\_files = st.upload\_file\_uploader("Choose files", accept\_multiple\_files=True)
  - (c) uploaded\_files = st.file\_uploader("Choose files")
  - (d) uploaded\_files = st.upload\_file\_uploader("Choose files")
5. Hàm nào sau đây không được sử dụng để hiển thị code trong streamlit?
  - (a) st.code(...)

- (b) st.echo(...)  
(c) st.markdown(...)  
(d) st.slider(...)
6. Dung lượng mặc định tối đa mỗi file được tải lên trong streamlit là bao nhiêu?  
(a) 100 MB  
(b) 200 MB  
(c) 300 MB  
(d) 400 MB
7. Khi sử dụng st.sidebar, điều gì sẽ xảy ra?  
(a) Hiển thị một bảng dữ liệu ở giữa trang  
(b) Thêm thông báo lỗi trong ứng dụng  
(c) Hiển thị biểu đồ dạng cột  
(d) Hiển thị widget trong thanh bên trái (sidebar)
8. Lệnh nào được dùng để chạy một ứng dụng Streamlit từ dòng lệnh?  
(a) python app.py  
(b) runstreamlit app.py  
(c) streamlit run app.py  
(d) launch streamlit app.py
9. Nếu bạn muốn hiển thị một biểu đồ Matplotlib hoặc Plotly trong Streamlit, bạn sẽ sử dụng hàm nào?  
(a) st.image()  
(b) st.dataframe()  
(c) st.pyplot() hoặc st.plotly\_chart()  
(d) st.audio()
10. Để tạo một thanh trượt (slider) trong Streamlit, bạn sử dụng hàm nào?  
(a) st.text\_area()

- (b) st.checkbox()
- (c) st.slider()
- (d) st.selectbox()

1. **Hint:** Các file code gợi ý có thể được tải [tại đây](#).
2. **Github:** Mã nguồn github để deploy có thể tham khảo thêm [tại đây](#)
3. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
4. **Rubric:**

| Streamlit - Rubric |                                                                                                                                   |                                                                                                                         |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| Câu                | Kiến Thức                                                                                                                         | Đánh Giá                                                                                                                |
| 1                  | <ul style="list-style-type: none"><li>- Làm quen với thư viện streamlit</li><li>- Hiểu rõ về thiết kế thành phần cơ bản</li></ul> | <ul style="list-style-type: none"><li>- Ứng dụng phân tích điểm số học sinh</li><li>- Ứng dụng tính giai thừa</li></ul> |
| 2                  | <ul style="list-style-type: none"><li>- Sử dụng tính năng phân quyền đơn giản</li><li>- Triển khai ứng dụng trên Cloud</li></ul>  | <ul style="list-style-type: none"><li>- Xây dựng giao diện với streamlit có tính năng đăng nhập</li></ul>               |

# Chương 5

## Project 2: Xây dựng Chatbot cá nhân nhanh chóng với RAG

### 5.1 Giới thiệu

**Large Language Models (LLMs)** (Tạm dịch: Các mô hình ngôn ngữ lớn) là loại mô hình cho phép người dùng nhập vào một văn bản với nội dung bất kì, thường sẽ là các yêu cầu hoặc câu hỏi. Từ đó, mô hình sẽ trả về câu trả lời dưới dạng văn bản thỏa mãn yêu cầu của người dùng. Các ứng dụng phổ biến nhất về LLMs có thể kể đến như ChatGPT, Gemini...

#### II. Bài toán Object Detection và các phiên bản YOLO đời trước

##### II.I. Bài toán Object Detection

Trong Computer Vision, bài toán Object Detection hướng dẫn xây dựng một chương trình có thể tự động xác định vị trí và nhận diện tên (class) của các vật thể trong một bức ảnh. Tổng hợp hai thông tin đầu ra này còn được gọi với tên là bounding box. Từ đây, ta có thể mô tả Input/Output của một chương trình Object Detection như sau:

- **Input:** Một bức ảnh.
- **Output:** Bounding box của các vật thể cần phát hiện trong ảnh.



Hình 3: Minh họa Input/Output của bài toán Object Detection.

Đến thời điểm hiện tại, các phương pháp sử dụng mạng Deep Learning cho thấy hiệu suất vượt trội. Ta có thể tóm tắt các hướng tiếp cận theo ba dạng như sau:

1. **One-stage Object Detection:** Việc xác định vị trí tọa độ và phân loại tên class của các vật thể được thực hiện trên một bước duy nhất. Diện hình cho hướng tiếp cận này có thể kể đến SSD [13] và YOLO [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].
2. **Two-stage Object Detection:** Việc xác định vị trí tọa độ và phân loại tên class của các vật thể được thực hiện riêng biệt. Diện hình cho hướng tiếp cận này có thể kể đến RCNN [14] và Faster RCNN [15].
3. **End-to-end Object Detection:** Việc xác định vị trí tọa độ và phân loại tên class của các vật thể được thực hiện bằng cách tích hợp cả hai bước trên. Diện hình cho hướng tiếp cận này có thể kể đến DETR [16], DINO [17], và DeFCN [18].

Question: Cho tôi danh sách một số phương pháp Object Detection theo hướng tiếp cận End-to-end Object Detection?

Helpful Answer:

- DETR [16]
- DINO [17]
- DeFCN [18]

Câu trả lời

source\_0

Detection [10]. Với những cải tiến mới, mô hình đã đạt được hiệu suất vượt trội hơn so với các phiên bản YOLO trước đó ở các khía cạnh khác nhau, tăng cường khả năng phát hiện đối tượng theo thời gian thực (real-time object detection).

Vùng thông tin truy vấn được

source\_1

kể đến SSD [13] và YOLO [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].  
2.Stage Object Detection: Việc xác định vị trí tọa độ và phân loại tên class của các vật thể được thực hiện riêng biệt. Diện hình cho hướng tiếp cận này có thể kể đến RCNN [14] và Faster RCNN [15].  
3.End-to-end Object Detection: Việc xác định vị trí tọa độ và phân loại tên class của các vật thể được thực hiện bằng cách tích hợp cả hai bước trên. Diện hình cho hướng tiếp cận này có thể kể đến DETR [16], DINO [17], và DeFCN [18].

Hình 5.1: Sử dụng RAG trong việc hỏi đáp tài liệu về YOLOv10 trong AIO.

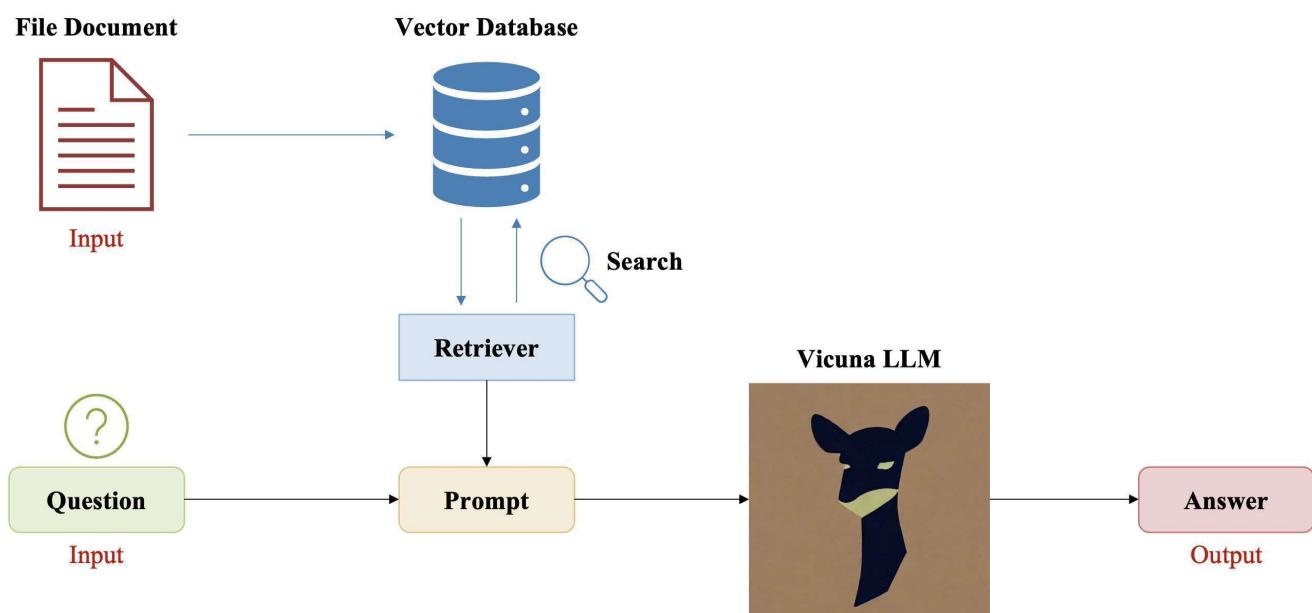
Trong LLMs, **Retrieval Augmented Generation (RAG)** là một kỹ thuật giúp LLMs cải thiện chất lượng kết quả tạo sinh bằng cách tích hợp nội dung truy vấn được từ một nguồn tài liệu nào đó để trả lời cho một câu hỏi đầu vào. Trong project này, chúng ta sẽ tìm hiểu cách xây dựng một chương trình RAG cơ bản. Đồng thời, ứng dụng chương trình vào việc hỏi đáp tài liệu bài học trong khóa AIO. Theo đó, Input và Output của chương trình là:

- **Input:** File tài liệu cần hỏi đáp và một câu hỏi liên quan đến nội dung

tài liệu.

- **Output:** Câu trả lời.

Tổng quan, luồng xử lý (pipeline) của chương trình RAG mà chúng ta sẽ xây dựng có dạng như sau:



Hình 5.2: Pipeline của chương trình RAG trong project.

## 5.2 Xây dựng chương trình

### 5.2.1 Chương trình RAG

Trước tiên, chúng ta cần nắm được các bước xử lý cơ bản trong RAG bằng cách xây dựng một chương trình RAG đơn giản. Tại đây, ta sẽ sử dụng thư viện LangChain để thực hiện việc này. Các bước triển khai như sau:



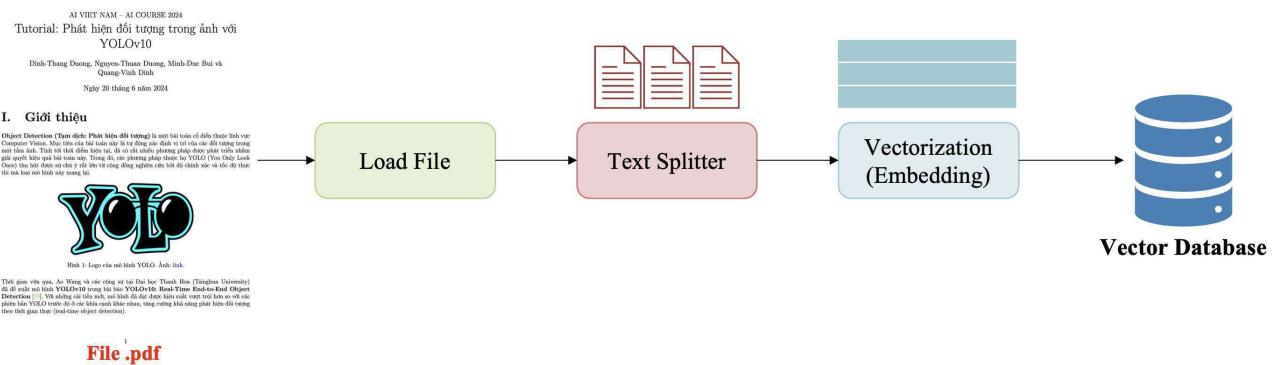
Hình 5.3: Thư viện LangChain. Một thư viện được thiết kế chuyên biệt cho việc xây dựng các ứng dụng về LLMs. Ảnh: [tại đây](#).

1. **Cài đặt các gói thư viện cần thiết:** Thực hiện cài đặt một số gói thư viện thông qua lệnh pip install như sau:

```
1 !pip install -q transformers==4.52.4
2 !pip install -q bitsandbytes==0.46.0
3 !pip install -q accelerate==1.7.0
4 !pip install -q langchain==0.3.25
5 !pip install -q langchainhub==0.1.21
6 !pip install -q langchain-chroma==0.2.4
7 !pip install -q langchain_experimental==0.3.4
8 !pip install -q langchain_community==0.3.24
9 !pip install -q langchain_huggingface==0.2.0
10 !pip install -q python-dotenv==1.1.0
11 !pip install -q pypdf
```

2. **Xây dựng vector database:** Để thực hiện truy vấn, chúng ta cần phải

có một cơ sở dữ liệu. Theo như nội dung project, với dữ liệu nguồn là một file pdf, chúng ta sẽ thực hiện đưa các nội dung trong file này vào cơ sở dữ liệu. Về các bước thực hiện, các bạn có thể coi qua ảnh sau:



Hình 5.4: Pipeline các bước thực hiện xây dựng vector database.

(a) **Import các thư viện cần thiết:** Để xây dựng vector database, chúng ta cần một số thư viện sau:

```

1 import torch
2
3 from transformers import BitsAndBytesConfig
4 from transformers import AutoTokenizer,
5 AutoModelForCausalLM,
6 pipeline
7
8 from langchain_huggingface import HuggingFaceEmbeddings
9 from langchain_huggingface.llms import
10 HuggingFacePipeline
11
12 from langchain.memory import ConversationBufferMemory
13 from langchain_community.chat_message_histories import
14 ChatMessageHistory
15
16 from langchain_community.document_loaders import
17 PyPDFLoader, TextLoader
18
19 from langchain.chains import ConversationalRetrievalChain
20
21 from langchain_experimental.text_splitter import
22 SemanticChunker
23
24
25 from langchain_chroma import Chroma

```

```

15 | from langchain_text_splitters import
 RecursiveCharacterTextSplitter
16 |
17 | from langchain_core.runnables import RunnablePassthrough
18 | from langchain_core.output_parsers import StrOutputParser
| from langchain import hub

```

- (b) **Đọc file pdf:** Từ một file pdf cho trước (trong bài này ta sẽ sử dụng file mô tả là bài hướng dẫn YOLOv10, các bạn có thể tải file này [tại đây](#)), ta sử dụng class PyPDFLoader để đọc file pdf này lên như sau:

```

1 Loader = PyPDFLoader
2 FILE_PATH = "./AIO-2024-All-Materials.pdf"
3 loader = Loader(FILE_PATH)
4 documents = loader.load()

```

- (c) **Khởi tạo mô hình embedding:** Để chuyển đổi các văn bản thành vector, ta sử dụng mô hình embedding. Mô hình này giúp biểu diễn văn bản dưới dạng các vector, từ đó giúp việc truy vấn và tìm kiếm thông tin trở nên chính xác hơn. Trong ví dụ này, chúng ta sử dụng mô hình HuggingFaceEmbeddings với mô hình `bkai-foundation-models/vietnamese-bi-encoder`, một mô hình được huấn luyện để xử lý ngữ nghĩa tiếng Việt.

Khởi tạo embedding model như sau:

```

1 embedding = HuggingFaceEmbeddings(
2 model_name="bkai-foundation-models/vietnamese-bi-
 encoder"
3)

```

Mô hình `embedding` này sẽ giúp chuyển các văn bản thành các vector. Các vector này sẽ được sử dụng trong quá trình **semantic chunking** để chia văn bản thành các phần nhỏ có ý nghĩa hơn, thay vì tách theo độ dài cố định.

- (d) **Khởi tạo bộ tách văn bản (text splitter):** Trước đây, khi sử

dụng phương pháp phân chia văn bản đơn giản, văn bản thường bị tách thành các phần nhỏ đồng đều mà không xem xét ngữ nghĩa. Tuy nhiên, để cải thiện chất lượng của các đoạn văn bản nhỏ, chúng ta chuyển sang sử dụng `SemanticChunker`, một kỹ thuật giúp tách văn bản dựa trên ngữ nghĩa, thay vì chỉ dựa vào độ dài đoạn văn.

```

1 semantic_splitter = SemanticChunker(
2 embeddings=embeddings,
3 buffer_size=1,
4 breakpoint_threshold_type="percentile",
5 breakpoint_threshold_amount=95,
6 min_chunk_size=500,
7 add_start_index=True
8)

```

**Semantic Chunking** giúp chia nhỏ văn bản thành các phần có ý nghĩa, giúp tăng tính chính xác khi thực hiện truy vấn. Các tham số trong `SemanticChunker` giúp điều chỉnh cách thức và cách tách các đoạn văn bản này:

- **buffer\_size=1**: Xác định số câu cần gom lại trước khi tách. Ở đây, giá trị `buffer_size=1` có nghĩa là mỗi nhóm bao gồm một câu, giúp mỗi chunk chứa các câu riêng biệt.
- **breakpoint\_threshold\_type="percentile"**: Tham số này chỉ định cách tính điểm phân đoạn. Trong trường hợp này, `percentile` có nghĩa là dựa trên tỷ lệ phần trăm của sự khác biệt về ngữ nghĩa giữa các đoạn văn. Tức là, nếu độ tương đồng của hai đoạn văn thấp hơn ngưỡng nhất định, chúng sẽ được tách ra làm các chunk riêng biệt.
- **breakpoint\_threshold\_amount=95**: Đây là ngưỡng phần trăm dùng để xác định khi nào nên cắt các đoạn văn bản. Giá trị 95 có nghĩa là nếu độ tương đồng giữa hai đoạn văn thấp hơn 95%, chúng sẽ được tách rời, tạo thành các chunk mới. Điều này đảm bảo rằng các đoạn văn bản không bị cắt ngắt một cách ngẫu nhiên mà vẫn giữ được sự liên kết ngữ nghĩa.
- **min\_chunk\_size=500**: Thiết lập kích thước tối thiểu của mỗi chunk. Khi giá trị này được đặt thành 500, mỗi đoạn văn bản

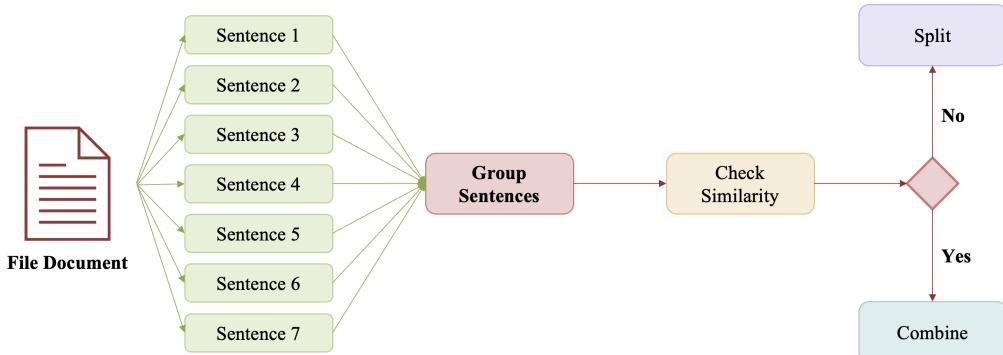
sẽ có ít nhất 500 ký tự. Điều này đảm bảo rằng các chunk không quá ngắn và vẫn chứa đủ thông tin để có thể sử dụng hiệu quả trong các tác vụ tìm kiếm hoặc truy vấn.

- **add\_start\_index=True:** Tham số này giúp đánh dấu vị trí của mỗi chunk trong văn bản gốc, tạo chỉ mục cho mỗi phần, giúp dễ dàng tra cứu và tổ chức các chunk sau khi đã được tách ra.

Với các tham số này, `SemanticChunker` giúp chia văn bản thành các phần có ý nghĩa và bảo đảm sự liên kết ngữ nghĩa trong mỗi chunk. Đoạn mã sau sẽ thực hiện quá trình này:

```
1 docs = semantic_splitter.split_documents(documents)
2 print("Number of semantic chunks: ", len(docs))
```

Kết quả của đoạn mã trên sẽ cho thấy số lượng chunk được tạo ra từ văn bản ban đầu, đồng thời mỗi chunk sẽ chứa một phần ngữ nghĩa đầy đủ từ tài liệu gốc.



Hình 5.5: Minh họa quy trình Semantic Chunking.

- (e) **Khởi tạo vector database:** Sau khi tách văn bản thành các chunk có ý nghĩa và biểu diễn chúng dưới dạng các vector với mô hình embedding, bước tiếp theo là khởi tạo một vector database để lưu trữ các vector của các chunk này. Ta sử dụng Chroma để khởi tạo vector database từ các documents đã được chia thành chunk và được vector hóa.

Mã thực hiện như sau:

```
1 vector_db = Chroma.from_documents(documents=docs,
2 embedding=embedding)
3
4 retriever = vector_db.as_retriever()
```

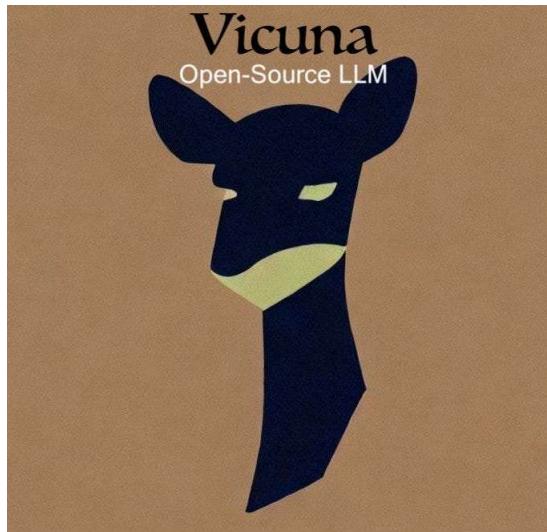
Kết quả của đoạn mã này là chúng ta đã tạo ra một vector database (`vector_db`) từ các chunk đã được vector hóa, và khởi tạo một `retriever` để có thể thực hiện các truy vấn từ cơ sở dữ liệu này.

Ta có thể thử thực hiện truy vấn với một đoạn văn bản bất kì tại đây:

```
1 result = retriever.invoke("What is YOLO?")
2
3 print("Number of relevant documents: ", len(result))
```

Kết quả của đoạn code này sẽ trả về cho ta danh sách các tài liệu có liên quan đến đoạn văn bản đầu vào.

3. **Khởi tạo mô hình ngôn ngữ lớn:** Trong project này, ta sẽ sử dụng mô hình Vicuna, một mô hình ngôn ngữ lớn nguồn mở có hiệu suất rất ổn, có thể phản hồi tốt với tiếng Việt. Các bạn có thể đọc thêm về mô hình Vicuna [tại đây](#).



Hình 5.6: Mô hình Vicuna. Ảnh: [link](#).

Một vấn đề của mô hình ngôn ngữ lớn đó là nó yêu cầu tài nguyên về phần cứng khá lớn. Vì vậy, để khởi tạo được mô hình Vicuna trên Colab, chúng ta sẽ cần thực hiện một số bước sau:

- (a) **Khai báo một số cài đặt cần thiết cho mô hình:**

```

1 nf4_config = BitsAndBytesConfig(
2 load_in_4bit=True,
3 bnb_4bit_quant_type="nf4",
4 bnb_4bit_use_double_quant=True,
5 bnb_4bit_compute_dtype=torch.bfloat16
6)
7

```

- (b) **Khởi tạo mô hình và tokenizer:**

```

1 MODEL_NAME = "lmsys/vicuna-7b-v1.5"
2
3 model = AutoModelForCausalLM.from_pretrained(
4 MODEL_NAME,
5 quantization_config=nf4_config,

```

```

6 low_cpu_mem_usage=True
7)
8
9 tokenizer = AutoTokenizer.from_pretrained(model_name)
10

```

- (c) Tích hợp tokenizer và model thành một pipeline để tiện sử dụng:

```

1 model_pipeline = pipeline(
2 "text-generation",
3 model=model,
4 tokenizer=tokenizer,
5 max_new_tokens=512,
6 pad_token_id=tokenizer.eos_token_id,
7 device_map="auto"
8)
9
10 llm = HuggingFacePipeline(
11 pipeline=model_pipeline,
12)
13

```

4. **Chạy chương trình:** Với vector database, retriever và mô hình Vicuna đã hoàn thiện. Ta sẽ kết hợp chúng lại để hoàn thành chương trình RAG đầu tiên của mình. Các bạn có thể test thử bằng cách đặt các câu hỏi có liên quan đến file tài liệu, câu trả lời của mô hình sẽ nằm ở phần "Answer:" trong output.

```

1 prompt = hub.pull("rlm/rag-prompt")
2
3 def format_docs(docs):
4 return "\n\n".join(doc.page_content for doc in docs)
5
6 rag_chain = (
7 {"context": retriever | format_docs, "question": RunnablePassthrough()}
8 | prompt

```

```

9 | llm
10 | StrOutputParser()
11)
12
13 USER_QUESTION = "YOLOv10 là gì?"
14 output = rag_chain.invoke(USER_QUESTION)
15 answer = output.split("Answer:")[1].strip()
16 print(answer)

```

Human: You are an assistant for question-answering tasks. Use the following pieces of retrieved context

Question: YOLO là gì?

Context: Quan sát kết quả trên, ta có thể thấy trên cùng một phiên bản, YOLOv10 có mức độ tối ưu tốt hơn về mặt tham số mô hình cũng như độ trễ trong inference trong khi vẫn giữ được độ chính xác ngang hoặc hơn so với các phiên bản trước.

18

khác.

- Ưu điểm : Cân bằng tốt giữa tốc độ và độ chính xác, dễ dàng sử dụng và triển khai.
- Nhược điểm : Yêu cầu phần cứng mạnh để đạt hiệu năng tối ưu.

5

AI VIETNAM (AIO2024) aivietnam.edu.vn

Hình 8: PGI và các kiến trúc tương tự. Ảnh: [9].

-Điểm mới : YOLOv9 sử dụng PGI và GELAN để cải thiện độ chính xác và hiệu suất của mô hình.

-Ưu điểm :

- + Kiến trúc tiên tiến: Sử dụng PGI và GELAN giúp mô hình duy trì thông tin quan trọng và tối ưu hóa quá trình huấn luyện, làm cho YOLOv9 trở nên mạnh mẽ và linh hoạt hơn trong nhiều ứng dụng khác nhau.
- + Tốc độ nhanh hơn: YOLOv9 có thể xử lý hình ảnh nhanh hơn so với YOLOv8 nhờ vào các cải tiến trong kiến trúc mạng.

-Nhược điểm :

- + Mặc dù nhanh hơn YOLOv8, YOLOv9 vẫn yêu cầu nhiều tài nguyên tính toán, đặc biệt là khi xử lý hình ảnh độ phân giải cao.
- ...

phiên bản YOLO trước đó ở các khía cạnh khác nhau, tăng cường khả năng phát hiện đối tượng theo thời gian thực (real-time object detection).

1

Answer: YOLO là một hệ thống mô hình máy học được sử dụng cho việc phát hiện đối tượng trong hình ảnh.

Hình 5.7: Minh họa kết quả đầu ra của chương trình.

Như vậy, chúng ta đã hoàn thành một chương trình Python về RAG, có khả năng hỏi đáp các nội dung trong một file pdf.

## 5.3 Xây dựng giao diện

Sau khi có chương trình RAG có thể trả về output đúng như dự định, ta có thể tích hợp vào với một giao diện chat để có được một ứng dụng chat hoàn chỉnh. Để xây dựng phần giao diện trong project này, chúng ta sẽ sử dụng thư viện streamlit.



Hình 5.8: Streamlit, một thư viện hỗ trợ tạo ứng dụng web nhanh chóng cho Python.

### 5.3.1 Cài đặt môi trường conda

Để xây dựng ứng dụng Streamlit một cách dễ dàng và tránh xung đột thư viện, ta nên tạo một môi trường ảo Conda.

```
conda create -n aio-rag python=3.11
```

Trong lệnh trên, các tham số được giải thích như sau:

- `conda create`: Lệnh dùng để tạo một môi trường ảo mới.
- `-n aio-rag`: Dùng để đặt tên cho môi trường ảo là `aio-rag`. Bạn có thể thay đổi tên này.
- `python=3.11`: Chỉ định phiên bản Python sẽ được cài đặt trong môi trường này là 3.11.

Kích hoạt môi trường:

```
conda activate aio-rag
```

### 5.3.2 Cài đặt thư viện cần thiết

Sau khi kích hoạt môi trường, ta cài đặt các môi trường sau:

```
pip install transformers==4.40.0
pip install langchain==0.1.20
pip install langchainhub==0.1.15
pip install langchain-chroma==0.1.8
pip install langchain_experimental==0.0.61
pip install langchain-community==0.0.38
pip install langchain_huggingface==0.0.3
pip install python-dotenv==1.0.0
pip install pypdf
pip install streamlit==1.36.0
```

Dưới đây là mô tả ngắn gọn về vai trò của từng nhóm thư viện trong dự án:

- **streamlit**: Xây dựng giao diện ứng dụng web tương tác.
- **transformers**: Load các mô hình LLM, tokenizer, pipeline.
- **langchain** (và các gói liên quan như langchainhub, langchain-chroma, langchain\_experimental, langchain-community, langchain\_huggingface):  
Bộ framework để xây dựng các ứng dụng dùng LLM, bao gồm kết nối dữ liệu (PDF), lưu trữ vector, và tích hợp các mô hình Hugging Face.

- python-dotenv: Quản lý các biến môi trường từ file .env.
- pypdf: Xử lý các file PDF (ví dụ: trích xuất văn bản).

Sau đó, ta tạo file code python và bắt đầu import các module cần thiết :

```

1 import streamlit as st
2 import tempfile
3 import os
4 import torch
5 from langchain_community.document_loaders import PyPDFLoader
6 from langchain_huggingface.embeddings import HuggingFaceEmbeddings
7 from langchain_experimental.text_splitter import SemanticChunker
8 from langchain_chroma import Chroma
9 from langchain_huggingface.llms import HuggingFacePipeline
10 from langchain import hub
11 from langchain_core.output_parsers import StrOutputParser
12 from langchain_core.runnables import RunnablePassthrough
13 from transformers import BitsAndBytesConfig, AutoModelForCausalLM,
 AutoTokenizer, pipeline

```

### 5.3.3 Khởi tạo Session State

Để đảm bảo ứng dụng Streamlit hoạt động ổn định và không bị tải lại models mỗi lần tương tác, ta cần khởi tạo các session state variables:

```

1 if "rag_chain" not in st.session_state:
2 st.session_state.rag_chain = None
3 if "models_loaded" not in st.session_state:
4 st.session_state.models_loaded = False
5 if "embeddings" not in st.session_state:
6 st.session_state.embeddings = None
7 if "llm" not in st.session_state:
8 st.session_state.llm = None

```

Các session state được khởi tạo bao gồm:

- **rag\_chain**: Lưu trữ RAG chain đã được xây dựng từ PDF.

- `models_loaded`: Kiểm tra xem các models đã được tải hay chưa.
- `embeddings`: Lưu trữ embedding model đã được tải.
- `llm`: Lưu trữ Large Language Model đã được tải.

### 5.3.4 Định nghĩa các hàm hỗ trợ

#### Hàm tải Embedding Model

Ta sử dụng decorator `@st.cache_resource` để cache model embeddings, tránh việc tải lại mỗi lần chạy:

```
1 @st.cache_resource
2 def load_embeddings():
3 return HuggingFaceEmbeddings(model_name="bkai-foundation-models/
4 vietnamese-bi-encoder")
```

Hàm này sử dụng model `vietnamese-bi-encoder` từ BKAI Foundation Models, được tối ưu hóa cho tiếng Việt.

#### Hàm tải Large Language Model

```
1 @st.cache_resource
2 def load_llm():
3 MODEL_NAME = "lmsys/vicuna-7b-v1.5"
4 nf4_config = BitsAndBytesConfig(
5 load_in_4bit=True,
6 bnb_4bit_quant_type="nf4",
7 bnb_4bit_use_double_quant=True,
8 bnb_4bit_compute_dtype=torch.bfloat16
9)
10 model = AutoModelForCausalLM.from_pretrained(
11 MODEL_NAME,
12 torch_dtype=torch.bfloat16,
13 low_cpu_mem_usage=True
14)
15 tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
16 model_pipeline = pipeline(
17 "text-generation",
```

```

18 model=model,
19 tokenizer=tokenizer,
20 max_new_tokens=512,
21 pad_token_id=tokenizer.eos_token_id,
22 device_map="auto"
23)
24 return HuggingFacePipeline(pipeline=model_pipeline)

```

Trong hàm này:

- MODEL\_NAME: Sử dụng model Vicuna 7B v1.5, một model mạnh mẽ cho generation tasks.
- torch\_dtype=torch.bfloat16: Sử dụng bfloat16 để tiết kiệm memory.
- low\_cpu\_mem\_usage=True: Tối ưu hóa việc sử dụng CPU memory.
- max\_new\_tokens=512: Giới hạn độ dài output tối đa là 512 tokens.
- device\_map="auto": Tự động phân bổ model lên GPU/CPU phù hợp.

## Hàm xử lý PDF

```

1 def process_pdf(uploaded_file):
2 with tempfile.NamedTemporaryFile(delete=False, suffix=".pdf") as
3 tmp_file:
4 tmp_file.write(uploaded_file.getvalue())
5 tmp_file_path = tmp_file.name
6
7 loader = PyPDFLoader(tmp_file_path)
8 documents = loader.load()
9
10 semantic_splitter = SemanticChunker(
11 embeddings=st.session_state.embeddings,
12 buffer_size=1,
13 breakpoint_threshold_type="percentile",
14 breakpoint_threshold_amount=95,
15 min_chunk_size=500,
16 add_start_index=True
17)

```

```

18 docs = semantic_splitter.split_documents(documents)
19 vector_db = Chroma.from_documents(documents=docs, embedding=st.
20 session_state.embeddings)
21 retriever = vector_db.as_retriever()
22
23 prompt = hub.pull("rlm/rag-prompt")
24
25 def format_docs(docs):
26 return "\n\n".join(doc.page_content for doc in docs)
27
28 rag_chain = (
29 {"context": retriever | format_docs, "question": RunnablePassthrough()}
30 | prompt
31 | st.session_state.llm
32 | StrOutputParser()
33)
34
35 os.unlink(tmp_file_path)
36 return rag_chain, len(docs)

```

Hàm process\_pdf thực hiện các bước sau:

- **Lưu file tạm:** Upload file từ Streamlit được lưu vào temporary file.
- **Load PDF:** Sử dụng PyPDFLoader để đọc nội dung PDF.
- **Semantic Chunking:** Chia tài liệu thành các chunks dựa trên ngữ nghĩa với các tham số:
  - buffer\_size=1: Kích thước buffer cho semantic chunking.
  - breakpoint\_threshold\_type="percentile": Sử dụng percentile để xác định breakpoint.
  - breakpoint\_threshold\_amount=95: Ngưỡng 95th percentile.
  - min\_chunk\_size=500: Kích thước chunk tối thiểu 500 ký tự.
- **Tạo Vector Database:** Sử dụng Chroma để lưu trữ embeddings của các chunks.
- **Xây dựng RAG Chain:** Kết hợp retriever, prompt template, và LLM thành một pipeline hoàn chỉnh.

- **Dọn dẹp:** Xóa temporary file sau khi xử lý xong.

### 5.3.5 Xây dựng giao diện người dùng

#### Cấu hình trang và tiêu đề

```

1 st.set_page_config(page_title="PDF RAG Assistant", layout="wide")
2 st.title("PDF RAG Assistant")
3
4 st.markdown("""
5 **Ứng dụng AI giúp bạn hỏi đáp trực tiếp với nội dung tài liệu PDF bằng tiếng Việt**
6 **Cách sử dụng đơn giản:**
7 1. **Upload PDF** -> Chọn file PDF từ máy tính và nhấn "Xử lý PDF"
8 2. **Đặt câu hỏi** -> Nhập câu hỏi về nội dung tài liệu và nhận câu trả lời ngay lập tức
9 ---
10 """)
```

#### Tải models

```

1 if not st.session_state.models_loaded:
2 st.info("Đang tải models...")
3 st.session_state.embeddings = load_embeddings()
4 st.session_state.llm = load_llm()
5 st.session_state.models_loaded = True
6 st.success("Models đã sẵn sàng!")
7 st.rerun()
```

Đoạn code này kiểm tra và tải models chỉ một lần duy nhất khi ứng dụng được khởi chạy.

## Upload và xử lý PDF

```
1 uploaded_file = st.file_uploader("Upload file PDF", type="pdf")
2 if uploaded_file and st.button("Xử lý PDF"):
3 with st.spinner("Đang xử lý..."):
4 st.session_state.rag_chain, num_chunks = process_pdf(
5 uploaded_file)
6 st.success(f"Hoàn thành! {num_chunks} chunks")
```

## Giao diện hỏi đáp

```
1 if st.session_state.rag_chain:
2 question = st.text_input("Đặt câu hỏi:")
3 if question:
4 with st.spinner("Đang trả lời..."):
5 output = st.session_state.rag_chain.invoke(question)
6 answer = output.split("Answer:") [1].strip() if "Answer:"
7 in output else output.strip()
8 st.write("**Trả lời:**")
9 st.write(answer)
```

Phần này cho phép người dùng đặt câu hỏi và nhận câu trả lời từ RAG system. Câu trả lời được xử lý để loại bỏ phần "Answer:" nếu có trong output của model.

### 5.3.6 Chạy ứng dụng và demo từng bước

#### Khởi chạy ứng dụng Streamlit

Để khởi chạy ứng dụng, ta sử dụng lệnh sau trong terminal:

```
streamlit run app.py
```

Trong đó `app.py` là tên file chứa code ứng dụng của chúng ta.

#### Bước 1: Tải Embedding Model

Khi ứng dụng được khởi chạy lần đầu tiên, hệ thống sẽ tự động tải embedding model. Giao diện sẽ hiển thị thông báo như trong Hình 5.9.



Hình 5.9: Giao diện khi đang tải embedding model

Quá trình này có thể mất vài phút tùy thuộc vào tốc độ internet và cấu hình máy tính. Model embedding `vietnamese-bi-encoder` sẽ được tải từ Hugging Face Hub.

#### Bước 2: Tải Large Language Model

Sau khi embedding model được tải xong, hệ thống tiếp tục tải LLM (Vicuna 7B). Đây là bước tốn thời gian nhất do kích thước model lớn (khoảng 14GB).

Quá trình này được thể hiện trong Hình 5.10.



Hình 5.10: Giao diện khi đang tải Large Language Model

### Bước 3: Upload và xử lý PDF



Hình 5.11: Quá trình upload và xử lý PDF

Sau khi models sẵn sàng, người dùng có thể upload file PDF và nhấn "Xử lý PDF". Quá trình xử lý sẽ hiển thị spinner "Đang xử lý..." và khi hoàn thành sẽ thông báo số chunks đã tạo như trong Hình 5.11.

Quá trình này bao gồm: đọc PDF, semantic chunking, tạo embeddings, và xây dựng RAG chain.

**Bước 4: Hỏi đáp với tài liệu**

Sau khi xử lý PDF thành công, người dùng có thể đặt câu hỏi và nhận câu trả lời ngay lập tức. Giao diện sẽ hiển thị "Đang trả lời..." khi xử lý và sau đó hiển thị câu trả lời định dạng như trong Hình 5.13.



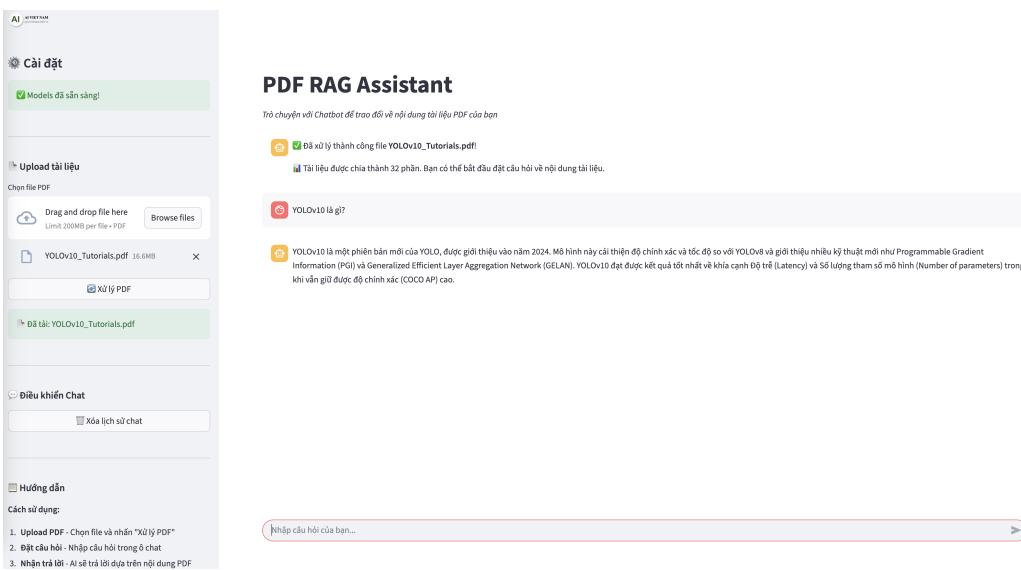
Hình 5.12: Giao diện hỏi đáp với tài liệu PDF

Người dùng có thể tiếp tục đặt nhiều câu hỏi khác nhau mà không cần xử lý lại PDF.

### 5.3.7 Xây dựng khung chat

Mở rộng từ đoạn mã trên, chúng ta có thể xây dựng khung chat để tương tác liên tục với mô hình, có thể tham khảo từ code sau [tại đây](#)

Giao diện mới thu được như sau:



Hình 5.13: Giao diện chatbot mở rộng hỏi đáp với tài liệu PDF

## 5.4 Câu hỏi trắc nghiệm

1. Tại sao phải sử dụng `@st.cache_resource` cho việc load models trong Streamlit?
  - (a) Để tránh reload models mỗi khi người dùng tương tác với giao diện.
  - (b) Để giảm thời gian phản hồi của ứng dụng.
  - (c) Để chia sẻ models giữa các session khác nhau.
  - (d) Cả A và C đều đúng.
2. Trong `SemanticChunker`, `breakpoint_threshold_amount=95` có nghĩa là gì?
  - (a) Chia tài liệu thành 95 chunks.
  - (b) Sử dụng 95% nội dung để tạo chunks.
  - (c) Chỉ tạo breakpoint khi similarity < 95th percentile.
  - (d) Minimum chunk size là 95 tokens.
3. Tại sao sử dụng `torch_dtype=torch.bfloat16` thay vì `float32`?
  - (a) Tăng tốc độ inference.
  - (b) Giảm 50% memory usage.
  - (c) Tương thích tốt hơn với GPU.
  - (d) Tất cả các lý do trên.
4. Trong RAG chain, `RunnablePassthrough()` có vai trò gì?
  - (a) Truyền question trực tiếp qua các bước tiếp theo.
  - (b) Lọc và làm sạch input question.
  - (c) Chuyển đổi question thành embedding.
  - (d) Kết hợp question với context.
5. Trong `SemanticChunker`, `buffer_size=1` có ý nghĩa gì?
  - (a) Số lượng câu được so sánh khi tìm breakpoint.

- (b) Kích thước tối thiểu của mỗi chunk.  
(c) Số lượng chunks được xử lý cùng lúc.  
(d) Độ dài context window cho embedding.
6. Tại sao phải sử dụng `st.rerun()` sau khi load models xong?  
(a) Để refresh toàn bộ session state.  
(b) Để cập nhật UI sau khi thay đổi session state.  
(c) Để giải phóng memory đã sử dụng.  
(d) Để reset cache của `@st.cache_resource`.
7. Tham số `max_new_tokens=512` có ý nghĩa gì?  
(a) Giới hạn độ dài input context.  
(b) Số tokens tối đa model có thể sinh ra.  
(c) Kích thước vocabulary của model.  
(d) Số tokens xử lý song song.
8. Tại sao phải tạo temporary file khi xử lý PDF upload?  
(a) Streamlit không thể đọc trực tiếp file upload.  
(b) PyPDFLoader yêu cầu file path thật.  
(c) Để tránh conflict với session state.  
(d) Để backup file trước khi xử lý.
9. Prompt "rlm/rag-prompt" từ LangChain Hub có cấu trúc như thế nào?  
(a) Chỉ chứa context và question.  
(b) Context + Question + Answer format.  
(c) System prompt + Context + Question.  
(d) Question + Context + Instructions.
10. Tại sao phải xử lý `output.split("Answer:")[1].strip()`?  
(a) Loại bỏ phần context trong output.

- (b) Tách phần trả lời từ format prompt.
- (c) Làm sạch whitespace thừa.
- (d) Cả B và C đều đúng.

1. **Hint:** Các file code gợi ý có thể được tải [tại đây](#).
2. **Github:** Mã nguồn github để deploy có thể tham khảo thêm [tại đây](#)
3. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
4. **Kiến thức cần cho project:**

Để hoàn thành và hiểu rõ phần project một cách hiệu quả nhất, các học viên cần nắm rõ các kiến thức được trang bị

- Kiến thức lập trình Python trong các buổi học chính trong Module 1 của khoá học
- Sử dụng công cụ Streamlit trong Project 1.1-Web and Streamlit for Simple Deployment của Module 1
- Thao tác cơ bản với git và github để quản lý code và tương tác với Streamlit trong các buổi học nâng cao vào thứ 5 hoặc thứ 7

#### 5. Đề xuất phương pháp cải tiến project:

Project xây dựng chatbot hỏi đáp đơn giản thông qua kiến trúc của RAG. Để tiếp tục cải tiến, tối ưu hệ thống; nhóm tác biên soạn gợi ý một số phương pháp như sau:

- Phương pháp tách đoạn (Chunking): Gợi ý cải tiến phương pháp chia đoạn để tăng tốc độ xử lý và hiệu quả khi chia đoạn
- Mô hình nhúng (Embedding Model): Gợi ý sử dụng hoặc tối ưu các mô hình nhúng từ để có phương pháp biểu diễn từ tốt hơn, tối ưu khả năng truy vấn thông tin, đặc biệt cho ngôn ngữ tiếng việt
- Mô hình ngôn ngữ lớn (LLMs): Bên cạnh mô hình đề xuất, có thể sử dụng những mô hình khác hiệu quả hơn, hoặc sử dụng những kỹ thuật như lượng tử hoá, chắt lọc tri thức,... để tăng tốc độ xử lý, tăng hiệu quả và giảm kích thước của LLMs
- Bộ nhớ dài hạn cho hệ thống: Xây dựng và mở rộng hệ thống lưu trữ những câu hỏi đã được trả lời để tái sử dụng hoặc tận dụng các đáp án có sẵn

- Tri thức cho truy vấn: Bổ sung thêm những dạng tri thức biểu diễn khác cho văn bản hoặc kết hợp thêm các phương pháp tìm kiếm khác nhau như Bm25 kết hợp cosine similarity,...

Trên đây là một số gợi ý, giúp học viện có thể cải tiến thêm hệ thống để đạt hiệu quả tốt hơn về cả tốc độ xử lý và độ chính xác truy vấn, trả lời câu hỏi.

### 6. Rubric:

| Mục  | Kiến Thức                                                                                                                                                                        | Đánh Giá                                                                                                                                                                                                                                                                                                                                   |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| II.  | <ul style="list-style-type: none"> <li>- Kiến thức về sử dụng Google Colab.</li> <li>- Kiến thức cơ bản về LLMs.</li> <li>- Kiến thức cơ bản về một chương trình RAG.</li> </ul> | <ul style="list-style-type: none"> <li>- Nắm được khái niệm về bài toán RAG.</li> <li>- Hiểu được nguyên lý hoạt động của chương trình hỏi đáp tài liệu bài học AIO sử dụng RAG.</li> <li>- Biết cách sử dụng Google Colab để triển khai một ứng dụng về RAG.</li> <li>- Biết cách sử dụng một mô hình ngôn ngữ lớn trên Colab.</li> </ul> |
| III. | <ul style="list-style-type: none"> <li>- Kiến thức về Streamlit.</li> </ul>                                                                                                      | <ul style="list-style-type: none"> <li>- Biết cách sử dụng Streamlit để thiết kế một giao diện web đơn giản nhằm phục vụ cho demo RAG.</li> </ul>                                                                                                                                                                                          |

## **Phần II**

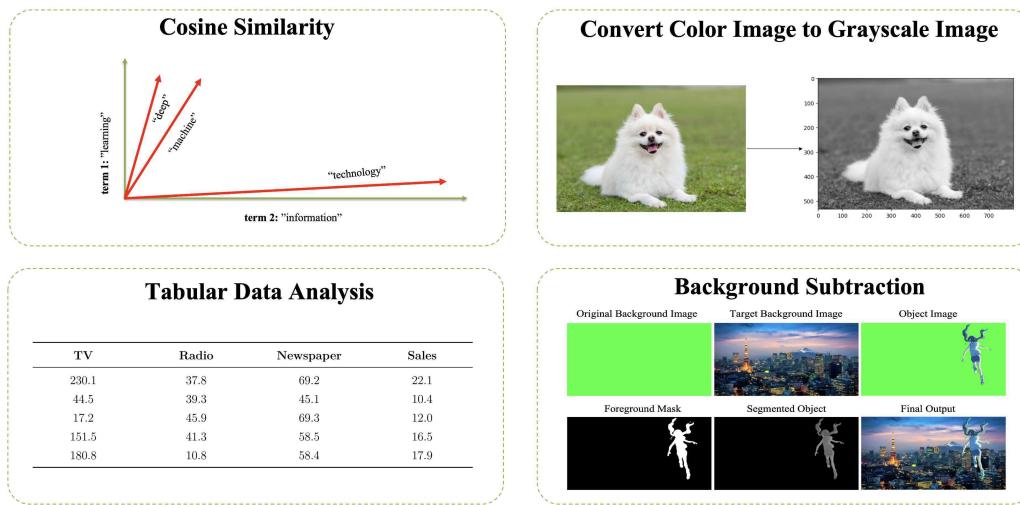
### **Module 2: Toán nâng cao và lập trình nâng cao**

# Chương 6

## Numpy và đại số tuyến tính

### 6.1 Giới thiệu

**Giới thiệu về bài tập:** Ở phần bài tập này, ta sẽ học cách sử dụng thư viện numpy và ôn lại kiến thức đại số tuyến tính cơ bản



Hình 6.1: Minh họa nội dung bài tập.

## 6.2 Câu hỏi tự luận

### 6.2.1 Các phép toán trên vector và ma trận

#### 1. Độ dài của vector:

Length of a vector

- Vector:  $\mathbf{v} = [v_1, v_2, \dots, v_n]^T$
- Length of a vector:  $\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + \dots + v_n^2}$

Hãy hoàn thiện hàm `compute_vector_length()` để tính độ dài của vector sử dụng thư viện numpy:

```

1 import numpy as np
2 def compute_vector_length(vector):
3 # ***** Your code here *****
4
5 return len_of_vector

```

#### 2. Phép tích vô hướng:

Dot product

$$\bullet \text{ Vector: } \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \dots \\ u_n \end{bmatrix}$$

- Dot Product:  $\mathbf{v} \cdot \mathbf{u} = v_1 * u_1 + v_2 * u_2 + \dots + v_n * u_n$

Hãy hoàn thiện hàm `compute_dot_product()` sử dụng thư viện numpy:

```

1 def compute_dot_product(vector1, vector2):
2 # ***** Your code here *****
3

```

```
4 return result
```

### 3. Nhân vector với ma trận:

#### Multiplying a vector by a matrix

- Matrix:  $\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}, \mathbf{A} \in R^{m*n}$
- Vector:  $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix}, \mathbf{v} \in R^n$
- $\mathbf{c} = \mathbf{Av} = \begin{bmatrix} a_{11} * v_1 + \dots + a_{1n} * v_n \\ \dots \\ a_{m1} * v_1 + \dots + a_{mn} * v_n \end{bmatrix}, \mathbf{c} \in R^m$

Hãy hoàn thiện hàm **matrix\_multi\_vector()** sử dụng thư viện numpy:

```
1 def matrix_multi_vector(matrix, vector):
2 # **** Your code here ****
3
4 return result
```

### 4. Nhân ma trận với ma trận:

## Multiplying a matrix by a matrix

- Matrix A:  $\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}, \mathbf{A} \in R^{m*n}$
- Matrix B:  $\mathbf{B} = \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{nk} \end{bmatrix}, \mathbf{B} \in R^{n*k}$
- $\mathbf{C} = \mathbf{AB} = \begin{bmatrix} a_{11} * b_{11} + \dots + a_{1n} * b_{n1} & \dots & a_{11} * b_{1k} + a_{1n} * b_{nk} \\ \dots & \dots & \dots \\ a_{m1} * b_{11} + \dots + a_{mn} * b_{n1} & \dots & a_{m1} * b_{1k} + a_{mn} * b_{nk} \end{bmatrix}, \mathbf{C} \in R^{m*k}$

Hãy hoàn thiện hàm `matrix_multi_matrix()` sử dụng thư viện numpy:

```

1 import numpy as np
2 def matrix_multi_matrix(matrix1, matrix2):
3 # **** Your code here ****
4
5 return result

```

## 6.2.2 Cosine Similarity

## Cosine Similarity

- Data (vector  $\mathbf{x}, \mathbf{y}$ ):  $\mathbf{x} = \{x_1, \dots, x_N\}$   $\mathbf{y} = \{y_1, \dots, y_N\}$
- Cosine Similarity:  $cs(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} = \frac{\sum_i^n x_i y_i}{\sqrt{\sum_i^n x_i^2} \sqrt{\sum_i^n y_i^2}}$

Cho trước  $\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$   $\mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 3 \\ 0 \end{bmatrix}$  Tìm Cosine similarity  $cs(\mathbf{x}, \mathbf{y})$ .

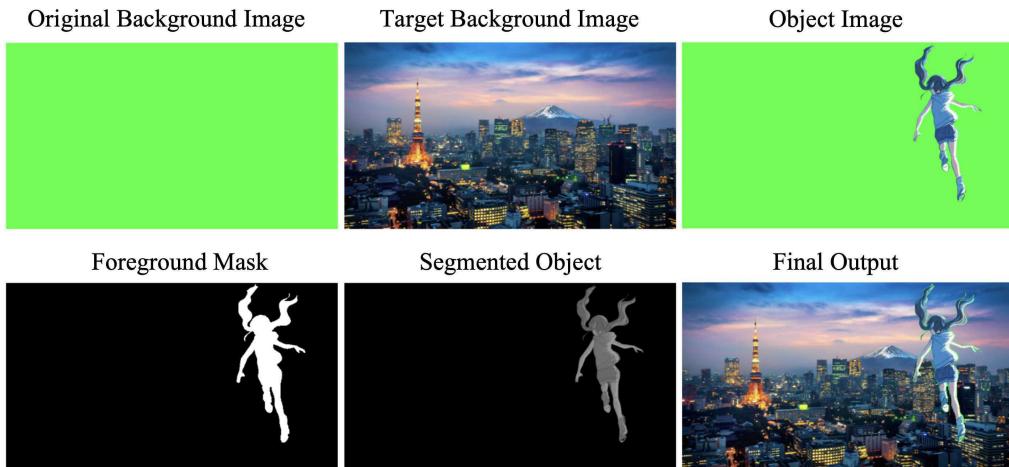
Dựa vào trên quả tính tay ở trên, hãy hoàn thiện hàm **compute\_cosine()** sử dụng thư viện numpy:

```

1 def compute_cosine(v1, v2):
2 # ***** Your code here *****
3
4 return cos_sim

```

### 6.2.3 Background subtraction (tách nền)



Hình 6.2: Kết quả trích xuất foreground (object mong muốn) và dán vào background mới

Viết chương trình sử dụng kỹ thuật background subtraction để trích xuất foreground (object mong muốn) và dán vào background mới. Input/Output của chương trình như sau:

- **Input:** Gồm 3 ảnh là Original Background Image (Greenscreen), Target Background Image và Object Image.
- **Output:** Ảnh mới khi trích xuất object từ Object Image và dán vào Target Background Image.

**Gợi ý:**

1. Đưa cả 3 ảnh về cùng kích thước.
2. Dùng kỹ thuật background subtraction với Object Image và Original Background Image để lấy mask của object.
3. Mask object là một ảnh binary (Foreground Mask), sẽ gồm 2 giá trị: 0 là background, 1 các vùng pixel chứa object.
4. Tạo ra ảnh output bằng cách: tại vị trí pixel nào = 1 thì lấy giá trị của Object Image và vị trí nào = 0 thì lấy giá trị của Target Background Image.

**Hướng dẫn:** Để hoàn thành bài tập trên, bạn cần hoàn thiện các hàm sau đây (các bạn chú ý chỉnh lại đường dẫn đến ảnh cho phù hợp):

**1. Resize các ảnh đầu vào về cùng kích thước:**

```
1 import numpy as np
2 from google.colab.patches import cv2_imshow
3 import cv2
4
5 bg1_image = cv2.imread("GreenBackground.png", 1)
6 bg1_image = cv2.resize(bg1_image, (678, 381))
7
8 ob_image = cv2.imread("Object.png", 1)
9 ob_image = cv2.resize(ob_image, (678, 381))
10
11 bg2_image = cv2.imread("NewBackground.jpg", 1)
12 bg2_image = cv2.resize(bg2_image, (678, 381))
```

**2. Xây dựng hàm compute\_difference():**

```
1 def compute_difference(bg_img, input_img):
2 # **** Your code here ****
3
4 return difference_single_channel
```

Các bạn có thể sử dụng đoạn code bên dưới để hiển thị kết quả (hình 2) của hàm **compute\_difference()** như sau:

```
1 difference_single_channel = compute_difference(bg1_image,
 ob_image)
2 cv2_imshow(difference_single_channel)
```

### 3. Xây dựng hàm **compute\_binary\_mask()**:

```
1 def compute_binary_mask(difference_single_channel):
2 # ***** Your code here *****
3
4 return difference_binary
```

Các bạn có thể sử dụng đoạn code bên dưới để hiển thị kết quả (hình 3) của hàm **compute\_binary\_mask()** như sau:

```
1 binary_mask = compute_binary_mask(difference_single_channel)
2 cv2_imshow(binary_mask)
```

### 4. Xây dựng hàm **replace\_background()**:

```
1 def replace_background(bg1_image, bg2_image, ob_image):
2 difference_single_channel = compute_difference(
3 bg1_image,
4 ob_image
5)
6
7 binary_mask = compute_binary_mask(
8 difference_single_channel)
9
10 output = np.where(binary_mask==255, ob_image, bg2_image)
11
12 return output
```

## 6.3 Câu hỏi trắc nghiệm

### 6.3.1 Numpy

Câu hỏi 1: Tính độ dài của vector:

```
1 import numpy as np
2
3 def compute_vector_length(vector):
4 return # Your code here
5
6 vector = np.array([-2, 4, 9, 21])
7 result = compute_vector_length(vector)
8 print(round(result, 2))
```

Kết quả là gì?

- 1 (a) 43.28
- 2 (b) 33.28
- 3 (c) 13.28
- 4 (d) 23.28

Câu hỏi 2: Tính tích vô hướng:

```
1 import numpy as np
2
3 def compute_dot_product(vector1, vector2):
4 return # Your code here
5
6 v1 = np.array([0, 1, -1, 2])
7 v2 = np.array([2, 5, 1, 0])
8 result = compute_dot_product(v1, v2)
9 print(round(result, 2))
```

Kết quả là gì?

- 1 (a) 3
- 2 (b) 4
- 3 (c) 5
- 4 (d) 6

Câu hỏi 3: Nhân vector với ma trận:

```

1 import numpy as np
2
3 def matrix_multi_vector(matrix, vector):
4 return # Your code here
5
6 matrix = np.array([-1, 1, 1], [0, -4, 9])
7 vector = np.array([0, 2, 1])
8 result = matrix_multi_vector(matrix, vector)
9 print(result)

```

Kết quả là gì?

- 1 (a) [3 1]
- 2 (b) [1 3]
- 3 (c) [2 3]
- 4 (d) [3 2]

Câu hỏi 4: Nhân ma trận với ma trận:

```

1 import numpy as np
2
3 def matrix_multi_matrix(matrix1, matrix2):
4 return # Your code here
5
6 m1 = np.array([0, 1, 2], [2, -3, 1])
7 m2 = np.array([1, -3], [6, 1], [0, -1])
8 result = matrix_multi_matrix(m1, m2)
9 print(result)

```

Kết quả là gì?

- 1 (a) [[9 -1], [-16 -10]]
- 2 (b) [[8 -1], [-16 -10]]
- 3 (c) [[6 -1], [-16 -10]]
- 4 (d) [[7 -1], [-16 -10]]

Câu hỏi 5: Cosine Similarity:

```

1 import numpy as np
2
3 def compute_cosine(v1, v2):
4 dot_product = np.dot(v1, v2)
5 norm_v1 = # **** Your code here ****
6 norm_v2 = # **** Your code here ****
7 return dot_product / (norm_v1 * norm_v2)
8
9 x = np.array([1, 2, 3, 4])
10 y = np.array([1, 0, 3, 0])
11 result = compute_cosine(x, y)
12 print(round(result, 3))

```

Kết quả là gì?

- 1 (a) 1.577
- 2 (b) 2.577
- 3 (c) 0.577
- 4 (d) 3.577

Câu hỏi 6: Lọc phần tử lẻ:

```

1 import numpy as np
2 arr = np.arange(0, 10)
3 # Your code here

```

Kết quả là gì?

- 1 (a) [1 3 5 7 9]
- 2 (b) [1 2 3 4 5]
- 3 (c) [2 4 6 8 9]
- 4 (d) [1 7 5 7 9]

Câu hỏi 7: Thay thế phần tử lẻ bằng giá trị -1:

```

1 import numpy as np
2 arr = np.arange(0, 10)
3 # Your code here = -1
4 print(arr)

```

Kết quả là gì?

- ```

1 (a) [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
2 (b) [0 -1 2 -1 4 -1 6 -1 8 -1]
3 (c) [-1 0 -1 0 -1 0 -1 0 -1 0]
4 (d) [-1 1 -1 1 -1 1 -1 1 -1 1]

```

Câu hỏi 8: Gộp mảng theo trục axis=0:

```

1 import numpy as np
2 arr1 = np.arange(10).reshape(2, -1)
3 arr2 = np.ones((2, 5), dtype=int)
4 c = # Your code here
5 print(c)

```

Kết quả là gì?

- ```

1 (a) [[0 1 2 3 4] [5 6 7 8 9] [1 1 1 1 1] [1 1 1 1 1]]
2 (b) [[0 1 2 3 4] [5 6 7 8 9]]
3 (c) [[0 1 2 3 4] [5 6 7 8 9] [1 1 1 1 1]]
4 (d) [[0 1 2 3 4]]

```

Câu hỏi 9: Gộp mảng theo trục axis=1:

```

1 import numpy as np
2 arr1 = np.arange(10).reshape(2, -1)
3 arr2 = np.ones((2, 5), dtype=int)
4 c = # Your code here
5 print(c)

```

Kết quả là gì?

- ```

1 (a) [[0 1 2 3 4 1 1 1 1 1] [5 6 7 8 9 1 1 1 1 2]]
2 (b) [[1 1 1 1 1 1 1 1 1] [5 6 7 8 9 5 6 7 8 9]]
3 (c) [[0 1 2 3 4 1 1 1 1 1] [5 6 7 8 9 1 1 1 1 1]]
4 (d) [[0 2 2 2 2 1 1 1 1 1] [2 2 2 2 2 1 1 1 1 1]]

```

Câu hỏi 10: Lọc giá trị theo điều kiện:

```

1 import numpy as np
2 a = np.array([2, 6, 1, 9, 10, 3, 27])
3 index = # Your code here
4 # điều kiện phần tử >= 5 và <= 10

```

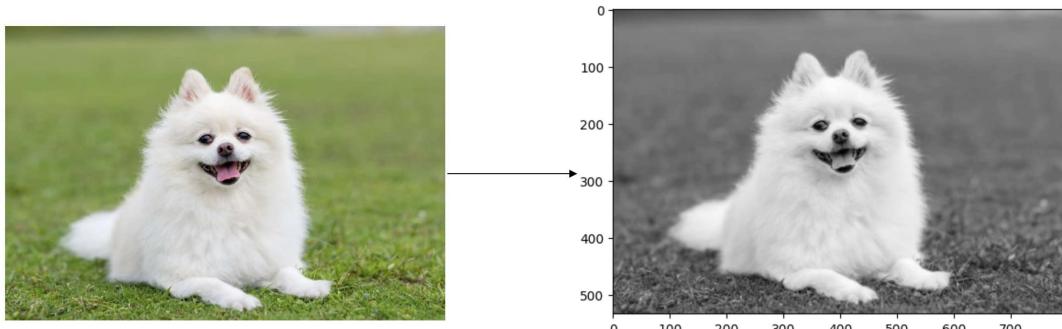
```
5 | print(a[index])|
```

Kết quả là gì?

- 1 (a) [3 9 10]
- 2 (b) [1 9 10]
- 3 (c) [6 9 10]
- 4 (d) [2 1 10]

6.3.2 Xử lý ảnh

Các câu hỏi phần này giải quyết bài toán chuyển ảnh màu về ảnh xám. Ví dụ được mô tả như hình sau: Ảnh sẽ được biểu diễn bởi tập các điểm ảnh



Hình 6.3: Chuyển ảnh màu thành ảnh xám

(Pixel) có giá trị từ 0 đến 255. Ảnh màu sẽ được biểu diễn thành ma trận có kích thước (Height, Width, Channel); trong đó số Channel là 3 tương ứng là giá trị biểu diễn cho các kênh màu R-Red, G-Green, B-Blue. Ảnh xám sẽ được biểu diễn thành ma trận có kích thước (Height, Width). Vì vậy, để chuyển ảnh màu thành ảnh xám, chúng ta sẽ tổng hợp từ 3 kênh màu RGB thành 1 giá trị duy nhất trong khoảng 0 đến 255. Có 3 phương pháp chính để tính giá trị chuyển đổi:

- Lightness: Tính giá trị trung bình của giá trị lớn nhất và nhỏ nhất cho các kênh màu: $(\max(R,G,B) + \min(R,G,B))/2$
- Average: Tính giá trị trung bình của 3 kênh màu: $(R+G+B)/3$
- Luminosity: Nhân hệ số tương ứng của 3 kênh màu như sau: $0.21*R + 0.72*G + 0.07*B$

Tải và đọc ảnh:

```

1 !gdown 1i9dqan21DjQoG5Q_VEvm0LrVwA1XD0vB
2 import matplotlib.image as mpimg
3 import numpy as np
4 img = mpimg.imread("/content/dog.jpeg")

```

Câu hỏi 11: Chuyển ảnh màu sang xám (Lightness): Hoàn thiện đoạn code sau để chuyển đổi ảnh màu thành ảnh xám theo phương pháp Lightness:

```
1 # ***** Your code here *****
2 # Sử dụng công thức Lightness: (max(R,G,B) + min(R,G,B))/2
3 gray_img_01 = # Your code here
4
5 print(gray_img_01[0, 0])
```

Kết quả là gì?

- 1 a) [102.5]
- 2 b) [92.5]
- 3 c) [82.5]
- 4 d) [72.5]

Câu hỏi 12: Chuyển ảnh màu sang xám (Average): Hoàn thiện đoạn code sau để chuyển đổi ảnh màu thành ảnh xám theo phương pháp Average:

```

1 # ***** Your code here *****
2 # Sử dụng công thức Average: (R+G+B)/3
3 gray_img_02 = # Your code here
4
5 print(gray_img_02[0, 0])

```

Kết quả là gì?

- 1 a) [107.7]
- 2 b) [97.7]
- 3 c) [87.7]
- 4 d) [77.7]

Câu hỏi 13: Chuyển ảnh màu sang xám (Luminosity): Hoàn thiện đoạn code sau để chuyển đổi ảnh màu thành ảnh xám theo phương pháp Luminosity:

```

1 # ***** Your code here *****
2 # Sử dụng công thức Luminosity: 0.21*R + 0.72*G + 0.07*B
3 gray_img_03 = # Your code here
4
5 print(gray_img_03[0, 0])

```

Kết quả là gì?

- 1 a) [96.2]
- 2 b) [116.2]
- 3 c) [126.2]
- 4 d) [136.2]

Câu hỏi 14: Background Subtraction (Tính độ chênh lệch):

```

1 import numpy as np
2 import cv2
3 bg1_image = cv2.imread("GreenBackground.png", 1)
4 bg1_image = cv2.resize(bg1_image, (678, 381))
5 ob_image = cv2.imread("Object.png", 1)
6 ob_image = cv2.resize(ob_image, (678, 381))
7 def compute_difference(bg_img, input_img):
8     return # Your code here
9 difference_single_channel = compute_difference(bg1_image, ob_image)
10 print(difference_single_channel.shape)

```

Kết quả là gì?

- 1 (a) (678, 381)
- 2 (b) (678, 381, 3)
- 3 (c) (381, 678)
- 4 (d) (381, 678, 3)

Câu hỏi 15: Background Subtraction (Tạo binary mask):

```

1 import numpy as np
2 import cv2
3 def compute_binary_mask(difference_single_channel):
4     binary_mask = (difference_single_channel > 50).astype(np.uint8)
5         * 255
6     return binary_mask
7 difference_single_channel = np.random.rand(678, 381) * 100
8 binary_mask = # Your code here
9 print(binary_mask.dtype)

```

Kết quả là gì?

- 1 (a) uint8
- 2 (b) float64
- 3 (c) int32
- 4 (d) float32

6.3.3 Phân tích dữ liệu dạng bảng

Bảng 6.1: Table 6.1: Dữ liệu Advertising dạng bảng

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12.0
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9

Câu hỏi 16: Tìm giá trị lớn nhất cột Sales:

```

1 !gdown 1YF4WkCaaGYd2Zm-PNoLa2A2tFf3e64ci
2 import pandas as pd
3 import numpy as np
4 df = pd.read_csv("/content/advertising.csv")
5 data = df.to_numpy()
6 sales = data[:, -1]
7 max_value = # Your code here
8 max_index = # Your code here
9 print(f"Max: {max_value} - Index: {max_index}")

```

Kết quả là gì?

- 1 (a) Max: 25 - Index: 30
- 2 (b) Max: 27 - Index: 30
- 3 (c) Max: 27 - Index: 175
- 4 (d) Max: 25 - Index: 175

Câu hỏi 17: Tính trung bình cột TV:

```

1 import pandas as pd
2 import numpy as np
3 df = pd.read_csv("/content/advertising.csv")
4 data = df.to_numpy()

```

```
5 tv = data[:, 0]
6 mean_tv = # Your code here
7 print(mean_tv)
```

Kết quả là gì?

- 1 (a) 146.0
- 2 (b) 147.0
- 3 (c) 148.0
- 4 (d) 149.0

Câu hỏi 18: Đếm số bản ghi Sales ≥ 20 :

```
1 import pandas as pd
2 import numpy as np
3 df = pd.read_csv("/content/advertising.csv")
4 data = df.to_numpy()
5 sales = data[:, -1]
6 count = # Your code here
7 print(count)
```

Kết quả là gì?

- 1 (a) 40
- 2 (b) 41
- 3 (c) 42
- 4 (d) 43

Câu hỏi 19: Trung bình Radio với điều kiện Sales ≥ 15 :

```
1 import pandas as pd
2 import numpy as np
3 df = pd.read_csv("/content/advertising.csv")
4 data = df.to_numpy()
5 sales = data[:, -1]
6 radio = data[:, 1]
7 mean_radio = # Your code here
8 print(mean_radio)
```

Kết quả là gì?

- 1 (a) 25.2
- 2 (b) 26.2
- 3 (c) 27.2
- 4 (d) 28.2

Câu hỏi 20: Tổng Sales với điều kiện Newspaper > trung bình:

```
1 import pandas as pd
2 import numpy as np
3 df = pd.read_csv("/content/advertising.csv")
4 data = df.to_numpy()
5 sales = data[:, -1]
6 newspaper = data[:, 2]
7 mean_newspaper = np.mean(newspaper)
8 sum_sales = # Your code here
9 print(sum_sales)
```

Kết quả là gì?

- (a) 1403.1
- (b) 1404.1
- (c) 1405.1
- (d) 1406.1

1. Hint: Các file code gợi ý có thể được tải [tại đây](#).

2. Solution: Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

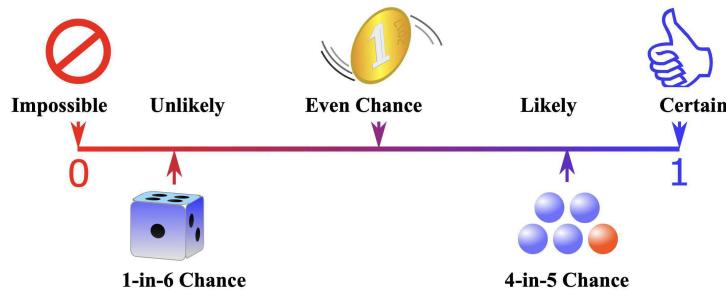
3. Rubric:

Mục	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Thực hành các thao tác với numpy + Khởi tạo array (1D, 2D tạo mảng với phần tử lặp lại, random, ...) + Tìm element hoặc giá trị theo điều kiện + Lấy hoặc replace các giá trị trong array theo điều kiện (slicing, indexing, ...) + Thay đổi shape của array (1D-2D, 2D-1D, ...) + Xếp chồng array theo các chiều + Thao tác với 2 array (tìm giá trị giống, khác, max, min, ...) + Hoán đổi hàng, cột, đảo ngược array + Thực hiện các phép toán trên array (tìm max, min, mean theo chiều nhất định, tính khoảng cách 2 array, tìm cực đại cục bộ) 	<ul style="list-style-type: none"> - Có thể hiểu và thực hiện được các thao tác chính và quan trọng trong thư viện numpy - Có nền tảng cơ bản để học và thực hiện các thao tác với mảng nhiều chiều - Nền tảng cơ bản để thực hiện các bài toán Linear Algebra, xử lý data cơ bản trong machine learning và deeplearning
2	<ul style="list-style-type: none"> - Hiểu cơ bản về các bài toán xử lý ảnh như chuyển ảnh màu thành ảnh xám - Cách biểu diễn và tính giá trị cho các kênh màu - Cách áp dụng phương pháp thay nền cho ảnh cơ bản 	<ul style="list-style-type: none"> - Áp dụng các phép tính trên ma trận của numpy để chuyển đổi ảnh màu sang ảnh xám và thay nền cho đối tượng ảnh
3	<ul style="list-style-type: none"> - Một số kỹ thuật phân tích dữ liệu dạng bảng - Cách sử dụng pandas để load file csv 	<ul style="list-style-type: none"> - Ứng dụng numpy để phân tích dữ liệu dạng bảng

Chương 7

Xác suất và bài toán phân loại dùng Bayes

7.1 Giới thiệu



Hình 7.1: Lý thuyết xác suất.

Trong lĩnh vực học máy, bài toán phân loại là một trong những vấn đề cơ bản và quan trọng nhất. Một phương pháp phổ biến để giải quyết bài toán này là sử dụng lý thuyết xác suất, cụ thể là công thức Bayes.

Giả sử X có các đặc trưng thuộc tính độc lập với nhau x_1, x_2, \dots, x_n , để phân loại X vào lớp một trong các lớp $C = c_1, c_2, \dots, c_m$, dựa vào công thức Bayes ta có:

$$P(c|X) = \frac{P(X|c).P(c)}{P(X)}$$

Bằng cách ước lượng tối đa xác suất hậu nghiệm (MAP - Maximum A Posterior) và MLE (Maximum Likelihood Estimation) ta được:

$$P(c|X) \propto P(X|c).P(c)$$

$$P(c|X) \propto P(x_1|c).P(x_2|c)\dots P(x_n|c).P(c)$$

Phương pháp này đã được chứng minh là hiệu quả, đơn giản, tính toán nhanh và được ứng dụng rộng rãi trong nhiều bài toán thực tế như lọc thư rác, phân loại văn bản, nhận dạng giọng nói, ... Trong phần này, chúng ta sẽ

sử dụng công thức trên cho bộ phân loại Naive Bayes để phân loại dữ liệu dạng bảng.

7.2 Câu hỏi tự luận

Cho trước dữ liệu thời tiết của 10 ngày (D1-D10, như bảng sau). Tập dữ liệu huấn luyện mô hình phân loại nhị phân Naive Bayes gồm các thuộc tính "Outlook", "Temperature", "Humidity", "Wind":

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Overcast	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes

Bảng 7.1: Play Tennis - Tập dữ liệu huấn luyện

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D11	Sunny	Cool	High	Strong	???

Bảng 7.2: Play Tennis - Dữ liệu testing

Hãy phát triển chương trình sử dụng mô hình phân loại Naive Bayes để dự đoán xem ngày thứ 11 (D11), AD có thể chơi tennis hay không?

- (a) "Play Tennis" = "Yes"
- (b) "Play Tennis" = "No"

Để hoàn thành bài tập này bạn cần hoàn thành các function sau đây bằng cách sử dụng thư viện numpy:

7.2.1 Hàm tạo training dataset

Hoàn thiện function `create_training_data()` để tổ chức dữ liệu bảng 1 vào array 2 chiều như bên dưới.

```

1 def create_training_data():
2     """Create the training dataset for tennis prediction."""
3     data = ### Your code here
4     return np.array(data)
5
6 train_data = create_training_data()
7 print(train_data)
8
9 *****Sample Result when we print out train_data
10 ****
11 [["Sunny" "Hot" "High" "Weak" "no"]
12 ["Sunny" "Hot" "High" "Strong" "no"]
13 ["Overcast" "Hot" "High" "Weak" "yes"]
14 ["Rain" "Mild" "High" "Weak" "yes"]
15 ["Rain" "Cool" "Normal" "Weak" "yes"]
16 ["Rain" "Cool" "Normal" "Strong" "no"]
17 ["Overcast" "Cool" "Normal" "Strong" "yes"]
18 ["Overcast" "Mild" "High" "Weak" "no"]
19 ["Sunny" "Cool" "Normal" "Weak" "yes"]
20 ["Rain" "Mild" "Normal" "Weak" "yes"]]

```

7.2.2 Hàm tính prior probabilities

Hoàn thiện function `compute_prior_probabilities` như bên dưới để tính xác suất tiên nghiệm:

```

1 def compute_prior_probabilities(train_data):
2     """
3         Calculate prior probabilities  $P(\text{Play Tennis} = \text{Yes}/\text{No})$ .
4
5     Args:
6         train_data: Training dataset
7
8     Returns:
9         Array of prior probabilities [ $P(\text{No})$ ,  $P(\text{Yes})$ ]

```

```

10 """
11     class_names = ["No", "Yes"]
12     total_samples = len(train_data)
13     prior_probs = np.zeros(len(class_names))
14
15     ### Your code here
16
17     return prior_probs
18
19 prior_probablity = compute_prior_probabilities(train_data)
20 print("P('Play Tennis' = No)", prior_probablity[0])
21 print("P('Play Tennis' = Yes)", prior_probablity[1])

```

7.2.3 Hàm tính conditional probabilities

Hoàn thiện function `compute_conditional_probabilities` như bên dưới để tính xác suất có điều kiện:

```

1 def compute_conditional_probabilities(train_data):
2     """
3         Calculate conditional probabilities  $P(\text{Feature}|\text{Class})$  for all
4             features.
5
6     Args:
7         train_data: Training dataset
8
9     Returns:
10        Tuple of (conditional_probabilities, feature_values)
11    """
12    class_names = ["No", "Yes"]
13    n_features = train_data.shape[1] - 1 # Exclude target column
14    conditional_probs = []
15    feature_values = []
16
17    for feature_idx in range(n_features):
18        # Get unique values for this feature
19        unique_values = np.unique(train_data[:, feature_idx])
20        feature_values.append(unique_values)
21
22        # Initialize conditional probability matrix

```

```

22     feature_cond_probs = np.zeros((len(class_names), len(
23                                     unique_values)))
24
25     for class_idx, class_name in enumerate(class_names):
26         # Get samples for this class
27         # Your code here
28
29         for value_idx, value in enumerate(unique_values):
30             # Count occurrences of this feature value in this
31             # class
32             # Calculate conditional probability
33             # Your code here
34
35             conditional_probs.append(feature_cond_probs)
36
37     # Test
38     _, feature_values = compute_conditional_probabilities(train_data)
39     print("x1 = ", feature_values[0])
40     print("x2 = ", feature_values[1])
41     print("x3 = ", feature_values[2])
42     print("x4 = ", feature_values[3])

```

7.2.4 Hàm lấy feature index

Hoàn thiện function `get_feature_index` để tính trả về index tương ứng với feature name:

```

1 def get_feature_index(feature_value, feature_values):
2     """
3         Get the index of a feature value in the feature values array.
4
5     Args:
6         feature_value: Value to find
7         feature_values: Array of possible feature values
8
9     Returns:
10        Index of the feature value
11        """

```

```

12     return # Your code here
13
14 _, feature_values = compute_conditional_probabilities(train_data)
15 outlook = feature_values[0]
16 i1 = get_feature_index("Overcast", outlook)
17 i2 = get_feature_index("Rain", outlook)
18 i3 = get_feature_index("Sunny", outlook)
19
20 print(i1, i2, i3)

```

7.2.5 Hàm huấn luyện mô hình Naive Bayes

Hoàn thiện function `train_naive_bayes` như bên dưới để huấn luyện bộ phân loại:

```

1 def train_naive_bayes(train_data):
2     """
3         Train the Naive Bayes classifier.
4
5     Args:
6         train_data: Training dataset
7
8     Returns:
9         Tuple of (prior_probabilities, conditional_probabilities,
10                  feature_names)
11
12     # Calculate prior probabilities
13     prior_probabilities = # Your code here
14
15     # Calculate conditional probabilities
16     conditional_probabilities, feature_names = # Your code here
17
18     return prior_probabilities, conditional_probabilities,
19                      feature_names
20
21 # Train the model
22 prior_probs, conditional_probs, feature_names = train_naive_bayes(
23                                         train_data)

```

7.2.6 Hàm dự đoán cho một test sample

Hoàn thiện function `predict_tennis` để hỗ trợ AD có nên đi chơi tennis vào ngày D11 không?

```
1 def predict_tennis(
2     X, prior_probabilities, conditional_probabilities,
3         feature_names
4 ):
5     """
6     Make a prediction for given features.
7
8     Args:
9         X: List of feature values [Outlook, Temperature, Humidity,
10            Wind]
11        prior_probabilities: Prior probabilities for each class
12        conditional_probabilities: Conditional probabilities for
13            each feature
14        feature_names: Names/values for each feature
15
16    Returns:
17        Tuple of (prediction, probabilities)
18    """
19    class_names = ["No", "Yes"]
20
21    # Get feature indices
22    feature_indices = []
23    for i, feature_value in enumerate(X):
24        feature_indices.append(get_feature_index(feature_value,
25                                              feature_names[i]))
26
27    # Calculate probabilities for each class
28    class_probabilities = []
29
30    for class_idx in range(len(class_names)):
31        # Start with prior probability
32        # Multiply by conditional probabilities
33        # Your code here
34
35        # Normalize probabilities
36    total_prob = sum(class_probabilities)
37    if total_prob > 0:
```

```
34         normalized_probs = [p / total_prob for p in
35                               class_probabilities]
36     else:
37         normalized_probs = [0.5, 0.5] # Default if all
38                               probabilities are 0
39
39     # Make prediction
40     predicted_class_idx = np.argmax(class_probabilities)
41     prediction = class_names[predicted_class_idx]
42
42     # Create probability dictionary
43     prob_dict = {
44         "No": round(normalized_probs[0].item(), 2),
45         "Yes": round(normalized_probs[1].item(), 2)
46     }
47
48     return prediction, prob_dict
49
50 X = ["Sunny", "Cool", "High", "Strong"]
51 prior_probs, conditional_probs, feature_names = train_naive_bayes(
52                                         train_data)
52 prediction, prob_dict = predict_tennis(
53     X, prior_probs, conditional_probs, feature_names
54 )
55 if prediction:
56     print("Ad should go!")
57 else:
58     print("Ad should not go!")
```

7.3 Câu hỏi trắc nghiệm

7.3.1 Binary Classification - Play Tennis

Dựa vào dữ liệu trong bảng 7.1 và sự kiện sau để trả lời các câu hỏi trắc nghiệm từ câu 1 đến câu 4:

X = (Outlook=Sunny, Temperature=Cool, Humidity=High, Wind=Strong)

1. Xác suất xảy ra sự kiện "Play Tennis"="Yes" và "Play Tennis"="No" lần lượt là?
 - a) $P(\text{"Play Tennis" = "Yes"}) = 6/10$, $P(\text{"Play Tennis" = "No"}) = 4/10$
 - b) $P(\text{"Play Tennis" = "Yes"}) = 4/10$, $P(\text{"Play Tennis" = "No"}) = 6/10$
 - c) $P(\text{"Play Tennis" = "Yes"}) = 6/10$, $P(\text{"Play Tennis" = "No"}) = 6/10$
 - d) $P(\text{"Play Tennis" = "Yes"}) = 4/10$, $P(\text{"Play Tennis" = "No"}) = 4/10$
2. Xác suất xảy ra sự kiện "Play Tennis"="Yes" khi sự kiện X xảy ra là?
 - a) $P(\text{"Play Tennis" = "Yes"}|X) \propto 0.0014$
 - b) $P(\text{"Play Tennis" = "Yes"}|X) \propto 0.0028$
 - c) $P(\text{"Play Tennis" = "Yes"}|X) \propto 0.0188$
 - d) $P(\text{"Play Tennis" = "Yes"}|X) \propto 0.0098$
3. Xác suất xảy ra sự kiện "Play Tennis"="No" khi sự kiện X xảy ra là?
 - a) $P(\text{"Play Tennis" = "No"}|X) \propto 0.0014$
 - b) $P(\text{"Play Tennis" = "No"}|X) \propto 0.0028$
 - c) $P(\text{"Play Tennis" = "No"}|X) \propto 0.0188$
 - d) $P(\text{"Play Tennis" = "No"}|X) \propto 0.0098$
4. Khi xảy ra sự kiện X, nhãn của "Play Tennis" sẽ là?
 - a) "Play Tennis" = "Yes"
 - b) "Play Tennis" = "No"

Dựa vào yêu cầu hoàn thành code trong phần 7.2 để trả lời các câu hỏi trắc nghiệm sau:

5. Kết quả nào sau đây là output từ chương trình trong phần tính xác suất tiên nghiệm 7.2.2:

- a) $P("Play Tennis" = "Yes") = 0.6, P("Play Tennis" = "No") = 0.4$
- b) $P("Play Tennis" = "Yes") = 0.3, P("Play Tennis" = "No") = 0.7$
- c) $P("Play Tennis" = "Yes") = 0.4, P("Play Tennis" = "No") = 0.8$
- d) $P("Play Tennis" = "Yes") = 0.4, P("Play Tennis" = "No") = 0.3$

6. Kết quả nào sau đây là output từ chương trình trong phần tính xác suất có điều kiện 7.2.3:

- a) $x1 = ["Cool", "Hot", "Mild"]$
 $x2 = ["Overcast", "Rain", "Sunny"]$
 $x3 = ["High", "Normal"]$
 $x4 = ["Strong", "Weak"]$
- b) $x1 = ["Overcast", "Rain", "Sunny"]$
 $x2 = ["Cool", "Hot", "Mild"]$
 $x3 = ["High", "Normal"]$
 $x4 = ["Strong", "Weak"]$
- c) $x1 = ["Strong", "Weak"]$
 $x2 = ["Cool", "Hot", "Mild"]$
 $x3 = ["High", "Normal"]$
 $x4 = ["Overcast", "Rain", "Sunny"]$
- d) $x1 = ["Overcast", "Rain", "Sunny"]$
 $x2 = ["Cool", "Hot", "Mild"]$
 $x3 = ["Strong", "Weak"]$
 $x4 = ["High", "Normal"]$

7. Kết quả nào sau đây là output từ chương trình trong phần tìm index trong 7.2.4:

- a) 1 2 0
 - b) 0 1 1
 - c) 0 1 2
 - d) 0 2 3
8. Kết quả nào sau đây là output từ chương trình của hàm dự đoán trong 7.2.6:
- a) Ad should not go!
 - b) Ad should go!

7.3.2 Multi-label Classification - Traffic Data

Cho tập dữ liệu huấn luyện mô hình phân loại Naive Bayes gồm các thuộc tính "Day", "Season", "Fog", "Rain".

Day	Season	Fog	Rain	Class
Weekday	Spring	None	None	On Time
Weekday	Winter	None	Slight	On Time
Weekday	Winter	None	None	On Time
Holiday	Winter	High	Slight	Late
Saturday	Summer	Normal	None	On Time
Weekday	Autumn	Normal	None	Very Late
Holiday	Summer	High	Slight	On Time
Sunday	Summer	Normal	None	On Time
Weekday	Winter	High	Heavy	Very Late
Weekday	Summer	None	Slight	On Time
Saturday	Spring	High	Heavy	Cancelled
Weekday	Summer	High	Slight	On Time
Weekday	Winter	Normal	None	Late
Weekday	Summer	High	None	On Time
Weekday	Winter	Normal	Heavy	Vary Late
Saturday	Autumn	High	Slight	On Time
Weekday	Autumn	None	Heavy	On Time
Holiday	Spring	Normal	Slight	On Time
Weekday	Spring	Normal	None	On Time
Weekday	Spring	Normal	Heavy	On Time

Bảng 7.3: Traffic Data - Tập dữ liệu huấn luyện

Cho sự kiện thử nghiệm:

X = (Day=Weekday, Season=Winter, Fog=High, Rain=Heavy)

9. Xác suất xảy ra sự kiện "Class" = "On Time", sự kiện "Class" = "Late", sự kiện "Class" = "Very Late" và sự kiện "Class" = "Cancelled" lần lượt là?

- (A) $P(\text{"Class" = "On Time"}) = 14/20$, $P(\text{"Class" = "Late"}) = 2/20$,
 $P(\text{"Class" = "Very Late"}) = 3/20$, $P(\text{"Class" = "Cancelled"}) = 1/20$
- (B) $P(\text{"Class" = "On Time"}) = 2/20$, $P(\text{"Class" = "Late"}) = 3/20$,
 $P(\text{"Class" = "Very Late"}) = 1/20$, $P(\text{"Class" = "Cancelled"}) = 14/20$
- (C) $P(\text{"Class" = "On Time"}) = 3/20$, $P(\text{"Class" = "Late"}) = 1/20$,
 $P(\text{"Class" = "Very Late"}) = 2/20$, $P(\text{"Class" = "Cancelled"}) = 14/20$
- (D) $P(\text{"Class" = "On Time"}) = 1/20$, $P(\text{"Class" = "Late"}) = 1/20$,
 $P(\text{"Class" = "Very Late"}) = 14/20$, $P(\text{"Class" = "Cancelled"}) = 3/20$

10. Xác suất xảy ra sự kiện "Class" = "On Time" khi sự kiện X xảy ra là:

- (A) $P(\text{"Class" = "On Time" | X}) \propto 0.0222$
- (B) $P(\text{"Class" = "On Time" | X}) \propto 0.0013$
- (C) $P(\text{"Class" = "On Time" | X}) \propto 0.0026$
- (D) $P(\text{"Class" = "On Time" | X}) \propto 0.0000$

11. Xác suất xảy ra sự kiện "Class" = "Late" khi sự kiện X xảy ra là:

- (A) $P(\text{"Class" = "Late" | X}) \propto 0.0222$
- (B) $P(\text{"Class" = "Late" | X}) \propto 0.0013$
- (C) $P(\text{"Class" = "Late" | X}) \propto 0.0026$
- (D) $P(\text{"Class" = "Late" | X}) \propto 0.0000$

12. Xác suất xảy ra sự kiện "Class" = "Very Late" khi sự kiện X xảy ra là:

- (A) $P(\text{"Class" = "Very Late" | X}) \propto 0.0222$
- (B) $P(\text{"Class" = "Very Late" | X}) \propto 0.0013$
- (C) $P(\text{"Class" = "Very Late" | X}) \propto 0.0026$
- (D) $P(\text{"Class" = "Very Late" | X}) \propto 0.0000$

13. Xác suất xảy ra sự kiện "Class" = "Cancelled" khi sự kiện X xảy ra là:

- (A) $P(\text{"Class" = "Cancelled" | X}) \propto 0.0222$

- (B) $P(\text{"Class"} = \text{"Cancelled"} \mid X) \approx 0.0013$
- (C) $P(\text{"Class"} = \text{"Cancelled"} \mid X) \approx 0.0026$
- (D) $P(\text{"Class"} = \text{"Cancelled"} \mid X) \approx 0.0000$

14. Dự đoán "Class" của sự kiện X là:

- (A) "On Time"
- (B) "Late"
- (C) "Very Late"
- (D) "Cancelled"

7.3.3 Iris Classification

Cho một tập dữ liệu huấn luyện phân loại hoa Iris dựa vào chiều dài cánh hoa như bảng dữ liệu bên dưới. Các bạn hãy trả lời các câu hỏi sau khi dùng Gaussian Naive Bayes cho data Iris này.

Length	1.4	1.0	1.3	1.9	2.0	1.8	3.0	3.8	4.1	3.9	4.2	3.4
Class	0	0	0	0	0	0	1	1	1	1	1	1

Bảng 7.4: Phân loại cánh hoa Iris dựa vào chiều dài cánh hoa - Tập dữ liệu huấn luyện

15. Giá trị mean và variance của biến đầu vào (Length) cho "Class"="0" lần lượt là?
 - a) mean = 1.566 và variance = 0.128
 - b) mean = 3.733 và variance = 0.172
 - c) mean = 1.566 và variance = 0.172
16. Giá trị mean và variance của biến đầu vào (Length) cho "Class"="1" lần lượt là:
 - a) mean = 1.566 và variance = 0.128
 - b) mean = 3.733 và variance = 0.172
 - c) mean = 1.566 và variance = 0.172
17. Cho dữ liệu kiểm thử $X = (\text{Length}=3.4)$. Xác suất dữ liệu kiểm thử X thuộc vào "Class" = "0" và "Class" = "1" lần lượt là:
 - a) $P(\text{"Class"} = "0" \mid X) \propto 1.09 \times 10^{-6}$ và $P(\text{"Class"} = "1" \mid X) \propto 0.3486$
 - b) $P(\text{"Class"} = "0" \mid X) \propto 1.09 \times 10^{-4}$ và $P(\text{"Class"} = "1" \mid X) \propto 0.3486$
 - c) $P(\text{"Class"} = "0" \mid X) \propto 1.09 \times 10^{-2}$ và $P(\text{"Class"} = "1" \mid X) \propto 0.3486$

- 1. Hint:** Các file code gợi ý có thể được tải [tại đây](#).
- 2. Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

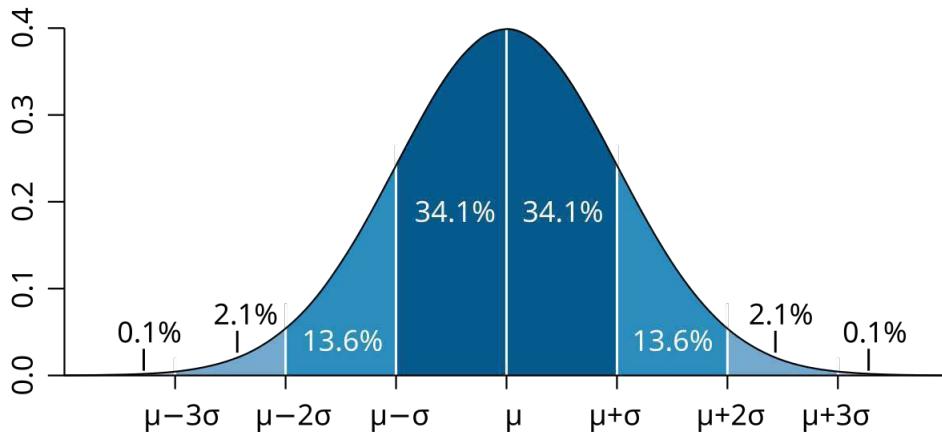
3. Rubric:

Mục	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Kiến thức cơ bản về xác suất thống kê - Sử dụng Naive Bayes cho bài toán phân loại 2 lớp 	<ul style="list-style-type: none"> - Hiểu và tính toán xác suất tiên nghiệm - Hiểu và tính toán được xác suất phân loại hai lớp dựa vào công thức Naive Bayes cho dữ liệu kiểm thử trên tập dữ liệu Play Tennis
2	<ul style="list-style-type: none"> - Kiến thức cơ bản về xác suất thống kê - Sử dụng Naive Bayes cho bài toán phân loại nhiều lớp 	<ul style="list-style-type: none"> - Hiểu và tính toán xác suất tiên nghiệm - Hiểu và tính toán được xác suất phân loại nhiều lớp dựa vào công thức Naive Bayes cho dữ liệu kiểm thử trên tập dữ liệu Traffic
3	<ul style="list-style-type: none"> - Kiến thức cơ bản về phân phối Gaussian Naive Bayes - Áp dụng vào bài toán phân loại nhị phân 	<ul style="list-style-type: none"> - Tính mean và variance trên mỗi lớp - Sử dụng phân phối Gaussian Naive Bayes dự đoán nhãn cho dữ liệu kiểm thử trên tập dữ liệu phân loại hoa dựa vào độ dài cánh hóa Iris.
4	<ul style="list-style-type: none"> - Kiến thức cơ bản về xác suất thống kê - Sử dụng Naive Bayes cho bài toán phân loại 2 lớp - Sử dụng thư viện numpy với 2D and 3D Array 	<ul style="list-style-type: none"> - Biết cách hiện thực giải thuật Naive Bayes cho bài toán phân loại 2 lớp sử dụng thư viện numpy

Chương 8

Thống kê và các phép đo similarity

8.1 Mô tả



- + **Biến ngẫu nhiên:** Phân biệt và phân tích đặc điểm của biến ngẫu nhiên rời rạc và liên tục – hai mô hình hóa cơ bản trong xử lý dữ liệu định lượng.
- + **Giá trị kỳ vọng (Mean):** Trình bày công thức và ý nghĩa thống kê của trung bình cộng trong việc đại diện cho xu hướng trung tâm của một phân phối.
- + **Phương sai và độ lệch chuẩn:** Là thước đo chính xác mức độ phân tán của dữ liệu so với trung bình, với các công thức định lượng cụ thể.
- + **Hiệp phương sai và hệ số tương quan:** Cung cấp công cụ để định lượng mối liên hệ tuyến tính giữa hai biến ngẫu nhiên, đóng vai trò quan trọng trong phân tích đa biến.
 - **Các phân phối xác suất quan trọng:**
 - Bernoulli: Mô hình hóa các hiện tượng nhị phân (có/không, đúng/sai).

- Uniform: Đại diện cho phân phối đồng đều – nơi mỗi kết quả có xác suất như nhau.
- Normal: Phân phối chuẩn – nền tảng của nhiều mô hình phân tích thống kê và học máy.

8.2 Câu hỏi trắc nghiệm

8.2.1 Basic Probability

1. Kết quả của đoạn chương trình tính mean sau đây là:

Cho Data $X = \{2, 0, 2, 2, 7, 4, -2, 5, -1, -1\}$

Hãy hoàn thiện function `compute_mean()` để tính mean μ của X đã cho.

- Data: $X = \{x_1, \dots, x_N\}$

- Mean: $\mu = \frac{1}{N} \sum_{i=1}^N x_i$

Tính giá trị trung bình

```

1 ### Question 1
2 import numpy as np
3
4 def compute_mean(X):
5     # Your code here #
6     return
7
8 X = [2, 0, 2, 2, 7, 4, -2, 5, -1, -1]
9
10 print("Mean : ", compute_mean(X))

```

Kết quả của đoạn chương trình trên là:

- (a) 1.8
- (b) 2.8

- (c) 2.8
 (d) 3.8
2. Kết quả của đoạn chương trình tính median sau đây là:
 Cho Data $X = \{1, 5, 4, 4, 9, 13\}$. Hãy hoàn thiện function **compute_median()** để tìm median m của X đã cho.

- Data: $X = \{x_1, \dots, x_N\}$
- Median:
 - Sort $X \rightarrow S$ (**tăng dần**)
 - If N is odd: $m = \frac{S_{N+1}}{2}$
 - If N is even: $m = \frac{1}{2}(S_{\frac{N}{2}} + S_{\frac{N}{2}+1})$

Tính giá trị trung vị

```

1  ### Question 2
2
3 def compute_median(X):
4     size = len(X)
5     X = np.sort(X)
6     print(X)
7     if (size % 2 == 0):
8         return # your code here #
9     else:
10        return # your code here #
11
12 X = [1, 5, 4, 4, 9, 13]
13 print("Median: ", compute_median(X))

```

Kết quả của đoạn chương trình trên là:

- (a) 4.0
 (b) 4.5
 (c) 4.6

(d) 4.7

3. Kết quả của đoạn chương trình tính variance và standard deviation sau đây là:

Cho Data $X = \{171, 176, 155, 167, 169, 182\}$

Hãy hoàn thiện function **compute_std()** để tìm standard deviation σ của X đã cho.

- Data: $X = \{x_1, \dots, x_N\}$
- Mean: $\mu = \frac{1}{N} \sum_{i=1}^N x_i$
- Variance: $var(X) = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$
- Standard Deviation: $\sigma = \sqrt{var(X)}$

Tính độ lệch chuẩn

```

1  ### Question 3
2
3 def compute_std(X):
4     mean = compute_mean(X)
5     variance = 0
6     # your code here #
7
8     return np.sqrt(variance)
9
10 X = [ 171, 176, 155, 167, 169, 182]
11 print(compute_std(X))

```

Kết quả của đoạn chương trình trên là:

- (a) 8.0
- (b) 8.23
- (c) 8.33
- (d) 8.13

4. Kết quả của đoạn chương trình tính correlation coefficient sau đây là:

Cho Data $X = \{-2, -5, -11, 6, 4, 15, 9\}$ và

$Y = \{4, 25, 121, 36, 16, 225, 81\}$

Hoàn thiện function **compute_correlation_coefficient()** để tìm correlation coefficient của X và Y đã cho?

- Random variables X, Y: $X = \{x_1, \dots, x_N\}$ $Y = \{y_1, \dots, y_N\}$
- Correlation Coefficient: $p_{xy} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$

$$= \frac{N(\sum_i x_i y_i) - (\sum_i x_i)(\sum_i y_i)}{\sqrt{N \sum_i x_i^2 - (\sum_i x_i)^2} \sqrt{N \sum_i y_i^2 - (\sum_i y_i)^2}}$$

Tính hệ số tương quan

```

1  ### Question 4
2
3 def compute_correlation_coefficient(X, Y):
4     N = len(X)
5     numerator = 0
6     denominator = 0
7     # your code here #
8
9     return np.round(numerator / denominator, 2)
10
11 X = np.asarray([-2, -5, -11, 6, 4, 15, 9])
12 Y = np.asarray([4, 25, 121, 36, 16, 225, 81])
13 print("Correlation: ", compute_correlation_coefficient(X,Y))

```

- (a) 0.41
- (b) 0.44
- (c) 0.43
- (d) 0.42

8.2.2 Tabular Data Analysis

5. Kết quả của đoạn chương trình sau đây là:

Tính hệ số tương quan

```

1 # Download dataset: !gdown 1iAOVmVfW88HyJvTBSQDI5vesf-pgKabq
2 import pandas as pd
3
4 data = pd.read_csv("advertising.csv")
5
6 def correlation(x, y):
7     # Your code here #
8
9 # Example usage:
10 x = data['TV']
11 y = data['Radio']
12 corr_xy = correlation(x, y)
13 print(f"Correlation between TV and Sales: {round(corr_xy, 2)}")
```

- (a) 0.04
- (b) 0.05
- (c) 0.06
- (d) 0.07

6. Kết quả của đoạn chương trình sau đây là:

Tính hệ số tương quan

```

1 data = pd.read_csv("advertising.csv")
2
3 def correlation(x, y):
4     # Your code here #
5
6 features = ['TV', 'Radio', 'Newspaper']
7
8 for feature_1 in features:
9     for feature_2 in features:
10         correlation_value = correlation(data[feature_1], data[
```

```
11     feature_2])  
      print(f"Correlation between {feature_1} and {feature_2}: {  
           round(correlation_value, 2)}")
```

- (a) TV and TV: -1.0
TV and Radio: 0.05
TV and Newspaper: 0.06
Radio and TV: -0.05
Radio and Radio: 1.0
Radio and Newspaper: 0.35
Newspaper and TV: -0.06
Newspaper and Radio: 0.35
Newspaper and Newspaper: 1.0
- (b) TV and TV: 1.0
TV and Radio: 0.05
TV and Newspaper: 0.06
Radio and TV: 0.05
Radio and Radio: 1.0
Radio and Newspaper: 0.35
Newspaper and TV: -0.06
Newspaper and Radio: 0.35
Newspaper and Newspaper: 1.0
- (c) TV and TV: 1.0
TV and Radio: 0.05
TV and Newspaper: 0.06
Radio and TV: 0.05
Radio and Radio: 0.0
Radio and Newspaper: 0.35
Newspaper and TV: 0.06
Newspaper and Radio: 0.35
Newspaper and Newspaper: 1.0
- (d) TV and TV: 1.0
TV and Radio: 0.05

TV and Newspaper: 0.06
Radio and TV: 0.05
Radio and Radio: 1.0
Radio and Newspaper: 0.35
Newspaper and TV: 0.06
Newspaper and Radio: 0.35
Newspaper and Newspaper: 1.0

7. Hãy cho biết đoạn code phù hợp với kết quả sau đây là:

Tính hệ số tương quan

```
1 data = pd.read_csv("advertising.csv")
2 x = data['Radio']
3 y = data['Newspaper']
4
5 result = # Your code here #
6 print(result)
7
8 # Expected output: [[1.          0.35410375]
9 #                      [0.35410375 1.         ]]
```

- (a) np.correlation(x, y)
- (b) np.coefficient(x, y)
- (c) np.corrcoef(x, y)
- (d) correlation(x,y)

8. Hãy cho biết đoạn code phù hợp với kết quả sau đây là:

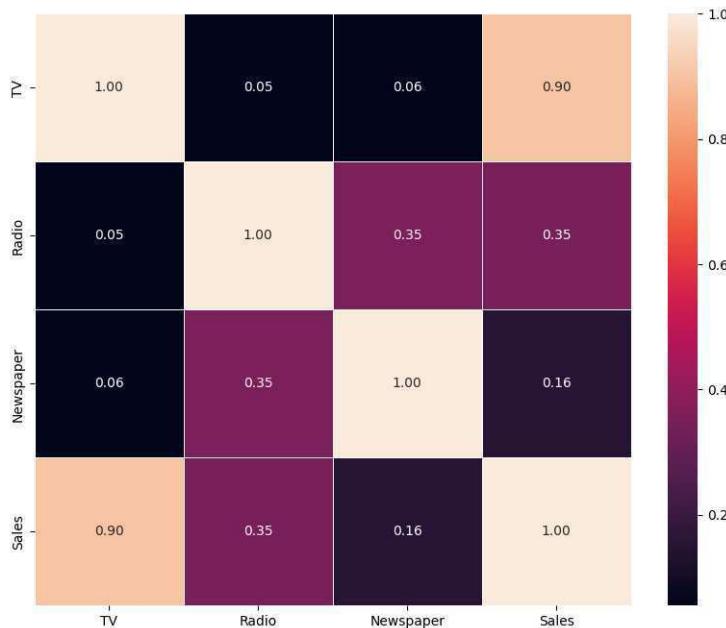
Lập bảng hệ số tương quan

```
1 data = pd.read_csv("advertising.csv")
2
3 # Your code here #
```

	TV	Radio	Newspaper	Sales
TV	1.000000	0.054809	0.056648	0.901208
Radio	0.054809	1.000000	0.354104	0.349631
Newspaper	0.056648	0.354104	1.000000	0.157960
Sales	0.901208	0.349631	0.157960	1.000000

- a) np.corr(x, y)
- b) data.corr(x, y)
- c) data.correlation(x, y)
- d) data.corr()

9. Hãy cho biết đoạn code phù hợp với kết quả sau đây là:



Trực quan hóa bảng hệ số tương quan

```

1 #Question 09
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 data = pd.read_csv("advertising.csv")
6
7 plt.figure(figsize=(10,8))
8 # Your code here #
9 plt.show()

```

- (a) sns.heatmap(data, annot=True, fmt=".2f", linewidth=.5)
- (b) data_corr = kết quả câu số 08
sns.heatmap(data_corr, annot=True, fmt=".2f", linewidth=.5)
- (c) data_corr = kết quả câu số 08
sns.heatmap(data_corr, annot=False, fmt=".2f", linewidth=.5)
- (d) data_corr = kết quả câu số 08
sns.heatmap(data_corr @ data, annot=False, fmt=".2f", linewidth=.5,)

8.2.3 Text Retrieval

10. Kết quả của đoạn chương trình đọc file và sử dụng TF-IDF để biểu diễn văn bản thành vector sau đây là:

TF-IDF Embedding

```

1 # Download dataset: !gdown 1jh2p2DlaWsDo_vEWIcTrNh3mUuXd-cw6
2 import pandas as pd
3 import numpy as np
4 from sklearn.metrics.pairwise import cosine_similarity
5 from sklearn.feature_extraction.text import TfidfVectorizer
6
7 vi_data_df = pd.read_csv("./vi_text_retrieval.csv")
8 context = vi_data_df['text']
9 context = [doc.lower() for doc in context]
10
11 tfidf_vectorizer = TfidfVectorizer()

```

```

12 context_embedded = # Your Code #
13 context_embedded.toarray()[7][0]

```

- a) 0.30
- b) 0.31
- c) 0.32
- d) 0.33

11. Kết quả của đoạn chương trình tính độ tương đồng cosine là:

Tìm kiếm với TF-IDF

```

1 def tfidf_search(question, tfidf_vectorizer, top_d=5):
2     # lowercasing before encoding
3     query_embedded = # Your Code Here *****
4     cosine_scores = # Your Code Here *****
5
6     # Get top k cosine score and index its
7     results = []
8     for idx in cosine_scores.argsort()[-top_d:][::-1]:
9         doc_score = {
10             'id': idx,
11             'cosine_score':cosine_scores[idx]
12         }
13         results.append(doc_score)
14     return results
15
16 question = vi_data_df.iloc[0]['question']
17 results = tfidf_search(question, tfidf_vectorizer, top_d=5)
18 results[0]['cosine_score']

```

- a) 0.60
- b) 0.61
- c) 0.62
- d) 0.63

12. Kết quả của đoạn chương trình tính độ tương đồng correlation là:

Tìm kiếm với Correlation

```

1 def corr_search(question, tfidf_vectorizer, top_d=5):
2     # lowercasing before encoding
3     query_embedded = # Your Code Here *****
4     corr_scores = # Your Code Here *****
5     corr_scores = corr_scores[0][1:]
6     # Get top k correlation score and index its
7     results = []
8     for idx in corr_scores.argsort()[-top_d:][::-1]:
9         doc = {
10             'id': idx,
11             'corr_score':corr_scores[idx]
12         }
13         results.append(doc)
14     return results
15
16 question = vi_data_df.iloc[0]['question']
17 results = corr_search(question, tfidf_vectorizer, top_d=5)
18 results[1]['corr_score']

```

- a) 0.20
- b) 0.21
- c) 0.22
- d) 0.23

8.2.4 Plagiarism Checker

Bài tập này giúp bạn xây dựng hệ thống kiểm tra đạo văn đơn giản bằng cách tính toán TF-IDF và Cosine Similarity chỉ với Python và NumPy, không dùng thư viện ngoài. Bạn sẽ học cách biểu diễn văn bản thành vector, đo độ tương đồng giữa các văn bản và phát hiện đạo văn thủ công. Tập dữ liệu gồm 4 câu văn Tiếng Việt có nội dung liên quan đến các khái niệm trong lĩnh vực trí tuệ nhân tạo, học máy và học sâu.

Dataset mẫu dùng để kiểm tra đạo văn

```

1 docs = [
2     "Học máy là một nhánh của trí tuệ nhân tạo",
3     "Trí tuệ nhân tạo bao gồm học máy và mạng nơ ron",
4     "Mạng nơ ron là một mô hình quan trọng trong học sâu",
5     "Học sâu là một lĩnh vực của trí tuệ nhân tạo và học máy"
6 ]

```

13. Tạo danh sách từ vựng (vocabulary) bằng cách hoàn thiện hàm build_vocabulary để thu thập tất cả từ trong văn bản, loại các từ trùng nhau và sắp xếp từ điển:

Xây dựng vocabulary

```

1 def build_vocabulary(docs):
2     vocab_set = set()
3     for doc in docs:
4         words = doc.lower().split()
5         vocab_set.update(words)
6     vocab = #Your code here#
7     return vocab

```

- a) list(vocab_set)
 - b) sorted(list(vocab_set))
 - c) vocab_set.sort()
 - d) vocab_set.append()
14. Tính TF (Term Frequency) bằng cách sử dụng thư viện Numpy để hoàn thiện đoạn code sau đây.

Xây dựng hàm tính Term Frequency

```

1 def compute_tf(doc, vocab):
2     words = doc.lower().split()
3     tf = np.zeros(len(vocab))
4     for i, term in enumerate(vocab):

```

```

5     tf[i] = #Your code here#
6     return tf

```

- a) words.count(term) / len(words)
b) len(words) / words.count(term)
c) words.index(term) / len(words)
d) 1 if term in words else 0
15. Tính IDF (Inverse Document Frequency) bằng cách hoàn thiện đoạn code sau đây có smoothing để tránh chia cho 0.

Xây dựng hàm tính Inverse Document Frequency

```

1 def compute_idf(docs, vocab):
2     N = len(docs)
3     idf = np.zeros(len(vocab))
4     for i, term in enumerate(vocab):
5         df = sum([1 for doc in docs if term in doc.lower().split()
6                   ()])
7         idf[i] = #Your code here#
8     return idf

```

- a) np.log(df / N)
b) np.log((df + 1) / (N + 1))
c) np.log((N + 1) / (df + 1)) + 1
d) np.log(N / (df + 1)) - 1
16. Tính TF-IDF vector bằng cách hoàn thiện đoạn code sau đây.

Xây dựng hàm tính TF-IDF

```

1 def compute_tfidf(tf, idf):
2     tf_idf = #Your code here#
3     return tf_idf

```

- a) $tf + idf$
 b) tf / idf
 c) $tf * idf$
 d) $idf - tf$
17. Tính độ tương đồng Cosine Similarity bằng cách hoàn thiện đoạn code sau đây.

Xây dựng hàm tính Cosine Similarity

```

1 def cosine_similarity(vec1, vec2):
2     dot_product = np.dot(vec1, vec2)
3     norm1 = np.linalg.norm(vec1)
4     norm2 = np.linalg.norm(vec2)
5     return #Your code here#

```

- a) $dot_product / (norm1 + norm2)$
 b) $dot_product / (norm1 * norm2)$
 c) $dot_product * (norm1 * norm2)$
 d) $(norm1 + norm2) / dot_product$
18. Hoàn thiện dòng vẽ heatmap để hiển thị giá trị số trong ô: Cho đoạn code dưới đây dùng để trực quan hóa ma trận độ tương đồng similarity_matrix giữa các văn bản bằng thư viện seaborn. Hãy chọn dòng sns.heatmap(...) đúng để hiển thị cả giá trị số trong từng ô, dùng định dạng 3 chữ số thập phân và bảng màu 'YlOrRd'.

Trực quan hóa kết quả kiểm tra đạo văn

```

1 import seaborn as sns
2 import matplotlib.pyplot as plt
3
4 plt.figure(figsize=(8, 6))
5 #Your code here#
6 plt.title('Ma trận độ tương đồng Cosine')

```

```
7| plt.show()
```

- a) sns.heatmap(similarity_matrix, text=True, format=' .3f ',
cmap='YlOrRd')
- b) sns.heatmap(similarity_matrix, show_text=True, fmt=' .3f '
, cmap='YlOrRd')
- c) sns.heatmap(similarity_matrix, annot=True, fmt=' .3f ',
cmap='YlOrRd')
- d) sns.heatmap(similarity_matrix, values=True, fmt=' .3f ',
cmap='YlOrRd')

8.3 Phụ lục

1. **Hint:** Các file code gợi ý có thể được tải [tại đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. Rubric:

Mục	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Kiến thức cơ bản về xác suất thống kê - Sử dụng Naive Bayes cho bài toán phân loại 2 lớp 	<ul style="list-style-type: none"> - Hiểu và tính toán xác suất tiên nghiệm - Hiểu và tính toán được xác suất phân loại hai lớp dựa vào công thức Naive Bayes cho dữ liệu kiểm thử trên tập dữ liệu Play Tennis
2	<ul style="list-style-type: none"> - Kiến thức cơ bản về xác suất thống kê - Sử dụng Naive Bayes cho bài toán phân loại nhiều lớp 	<ul style="list-style-type: none"> - Hiểu và tính toán xác suất tiên nghiệm - Hiểu và tính toán được xác suất phân loại nhiều lớp dựa vào công thức Naive Bayes cho dữ liệu kiểm thử trên tập dữ liệu Traffic
3	<ul style="list-style-type: none"> - Kiến thức cơ bản về phân phối Gaussian Naive Bayes - Áp dụng vào bài toán phân loại nhị phân 	<ul style="list-style-type: none"> - Tính mean và variance trên mỗi lớp - Sử dụng phân phối Gaussian Naive Bayes dự đoán nhãn cho dữ liệu kiểm thử trên tập dữ liệu phân loại hoa dựa vào độ dài cánh hóa Iris.
4	<ul style="list-style-type: none"> - Kiến thức cơ bản về xác suất thống kê - Sử dụng Naive Bayes cho bài toán phân loại 2 lớp - Sử dụng thư viện numpy với 2D and 3D Array 	<ul style="list-style-type: none"> - Biết cách hiện thực giải thuật Naive Bayes cho bài toán phân loại 2 lớp sử dụng thư viện numpy

Chương 9

Project 1: Vector database cho các ứng dụng AI

9.1 Giới thiệu

Trong bối cảnh công nghệ phát triển, lượng dữ liệu được tạo ra mỗi ngày đang tăng chóng mặt. Dữ liệu này không chỉ đa dạng về định dạng văn bản, hình ảnh, video, đến dữ liệu từ các thiết bị cảm biến (sensor data) và các file ghi lại hoạt động (log files) - mà còn có các yêu cầu xử lý khác nhau về tốc độ, quy mô và tính nhất quán.

Thách thức này đặt ra câu hỏi quan trọng: Làm sao để lưu trữ và truy xuất hiệu quả khối lượng dữ liệu đa dạng này? Liệu các giải pháp cơ sở dữ liệu truyền thống có đủ khả năng đáp ứng nhu cầu của thời đại hiện đại?

Câu trả lời đặc biệt trở nên phức tạp khi chúng ta bước vào kỷ nguyên AI. Với sự bùng nổ của machine learning và deep learning, một loại dữ liệu hoàn toàn mới đang trở nên cực kỳ quan trọng: Vector embeddings - những biểu diễn số học của thông tin phức tạp như văn bản, hình ảnh, âm thanh, và video. Lúc này các giải pháp truyền thống như RDBMS (Relational Database Management System) bộc lộ hạn chế khi xử lý dữ liệu đa phương tiện, trong khi NoSQL gặp khó khăn với các truy vấn phức tạp.

Đây là lúc Vector Database trở thành giải pháp chuyên biệt, giúp xử lý hiệu quả các tác vụ tìm kiếm tương tự (similarity search), matching và recommendation mà các hệ quản trị dữ liệu truyền thống như SQL hay NoSQL không được thiết kế để tối ưu.

Trong nội dung tiếp theo, chúng ta sẽ khám phá chi tiết về Vector Database, cách chúng hoạt động, các công cụ phổ biến hiện nay và ứng dụng thực tiễn qua hai project cụ thể:

- Nhận diện khuôn mặt để điểm danh nhân viên.
- Hệ thống gợi ý cocktail dựa trên khẩu vị người dùng.

Mục lục

AI VIET NAM (AIO2025)

aivietnam.edu.vn

9.2 Các loại cơ sở dữ liệu truyền thống

9.2.1 SQL Database

Cơ sở dữ liệu quan hệ (Relational Database), thường được gọi là SQL Database, là một trong những nền tảng lâu đời và ổn định nhất trong lĩnh vực lưu trữ và truy vấn dữ liệu. SQL, viết tắt của Structured Query Language, là ngôn ngữ tiêu chuẩn được sử dụng để tương tác với các hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) như MySQL, PostgreSQL, Oracle và SQL Server.

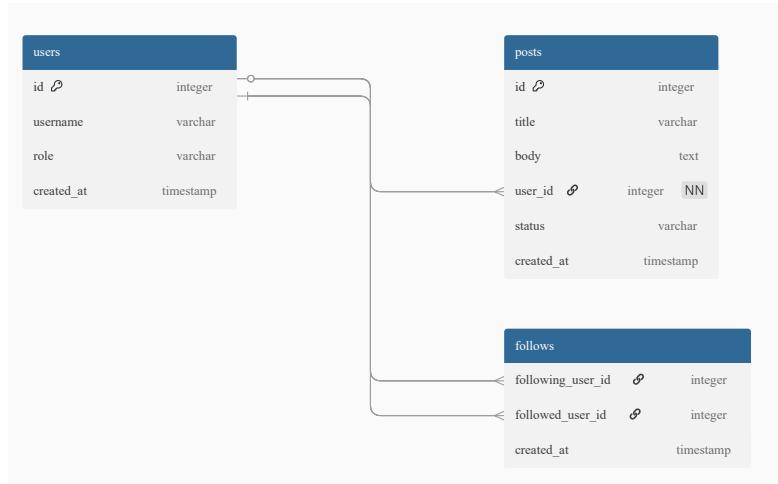
Điểm đặc trưng của SQL Database là:

- Dữ liệu được tổ chức theo bảng (table), trong đó mỗi bảng bao gồm các cột (column – thể hiện các thuộc tính dữ liệu) và hàng (row – mỗi hàng là một bản ghi dữ liệu).
- Mỗi bảng thường có khóa chính (primary key) để đảm bảo tính duy nhất, và có thể liên kết với các bảng khác thông qua khóa ngoại (foreign key), hình thành nên mối quan hệ giữa các thực thể trong hệ thống.
- Kiểu tổ chức này đặc biệt phù hợp với các hệ thống có cấu trúc dữ liệu rõ ràng và ổn định, chẳng hạn như hệ thống tài chính, quản lý nhân sự, hay quản lý đơn hàng trong thương mại điện tử.
- Một số hệ quản trị SQL phổ biến: MySQL, PostgreSQL, SQL Server, Oracle Database

Hình 1: Ví dụ về SQL Database

Ví dụ về cơ sở dữ liệu mạng xã hội đơn giản:

- Có bảng User chứa thông tin về tên người dùng, vai trò và thời gian tạo.
- Có bảng Posts đăng chứa tiêu đề, nội dung, ID của người đăng, trạng thái và thời gian đăng.



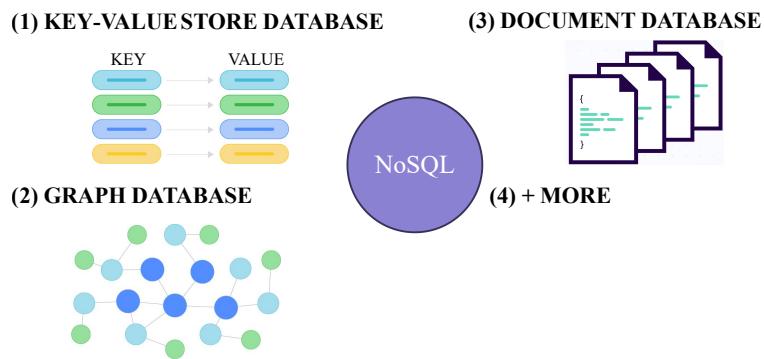
- Có bảng follows dõi chứa danh sách những ai đang theo dõi ai, cùng thời gian bắt đầu theo dõi.

Tuy nhiên, chính sự "nghiêm ngặt" trong thiết kế của SQL đôi khi lại trở thành điểm yếu khi xử lý dữ liệu phi cấu trúc hoặc cần mở rộng quy mô linh hoạt. Việc thay đổi cấu trúc bảng khi dữ liệu thay đổi có thể gây khó khăn, và khả năng scale-out (mở rộng theo chiều ngang) bị hạn chế do tính nhất quán dữ liệu chặt chẽ.

9.2.2 NoSQL Database

Khi dữ liệu ngày càng trở nên đa dạng về định dạng và tăng nhanh về khối lượng, các hệ quản trị cơ sở dữ liệu truyền thống như SQL bắt đầu bộc lộ những hạn chế rõ rệt. Các hệ SQL hoạt động rất tốt với dữ liệu có cấu trúc rõ ràng, như bảng thông tin khách hàng hay đơn hàng, nhưng lại gặp khó khăn khi xử lý dữ liệu phi cấu trúc như văn bản tự do, log hệ thống, hình ảnh, video hoặc dữ liệu cảm biến – vốn không phù hợp với mô hình bảng cố định.

NoSQL ra đời để giải quyết các vấn đề của SQL. NoSQL không ràng buộc theo mô hình bảng-trường-hàng mà thay vào đó cho phép lưu trữ dữ liệu ở nhiều dạng: document (tài liệu), key-value, graph (đồ thị). Điều này giúp hệ thống linh hoạt với cấu trúc dữ liệu thay đổi nhanh, đồng thời dễ dàng mở rộng theo chiều ngang để đáp ứng khối lượng truy cập và lưu trữ lớn.



Hình 2: Các loại cấu trúc dữ liệu trong NoSQL

Đặc trưng dữ liệu:

- Không yêu cầu thiết kế dữ liệu cố định theo lược đồ sẵn (schema). Điều này đồng nghĩa với việc người phát triển có thể thêm các trường dữ liệu mới vào một bản ghi mà không cần sửa đổi toàn bộ cấu trúc bảng như trong SQL. Sự linh hoạt này đặc biệt hữu ích với các ứng dụng đang trong quá trình phát triển nhanh, hoặc có dữ liệu đầu vào thay đổi liên tục như social media, IoT hoặc logs hệ thống.
- NoSQL nổi bật so với SQL là khả năng mở rộng theo chiều ngang (horizontal scaling). Khác với mở rộng theo chiều dọc (vertical scaling) – tức là nâng cấp phần cứng của một máy chủ duy nhất (thêm RAM, CPU, ổ cứng mạnh hơn) – thì mở rộng theo chiều ngang là hệ thống có thể thêm nhiều máy chủ mới (node) để chia sẻ gánh nặng xử lý và lưu trữ dữ liệu.
- Các hệ quản trị cơ sở dữ liệu NoSQL như MongoDB, Cassandra hay Couchbase được thiết kế với kiến trúc phân tán, cho phép tự động phân mảnh dữ liệu (sharding) và rebalancing giữa các node trong hệ thống.

Mặc dù NoSQL đã giải quyết tốt bài toán mở rộng và lưu trữ dữ liệu phi cấu trúc, nhưng khi AI ra đời nhu cầu truy vấn mới phát sinh. Lúc này việc tìm kiếm theo ngữ nghĩa – chẳng hạn như tìm văn bản, hình ảnh hoặc sản phẩm tương tự ngày càng cần thiết để phục vụ cho các mô hình DeepLearning thì các hệ thống truyền thống tỏ ra hạn chế. Đây chính là lúc Vector Database ra đời, với khả năng lưu trữ và truy vấn theo độ tương đồng giữa các vector

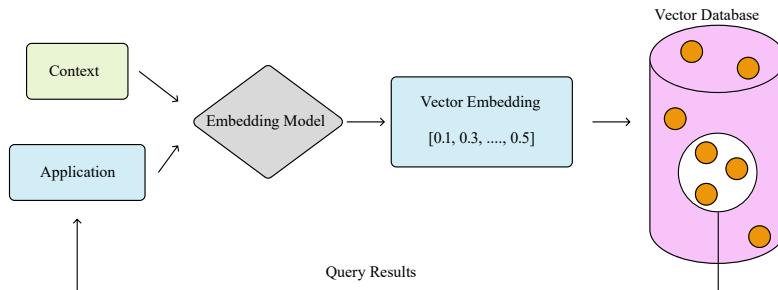
nhúng (vector embedding), mở ra hướng đi mới cho các ứng dụng AI và tìm kiếm thông minh.

9.3 Vector Database

9.3.1 Khái niệm

Vector Database (hay còn gọi là vector store, vector search engine) là một loại cơ sở dữ liệu được thiết kế chuyên biệt để lưu trữ, chỉ mục và truy vấn các vector embeddings — biểu diễn số của dữ liệu phức tạp như hình ảnh, văn bản, âm thanh, sensor data... .

Trong khi các cơ sở dữ liệu truyền thống quản lý dữ liệu dưới dạng scalar (số, chuỗi, ngày tháng...), thì Vector Database xử lý tốt dữ liệu đa chiều (hàng trăm đến hàng nghìn chiều) do các embedding tạo ra — chứa các đặc trưng ngữ nghĩa để phục vụ cho các ứng dụng AI như similarity search, semantic retrieval.



Hình 3: Tổng quan về vector database

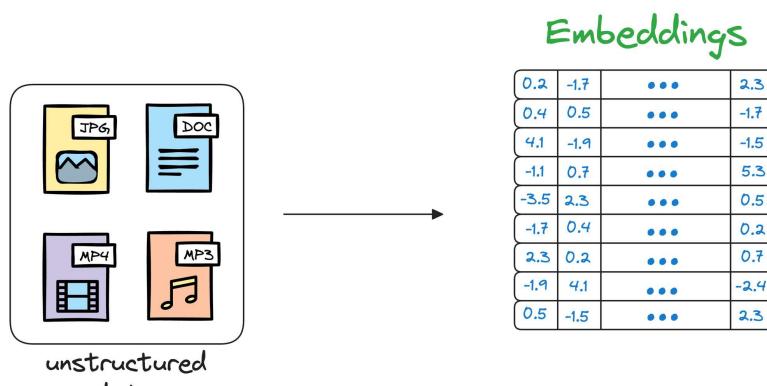
Hãy cùng phân tích hình trên đây:

- Đầu tiên, chúng ta sử dụng mô hình nhúng (Embedding Model) để tạo các vector nhúng cho nội dung chúng ta muốn lập chỉ mục.
- Vector embedding được chèn vào Vector Database , với một số tham chiếu đến nội dung gốc của vector đó
- Khi ứng dụng đưa ra một truy vấn, chúng ta sử dụng cùng một Embedding Model để tạo các Vector Embedding cho truy vấn và sử dụng các Vector embedding đó để truy vấn trong Vector Database tìm các vector embedding tương đồng. Sau đó từ các vector embedding tương đồng tham chiếu lại nội dung gốc như đề cập ở trên

9.3.2 Embedding

Embedding là quá trình chuyển đổi dữ liệu thô (văn bản, hình ảnh, âm thanh, video...) thành một vector số có nhiều chiều (multi-dimensional vector), trong đó mỗi chiều đại diện cho một đặc trưng (feature) ngữ nghĩa hoặc hình thái của dữ liệu.

Việc biểu diễn như vậy giúp các hệ thống AI và Vector Database có thể tính toán độ tương đồng giữa các đối tượng mà không cần hiểu trực tiếp ngôn ngữ tự nhiên, hình ảnh thô hay âm thanh.



Hình 4: Minh họa cho vector embedding

Vậy vì sao cần embedding ?

- Dữ liệu như văn bản, hình ảnh vốn ở dạng rời rạc, không thể so sánh trực tiếp.
- Embedding giúp đưa dữ liệu về cùng một không gian vector, nơi mọi dữ liệu đều trở thành một điểm trong không gian đó.
- Nhờ đó, việc tìm kiếm tương tự, phân cụm hay recommendation đều chỉ cần đo khoảng cách giữa các vector.

9.3.3 Indexing trong Vector Database

Sau khi dữ liệu đã được chuyển đổi thành vector embedding, bước tiếp theo trong Vector Database là indexing — xây dựng chỉ mục giúp tối ưu hoá khả năng truy vấn tìm kiếm gần đúng (Approximate Nearest Neighbor - ANN) với tốc độ nhanh và chi phí hợp lý ngay cả trên dữ liệu lớn.

Vậy tại sao chúng ta cần indexing ?

- Với hàng triệu đến hàng tỷ vector, việc so sánh trực tiếp (brute-force search) là phi thực tế do chi phí tính toán lớn.
- Index giúp tổ chức các vector thông minh để chỉ cần tìm kiếm trong một phần nhỏ không gian vector nhưng vẫn đảm bảo độ chính xác cao.

Trong phạm vi bài chúng ta sẽ tìm hiểu ba cách indexing dữ liệu từ cơ bản đến nâng cao.

Flat Index

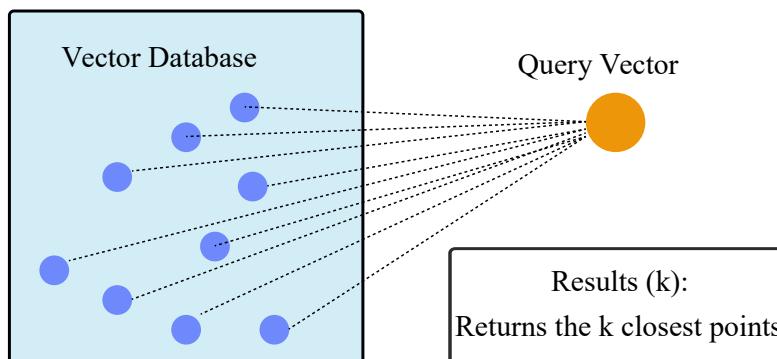
Flat index là một tên gọi khác của tìm kiếm brute-force. Tất cả các vector được lưu trữ trong một cấu trúc chỉ mục duy nhất mà không có bất kỳ tổ chức phân cấp nào.

Có phần giống với KNN Flat Index là cách tiếp cận đơn giản nhất: Vector Database sẽ so sánh vector truy vấn với tất cả các vector đã lưu trữ bằng một độ đo khoảng cách (L2, Cosine, Dot Product) sau đó trả về kết quả gần nhất.

Đặc điểm của phương pháp này là:

- Chính xác tuyệt đối: vì so sánh toàn bộ vector dataset.
- Không cần build index phức tạp.
- Chi phí tính toán cao nếu dataset lớn (hàng triệu vector trở lên).

Chúng ta sẽ dùng phương pháp này khi dataset (vài nghìn đến vài trăm nghìn vector) hoặc khi cần đảm bảo độ chính xác tuyệt đối và không giới hạn về tài nguyên.



Hình 5: Minh họa cho flat index

Inverted File Index - IVF

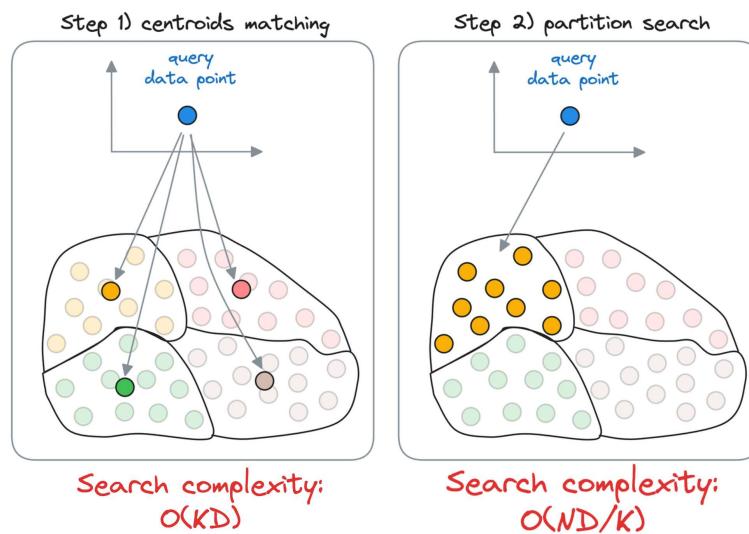
IVF là một trong những kỹ thuật lập chỉ mục đơn giản và trực quan nhất. Mặc dù thường được sử dụng trong các hệ thống truy xuất văn bản, nó có thể được điều chỉnh cho cơ sở dữ liệu vector để tìm kiếm xấp xỉ lân cận gần nhất.

Vậy IVF hoạt động thế nào:

- Sử dụng các thuật toán phân cụm để chia tất cả các vector trong dataset thành các vùng khác nhau (cluster).

- Kết quả là, mỗi cluster có một trọng tâm (centroid) tương ứng, và mỗi vectơ chỉ được liên kết với một cluster tương ứng với centroid gần nhất của nó. Do đó, mỗi centroid duy trì thông tin về tất cả các vectơ thuộc về phân vùng của nó.
- Khi truy vấn thay vì tìm kiếm vector gần nhất chúng ta chỉ cần centroid gần nhất với vector truy vấn. Và lúc này chúng ta chỉ cần tìm các vector gần nhất thuộc centroid của cluster đó

Giả sử chúng ta có N vectơ và mỗi vectơ có D chiều, độ phức tạp sẽ là $O(ND)$ để tìm vectơ gần nhất đối với Flat index. So với IVF chúng ta có có k trọng tâm, tổng cộng N vectơ, mỗi vectơ có D chiều và các vectơ được phân bổ đều trên tất cả các phân vùng thì độ phức tạp sẽ là $O(kD + NDk)$

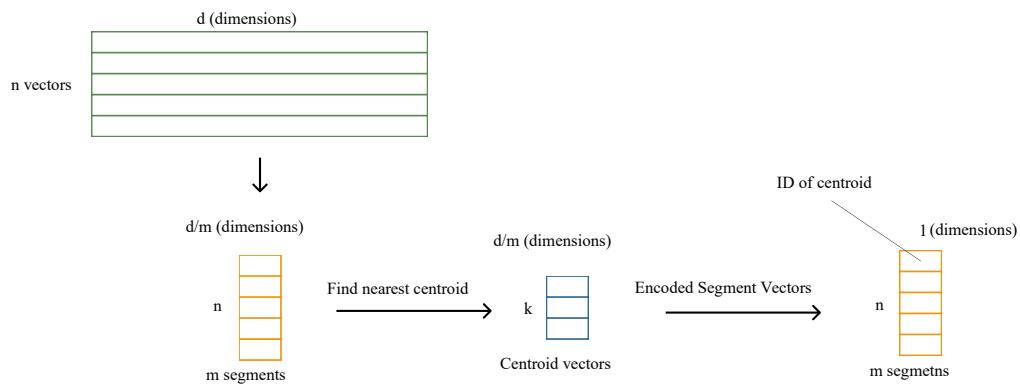


Hình 6: Minh họa cho Inverted File Index

Product Quantization (PQ)

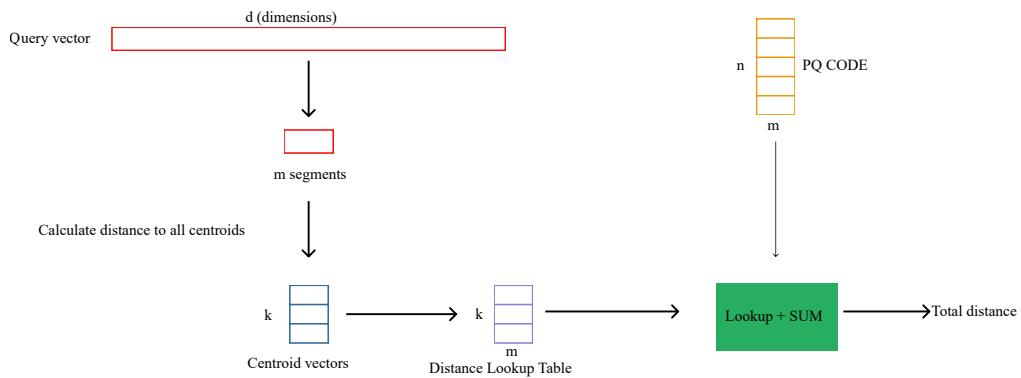
Ý tưởng về lượng tử hóa nói chung đề cập đến việc nén dữ liệu trong khi vẫn bảo toàn thông tin gốc. Chúng ta sẽ đi phần đầu là cách PQ tổ chức trong database trước

- Vector gốc có d chiều được chia thành m segment.
- Với mỗi đoạn, thuật toán K-means được áp dụng để lượng hóa và tạo ra một codebook — một tập hợp các centroid đại diện cho các giá trị có thể của đoạn đó. Thay vì lưu giá trị thực, ta lưu **chỉ số** của centroid tương ứng.
- Kết quả: vector gốc d chiều có thể chỉ còn là một chuỗi các chỉ số codebook, giúp giảm kích thước từ vài KB xuống chỉ còn vài bytes.



Hình 7: Minh họa cho database khi
dùng Product Quantization (PQ)

Với vector truy vấn Q , ta chia Q bằng cách tương tự, rồi tính khoảng cách từ từng segment của Q tới các centroid tương ứng, tạo nên một ma trận khoảng cách. Sau đó, với mỗi PQ code(vector trong database đã được mã hóa), ta tra lookup distance từ ma trận này (theo từng segment đã mã hóa) và cộng lại để ước tính khoảng cách giữa Q và các vector trong DB. Vector có tổng distance nhỏ nhất được xem là gần nhất.



Hình 8: Minh họa khi query trong database dùng PQ

So sánh một số công cụ Vector Database phổ biến

Vector Database Tools là các công cụ chuyên dụng được thiết kế để lưu trữ, indexing và tìm kiếm dữ liệu dưới dạng vector embeddings - những biểu diễn số học đa chiều của dữ liệu phi cấu trúc như text, hình ảnh, âm thanh hoặc video.

Bảng 9.0: So sánh các công cụ Vector Database phổ biến.

Công cụ	Đặc điểm chính	Ưu điểm	Hạn chế
Faiss	Library C++/Python, không có persistence.	Hiệu năng rất cao, nhiều thuật toán ANN.	Không lưu trữ dữ liệu lâu dài.
Milvus	Distributed, persistent, REST API.	Production-ready, hỗ trợ nhiều metric.	Cần vận hành hạ tầng riêng.
Weaviate	Semantic search, GraphQL API.	Tích hợp mô hình ngôn ngữ, dễ dùng.	Hạn chế backend tùy biến.
Pinecone	SaaS, dễ deploy, auto-scale.	Không cần quản lý hạ tầng.	Phụ thuộc vào nhà cung cấp.
pgvector	Extension cho PostgreSQL.	Dễ tích hợp với RDBMS.	Hiệu năng thấp hơn chuyên dụng.

9.4 Distance Metrics and Loss Functions

Trong Vector Database (VD) như FAISS, Pinecone, hay Weaviate, mỗi đối tượng (ảnh, văn bản, sản phẩm, người dùng...) được ánh xạ thành một **vector** trong không gian nhiều chiều.

Khi cần tìm kiếm đối tượng tương tự, hệ thống so sánh độ giống nhau hoặc độ khác biệt giữa các vector. Các hàm dùng để đo khoảng cách hoặc độ tương đồng này gọi là **Distance Loss Functions**.

9.4.1 L2 Distance (Euclidean Distance)

Công thức:

$$L2(A, B) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$$

Trong đó:

- $A = [a_1, a_2, \dots, a_n]$ là vector thứ nhất.
- $B = [b_1, b_2, \dots, b_n]$ là vector thứ hai.
- n là số chiều (số phần tử của vector).

Ví dụ: $A = [1, 2], B = [4, 6]$

$$L2 = \sqrt{(1-4)^2 + (2-6)^2} = \sqrt{9+16} = 5$$

Ứng dụng:

- Phổ biến trong tìm kiếm hình ảnh, nhận diện khuôn mặt.
- Phù hợp khi embedding chưa được chuẩn hóa.

9.4.2 L1 Distance (Manhattan Distance)

Công thức:

$$L1(A, B) = \sum_{i=1}^n |a_i - b_i|$$

Trong đó:

- $|a_i - b_i|$ là khoảng cách tuyệt đối giữa hai thành phần tương ứng.
- n là số chiều của vector.

Ví dụ: $A = [1, 2], B = [4, 6]$

$$L1 = |1-4| + |2-6| = 3 + 4 = 7$$

Ứng dụng:

- Dùng trong môi trường dữ liệu có nhiều nhiễu hoặc giá trị ngoại lai.
- Phù hợp với dữ liệu rời rạc (discrete) hoặc không liên tục.

9.4.3 Cosine Similarity / Cosine Distance

Công thức Cosine Similarity:

$$\cos(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

Trong đó:

- $A \cdot B = \sum_{i=1}^n a_i b_i$ là tích vô hướng giữa hai vector.
- $\|A\| = \sqrt{\sum_{i=1}^n a_i^2}$ là độ dài (chuẩn Euclidean) của vector A .
- $\|B\| = \sqrt{\sum_{i=1}^n b_i^2}$ là độ dài của vector B .

Cosine Distance:

$$\text{Distance}_{\cos} = 1 - \cos(A, B)$$

Ví dụ:

- $A = [1, 0], B = [0, 1] \Rightarrow \cos = 0$ (góc 90°).
- $A = [1, 1], B = [2, 2] \Rightarrow \cos = 1$ (cùng hướng).

Ứng dụng:

- Thường dùng trong NLP, so khớp câu hỏi, tìm kiếm văn bản.
- Không bị ảnh hưởng bởi độ dài vector (chỉ quan tâm đến góc).

9.4.4 Dot Product (Inner Product)

Công thức:

$$A \cdot B = \sum_{i=1}^n a_i b_i$$

Trong đó:

- a_i, b_i là thành phần thứ i của vector A và B .
- Kết quả là một số vô hướng, phản ánh mức độ "đồng hướng và độ lớn".

Ví dụ: $A = [1, 2]$, $B = [3, 4]$

$$A \cdot B = 1 \times 3 + 2 \times 4 = 3 + 8 = 11$$

Ứng dụng:

- Dùng trong các hệ thống tìm kiếm với vector lớn và hiệu suất cao.
- Phổ biến trong CLIP, BERT, hoặc khi dùng Approximate Nearest Neighbor (ANN).

9.5 Face-Based Employee Check-in System

Mục tiêu dự án

Project: Hệ thống điểm danh khuôn mặt nhân viên theo thời gian thực.

Project Objectives:

- Ứng dụng sử dụng FAISS để xây dựng hệ thống tìm kiếm ảnh tương đồng. Thử nghiệm **2-level embedding** nhằm nhấn mạnh tầm quan trọng của việc trích xuất đặc trưng dữ liệu
- **Level 1: Lưu ảnh thành vector raw pixel**
 - Mỗi ảnh được chuyển thành vector raw pixel bằng Numpy flatten.
- **Level 2 Lưu ảnh feature vector**
 - Sử dụng mô hình học sâu như InceptionResnetV1.
 - Mô hình hoạt động như một hộp đen, chỉ dùng để trích xuất feature map mà không huấn luyện lại.
- **Build streamlit app**
 - Hệ thống nhận ảnh từ camera, trích xuất vector đặc trưng.
 - Sử dụng FAISS để truy vấn ảnh giống nhất trong dữ liệu.
 - Nếu độ tương đồng vượt ngưỡng, hệ thống hiển thị tên nhân viên.

Hướng dẫn từng bước

Bước 1: Chuẩn bị tập dữ liệu

- Tải Dataset
- Xử lý dữ liệu

Bước 2: Embedding ảnh thành numpy

- Embedding ảnh thành numpy matrix.
- Dùng FAISS để tìm vector gần nhất trong index.
- Test với ảnh trong/ngoài dataset để đánh giá hiệu suất mô hình.

Bước 3: Embedding ảnh thành features map

- Embedding ảnh dùng InceptionResnetV1
- Dùng FAISS để tìm vector gần nhất trong index.
- Test với ảnh trong/ngoài dataset để đánh giá hiệu suất mô hình.

Bước 4: Nhận diện và hiển thị kết quả

- So sánh khoảng cách L2 giữa vector truy vấn và vector gần nhất.
- Nếu khoảng cách nhỏ hơn ngưỡng: hiển thị tên nhân viên.
- Ngược lại: thông báo “Không nhận diện được”.

Kết luận: Hệ thống nhận diện khuôn mặt dựa trên tìm kiếm vector trong FAISS index, độ chính xác phụ thuộc vào độ phân giải ảnh, ánh sáng và chất lượng dữ liệu đầu vào.

9.5.1 Chuẩn bị tập dữ liệu

Tải dataset theo đường link. Trích xuất nhãn labels và đường dẫn chứa ảnh image_path.

Tải dữ liệu

```
1 !gdown 1w7Riq_itzcijqkvSW0yyivvlw0IgRGa_
2 !unzip Dataset.zip
```

Sau bước unzip, chúng ta có thể thêm ảnh vào folder Dataset với tên file đúng như format chung "*Avatar_EmployeeName.jpg*"

Tạo labels và image_path

```
1 dataset_path = 'Dataset'
2 os.listdir(dataset_path)
3 image_paths = []
4 labels = []
5 for filename in os.listdir(dataset_path):
6     if filename.endswith(('.jpg', '.JPG', '.png', '.jpeg')):
7         image_paths.append(os.path.join(dataset_path, filename))
8         file_name = filename.split('.')[0]
9         label = file_name[7:]
10        labels.append(label)
```

9.5.2 Vectorize Image thành raw pixel

Chuẩn hóa hình ảnh về kích thước 300x300 pixels và chuyển đổi thành vector 270,000 chiều. Xử lý ảnh xám thành RGB, chuẩn hóa giá trị pixel về khoảng [0,1] và flatten thành vector 1 chiều. Sau đó thêm các vector vào FAISS index và lưu labels vào label_map.

Chuyển đổi tất cả hình ảnh thành vector bằng numpy và lưu trữ trong FAISS index để có thể tìm kiếm tương tự nhanh chóng.

Cấu hình tham số

```
1 IMAGE_SIZE = 300
2 VECTOR_DIM = 300 * 300 * 3 # For RGB images (300x300x3)
```

Khởi tạo FAISS index

```
1 index = faiss.IndexFlatL2(VECTOR_DIM)
2 label_map = []
```

Chuyển đổi hình ảnh thành vector

```
1 def image_to_vector(image_path):
2     img = Image.open(image_path).resize((IMAGE_SIZE, IMAGE_SIZE))
3     img_array = np.array(img)
4
5     # Handle grayscale images (convert to RGB)
6     if len(img_array.shape) == 2:
7         img_array = np.stack((img_array,), axis=-1)
8
9     # Normalize pixel values to [0, 1]
10    vector = img_array.astype('float32') / 255.0
11    return vector.flatten()
```

Xây dựng index từ dataframe

```
1 for idx, row in df.iterrows():
2     image_path = row['image_path']
3     label = row['label']
4
5     try:
6         vector = image_to_vector(image_path)
7         index.add(np.array([vector]))
8         label_map.append(label)
9     except Exception as e:
10        print(f"Error processing {image_path}: {e}")
```

Lưu kết quả

```
1 faiss.write_index(index, "employee_images.index")
2 np.save("label_map.npy", np.array(label_map))
```

9.5.3 Xây dựng hàm tìm kiếm và hiển thị kết quả

Tìm kiếm các hình ảnh tương tự nhất với ảnh đầu vào. Đầu tiên chuyển đổi ảnh truy vấn thành vector nó dùng FAISS index và label mapping để tìm kiếm k ảnh tương đồng nhất và hiển thị.

Hàm tìm kiếm ảnh tương đồng

```

1 def search_similar_images(query_image_path, k=5):
2     """Search for similar employee images"""
3     # Load index and labels
4     index = faiss.read_index("employee_images.index")
5     label_map = np.load("label_map.npy")
6
7     # Convert query image to vector
8     query_vector = image_to_vector(query_image_path)
9
10    # Search in Faiss
11    distances, indices = index.search(np.array([query_vector]), k)
12
13    # Get results
14    results = []
15    for i in range(len(indices[0])):
16        employee_name = label_map[indices[0][i]]
17        distance = distances[0][i]
18        results.append((employee_name, distance))
19
20    return results

```

Hàm hiển thị top 5 ảnh tương đồng

```

1 def display_query_and_top_matches(query_image_path):
2     query_img = Image.open(query_image_path)
3     query_img = query_img.resize((300, 300))
4
5     plt.figure(figsize=(5, 5))
6     plt.imshow(query_img)
7     plt.title("Query Image")
8     plt.axis('off')
9     plt.show()
10

```

```
11     matches = search_similar_images(query_image_path)
12
13     """Display the top 5 matching employee images with distances"""
14     # Get the image paths for the results
15     top_matches = []
16     for name, distance in matches:
17         # Find the image path for this employee in df
18         img_path = df[df['label'] == name]['image_path'].values[0]
19         top_matches.append((name, distance, img_path))
20     top_matches
21
22     # Create plot
23     plt.figure(figsize=(15, 5))
24     for i, (name, distance, img_path) in enumerate(top_matches):
25         img = Image.open(img_path)
26         img = img.resize((300, 300))
27
28         plt.subplot(1, 5, i+1)
29         plt.imshow(img)
30         plt.title(f"{name}\nDist: {distance:.2f}")
31         plt.axis('off')
32
33     plt.tight_layout()
34     plt.show()
```

Test độ hiệu quả khi chuyển ảnh thành vector bằng Numpy flatten

Khi thử nghiệm với các hình ảnh đã có trong tập dataset, phương pháp này hoạt động tốt và tìm đúng ảnh tương ứng. Tuy nhiên, khi thử với các ảnh mới chưa từng xuất hiện trong dataset, độ chính xác giảm rõ rệt, kết quả tìm được thường không đúng. Điều này cho thấy việc sử dụng Numpy flatten để tạo vector đặc trưng cho ảnh là chưa hiệu quả, vì cách này chỉ đơn giản trác phẳng các giá trị pixel mà không học được các đặc trưng sâu về khuôn mặt hay nội dung hình ảnh. Do đó, ảnh sau khi flatten dễ bị mất thông tin về cấu trúc và ngữ cảnh, dẫn đến chất lượng tìm kiếm kém khi gặp ảnh mới lạ.



Hình 9: Đánh giá với hình ảnh có trong dataset khi chuyển ảnh thành vector raw pixels



Hình 10: Đánh giá với hình ảnh không có trong dataset khi chuyển ảnh thành vector raw pixels

9.5.4 Vectorize image thành feature maps

Thay vì raw pixel chúng ta sẽ vectorize image thành feature map bằng cách sử dụng mô hình InceptionResnetV1 đã được pretrain trên dataset VGGFace. Lúc này thay vì chuyển đổi hình ảnh thành vector 270,000 chiều từ pixels, sử dụng deep learning để trích xuất features 512 chiều có ý nghĩa cao hơn. Vẫn dùng FAISS để indexing

Khởi tạo mô hình pretrained

```
1 from facenet_pytorch import InceptionResnetV1
2 face_recognition_model = InceptionResnetV1(pretrained='vggface2').
3                                         eval()
```

Resize ảnh

```
1 transform = transforms.Compose([
2     transforms.Resize((300, 300)),
3     transforms.ToTensor(),
4     transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
5 ])
```

Hàm trích xuất đặc trưng

```
1 def extract_feature(image_path, model):
2     img = Image.open(image_path).convert('RGB')
3     img_tensor = transform(img).unsqueeze(0)
4     with torch.no_grad():
5         features = model(img_tensor)
6     return features.squeeze().numpy()
```

Xây dựng FAISS index với features

```
1 VECTOR_DIM = 512
2 index = faiss.IndexFlatIP(VECTOR_DIM)
3 label_map = []
4
5 for idx, row in tqdm(df.iterrows(), total=len(df)):
6     features = extract_feature(row['image_path'],
7                                 face_recognition_model)
8     index.add(np.array([features]))
9     label_map.append(row['label'])
```

Xây dựng FAISS index với features

```

1 faiss.write_index(index, "facenet_features.index")
2 np.save("facenet_label_map.npy", np.array(label_map))

```

9.5.5 Cập nhật lại hàm tìm kiếm

Thêm hàm chuyển ảnh từ đường dẫn thành vector đặc trưng (embedding) bằng model đã huấn luyện. Ảnh được chuyển về tensor và qua model để lấy embedding, sau đó trả về dưới dạng numpy array.

Hàm chuyển ảnh thành vector feature

```

1 def image_to_feature(image_path, model):
2     """Convert image to face embedding using a pre-trained model"""
3     img = Image.open(image_path).convert('RGB')
4     img_tensor = transform(img).unsqueeze(0) # Add batch dimension
5     # Get the embedding
6     with torch.no_grad(): # Disable gradient calculation
7         embedding = model(img_tensor)
8
9     # Return the embedding as a numpy array
10    return embedding.squeeze().numpy()

```

Cập nhật lại hàm tìm kiếm ảnh tương đồng

```

1 def search_similar_images(query_image_path, k=5):
2     """Search for similar employee images using VGG16 features"""
3     # Load index and labels
4     index = faiss.read_index("facenet_features.index")
5     label_map = np.load("facenet_label_map.npy")
6
7     # Convert query image to vector
8     query_vector = image_to_feature(query_image_path,
9                                     face_recognition_model)
10
11    # Search in Faiss
12    similarities, indices = index.search(np.array([query_vector]), k)

```

```
12     # Get results
13     results = []
14     for i in range(len(indices[0])):
15         employee_name = label_map[indices[0][i]]
16         similarity = similarities[0][i]
17         results.append((employee_name, similarity))
18
19
20     return results
```

Hàm hiển thị top 5 ảnh tương đồng

```
1 def display_query_and_top_matches(query_image_path):
2     # Display query image
3     query_img = Image.open(query_image_path)
4     query_img = query_img.resize((300, 300))
5     plt.figure(figsize=(5, 5))
6     plt.imshow(query_img)
7     plt.title("Query Image")
8     plt.axis('off')
9     plt.show()
10
11     # Get matches
12     matches = search_similar_images(query_image_path)
13
14     # Display top matches
15     plt.figure(figsize=(15, 5))
16     for i, (name, similarity) in enumerate(matches):
17         # Find the image path for this employee
18         img_path = df[df['label'] == name]['image_path'].values[0]
19         img = Image.open(img_path)
20         img = img.resize((300, 300))
21
22         plt.subplot(1, 5, i+1)
23         plt.imshow(img)
24         plt.title(f"{name}\nSimilarity: {similarity:.2f}")
25         plt.axis('off')
26
27     plt.tight_layout()
28     plt.show()
```

Kiểm tra độ hiệu quả khi chuyển ảnh thành vector feature

Kết quả vector feature từ mô hình deep learning có hiệu quả vượt trội hơn raw pixel trong bài toán nhận diện khuôn mặt. Raw pixel vector (270,000 chiều) chỉ chứa thông tin màu sắc tại từng điểm ảnh và rất nhạy cảm với thay đổi ánh sáng, góc chụp, biểu cảm. Ngược lại, feature vector (512 chiều) từ FaceNet đã được huấn luyện để trích xuất các đặc trưng có ý nghĩa như khoảng cách mắt, hình dạng khuôn mặt, và bất biến với các yếu tố nhiễu.



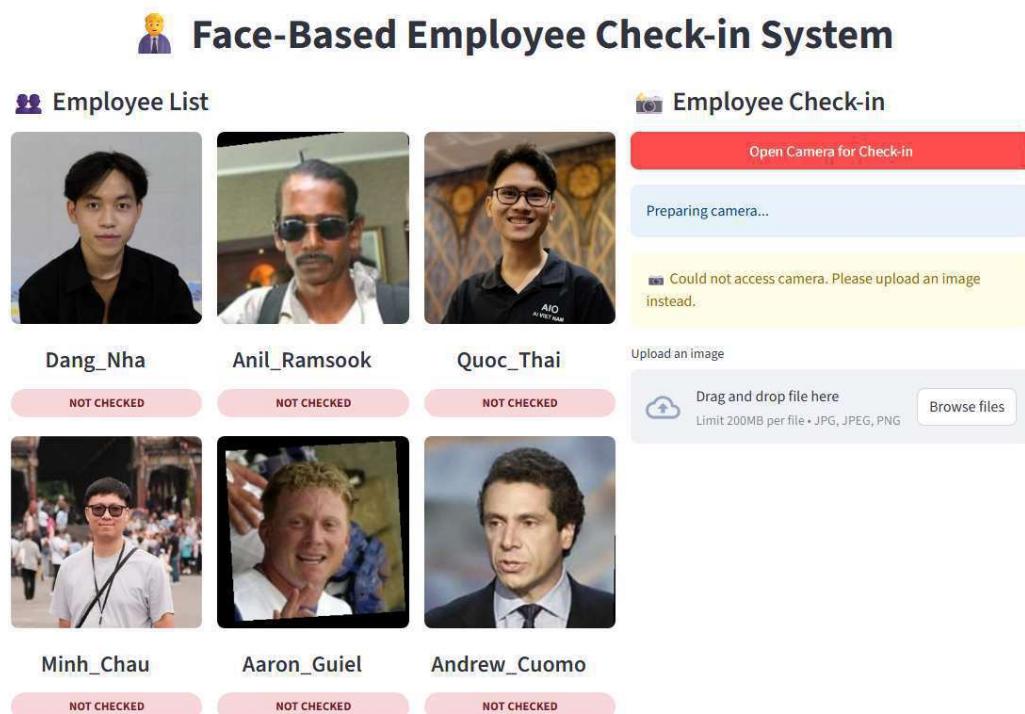
Hình 11: Đánh giá với hình ảnh không có trong dataset khi chuyển ảnh thành vector feature

Build Streamlit App

Tham khảo source code đầy đủ tại GitHub: https://github.com/dangnha/AIO_Module02_Project_Demo

Repository này bao gồm:

- Mã nguồn triển khai giao diện nhận diện khuôn mặt bằng Streamlit.
- Tích hợp mô hình Facenet, thư viện FAISS và PyTorch.
- Chức năng **check-in nhân viên theo ảnh khuôn mặt**.



Hình 12: Giao diện Streamlit khi build

9.6 Project 2: Cocktail Suggestion System

Mô tả dự án

Project: Hệ thống gợi ý cocktail theo mô tả cảm quan (vị chua, ngọt, ít cồn, đắng, ...)

Project Objectives:

- Ứng dụng sử dụng **pgvector** để lưu trữ và truy vấn các mô tả đặc trưng của cocktail dưới dạng vector hoá embedding.
- **Vector hoá mô tả cocktail**
 - Mỗi cocktail được gán một đoạn mô tả về mùi vị và đặc tính (VD: “ngot nhẹ, chua thanh, ít đắng, cồn thấp”).
 - Dùng mô hình embedding ngôn ngữ (như Sentence-BERT hoặc OpenAI Embedding API) để chuyển mô tả thành vector số cố định.
- **Lưu trữ bằng pgvector**
 - Các vector embedding được lưu vào PostgreSQL với extension **pgvector**, cho phép tìm kiếm gần đúng (approximate nearest neighbors).
- **Truy vấn dựa trên cảm quan người dùng**
 - Người dùng nhập yêu cầu: “chua, ngọt, ít cồn, không đắng”.
 - Hệ thống vector hoá truy vấn này và so sánh với database để tìm ra top 5 cocktail phù hợp nhất dựa trên độ tương đồng cosine.
- **Build ứng dụng demo với Streamlit**
 - Giao diện đơn giản cho người dùng nhập mô tả vị mong muốn.
 - Trả về danh sách các cocktail gợi ý cùng mô tả.

Thông tin Dataset

Nguồn: Kaggle – Cocktails Dataset by Aadya Singh

Định dạng: CSV

Dung lượng: ~53KB

Số dòng: 664

Số cột: 7

Các trường dữ liệu trong dataset:

- **Cocktail (string):** Tên của loại cocktail.
- **Ingredients (string):** Danh sách nguyên liệu, cách nhau bằng dấu phẩy.
- **Measure (string):** Định lượng tương ứng với từng nguyên liệu.
- **Category (string):** Phân loại cocktail (ví dụ: Alcoholic, Non-Alcoholic).
- **Glass (string):** Loại ly sử dụng để phục vụ cocktail.
- **Instructions (string):** Hướng dẫn cách pha chế.
- **Description (string):** Mô tả cảm quan về cocktail (ví dụ: “sweet and refreshing”).

Hướng dẫn từng bước

Bước 1: Thiết lập cơ sở dữ liệu

- Kết nối database tới PostgreSQL.
- Thiết lập pgvector

Bước 2: Xử lý dữ liệu

- Làm sạch dữ liệu.
- Tạo embeddings.
- Tạo công thức cocktail

- Lưu cocktail được embedding vào database

Bước 3: Hệ thống recommendation

- Tạo embedding cho query
- Query để thực hiện recommendation bằng cách sử dụng pgvector với cosine similarity.

Bước 4: Trả về recommendations

- Hiện thị kết quả

Kết luận: Hệ thống này cung cấp một giải pháp hoàn chỉnh để quản lý và gợi ý cocktail dựa trên nội dung, sử dụng các pgvector để indexing và SentenceTransformer để tạo embeddings trong xử lý ngôn ngữ tự nhiên và vector database.

9.6.1 Thiết lập kết nối

Nhắc lại một tí về pgvector là một PostgreSQL extension cho phép lưu trữ và truy vấn dữ liệu vector embedding trực tiếp trong cơ sở dữ liệu. Nó hỗ trợ các phép đo độ tương đồng như cosine similarity, inner product, và Euclidean distance, rất hữu ích cho các bài toán về tìm kiếm gần đúng (Approximate Nearest Neighbor - ANN). Để sử dụng pgvector, cần:

- Cài đặt extension trong PostgreSQL.
- Cài đặt Python package psycopg2 để giao tiếp với cơ sở dữ liệu.

Tham khảo chi tiết theo link sau: https://github.com/ThuanNaN/aio2025_cocktail_suggestions/blob/main/SETUP_GUIDE.md

Tạo database

Thiết lập kết nối tới database postgres với chế độ AUTOCOMMIT. Tạo database nếu chưa có. Đóng kết nối và cursor sau khi hoàn thành

Hàm tạo database

```

1 def create_database(self):
2     """Create the database if it doesn't exist"""
3     try:
4         # Connect to default postgres database
5         conn = psycopg2.connect(
6             host=self.host,
7             port=self.port,
8             user=self.user,
9             password=self.password,
10            database='postgres'
11        )
12        conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)
13        cursor = conn.cursor()
14
15        # Check if database exists
16        cursor.execute(f"SELECT 1 FROM pg_catalog.pg_database WHERE
17                         datname = '{self.db_name}'")
18        exists = cursor.fetchone()
19
20        if not exists:
21            cursor.execute(f'CREATE DATABASE {self.db_name}')
22            print(f"Database '{self.db_name}' created successfully")
23        else:
24            print(f"Database '{self.db_name}' already exists")
25
26        cursor.close()
27        conn.close()

```

Thiết lập pgvector extension

Sau khi thiết lập chúng ta cần:

- Tạo cursor và pgvector
- Tạo bảng cocktails với các trường thông tin cơ bản và trường vector(384) để lưu embeddings
- Tạo index ivfflat để tối ưu tìm kiếm similarity (cosine similarity). Tham số lists = 100 để cân bằng giữa tốc độ và độ chính xác

- Commit và đóng kết nối

Hàm tạo database

```
1 def setup_pgvector(self):
2     """"
3         Setup pgvector extension and create tables"""
4     try:
5         conn = psycopg2.connect(
6             host=self.host,
7             port=self.port,
8             user=self.user,
9             password=self.password,
10            database=self.db_name
11        )
12        cursor = conn.cursor()
13
14        # Enable pgvector extension
15        cursor.execute("CREATE EXTENSION IF NOT EXISTS vector")
16
17        # Create cocktails table with vector embeddings
18        cursor.execute("""
19            CREATE TABLE IF NOT EXISTS cocktails (
20                id SERIAL PRIMARY KEY,
21                name VARCHAR(255) NOT NULL,
22                ingredients TEXT NOT NULL,
23                recipe TEXT,
24                glass VARCHAR(100),
25                category VARCHAR(100),
26                iba VARCHAR(100),
27                alcoholic VARCHAR(50),
28                embedding vector(384)
29            )
30        """
31
32        # Create index for vector similarity search
33        cursor.execute("""
34            CREATE INDEX IF NOT EXISTS cocktails_embedding_idx
35            ON cocktails USING ivfflat (embedding vector_cosine_ops)
36            WITH (lists = 100)
37        """
38
39        conn.commit()
40        cursor.close()
41        conn.close()
```

```

41
42     print("Database tables and pgvector extension set up
        successfully")
43
44 except Exception as e:
45     print(f"Error setting up pgvector: {e}")

```

9.6.2 Xử lý dữ liệu

Làm sạch dữ liệu

Các bước xử lý bao gồm:

- Tạo định dạng mới cho các field dữ liệu
- Loại bỏ bản ghi trùng lặp dựa trên tên cocktail. Thay thế các giá trị null bằng chuỗi rỗng
- Kết hợp các trường thông tin quan trọng thành một trường combined_text để chuẩn bị cho embedding.

Clean data

```

1 def clean_data(self, df):
2     """Clean and preprocess the cocktail data"""
3     # Auto-detect column names (handle both old and new formats)
4     name_col = 'name' if 'name' in df.columns else 'strDrink'
5     category_col = 'category' if 'category' in df.columns else 'strCategory'
6     alcoholic_col = 'alcoholic' if 'alcoholic' in df.columns else 'strAlcoholic'
7     glass_col = 'glassType' if 'glassType' in df.columns else 'strGlass'
8     instructions_col = 'instructions' if 'instructions' in df.
9                                     columns else 'strInstructions'
10
11    print(f"Detected columns: name={name_col}, category={category_col},
12          alcoholic={alcoholic_col}, glass={glass_col}")

```

```
12     # Remove duplicates based on name
13     if name_col in df.columns:
14         df = df.drop_duplicates(subset=[name_col])
15         print(f"After removing duplicates: {len(df)} cocktails")
16
17     # Fill missing values
18     df = df.fillna('')
19
20     # Create a combined text for embedding
21     df['combined_text'] = ''
22
23     if name_col in df.columns:
24         df['combined_text'] += df[name_col].astype(str) + ','
25     if category_col in df.columns:
26         df['combined_text'] += df[category_col].astype(str) + ','
27     if alcoholic_col in df.columns:
28         df['combined_text'] += df[alcoholic_col].astype(str) + ','
29     if glass_col in df.columns:
30         df['combined_text'] += df[glass_col].astype(str) + ','
31
32     # Handle ingredients (could be in different formats)
33     if 'ingredients' in df.columns:
34         # New format: ingredients as string/list
35         df['combined_text'] += df['ingredients'].astype(str) + ','
36     else:
37         # Old format: strIngredient1, strIngredient2, etc.
38         ingredient_cols = [col for col in df.columns if col.
39                             startswith('strIngredient')]
40         for col in ingredient_cols:
41             df['combined_text'] += df[col].astype(str) + ','
42
43     # Add instructions if available
44     if instructions_col in df.columns:
45         df['combined_text'] += df[instructions_col].astype(str) + ','
46
47     # Clean the combined text
48     df['combined_text'] = df['combined_text'].str.replace(r'\s+', ' '
49                                         , regex=True).str.strip()
50
51     print(f"Sample combined text: {df['combined_text'].iloc[0][:100]
52           }...")
53
54     return df
```

Tạo embeddings từ văn bản

Khác với project trước lần này embedding ngôn ngữ thay vì hình ảnh. Sử dụng thư viện SentenceTransformer với mô hình all-MiniLM-L6-v2 là một trong các mô hình nhỏ gọn (mini) được huấn luyện dựa trên kiến trúc MiniLM.

Tạo model embedding

```
1 def __init__(self):
2     self.model_name = os.getenv('MODEL_NAME', 'all-MiniLM-L6-v2')
3     self.model = SentenceTransformer(self.model_name)
4     self.db_setup = DatabaseSetup()
```

Generate embeddings

```
1 def __init__(self):
2     self.model_name = os.getenv('MODEL_NAME', 'all-MiniLM-L6-v2')
3     self.model = SentenceTransformer(self.model_name)
4     self.db_setup = DatabaseSetup()
```

Tạo công thức cocktail

Tạo công thức cocktail từ ingredients

```
1 def create_recipe_text(self, row):
2     """Create a readable recipe from the row data"""
3     # Auto-detect column names
4     name_col = 'name' if 'name' in row else 'strDrink'
5     category_col = 'category' if 'category' in row else 'strCategory'
6     alcoholic_col = 'alcoholic' if 'alcoholic' in row else 'strAlcoholic'
7     glass_col = 'glassType' if 'glassType' in row else 'strGlass'
```

```
8     instructions_col = 'instructions' if 'instructions' in row else
9             'strInstructions'
10
11    recipe = f"Drink: {row.get(name_col, '')}\n"
12    recipe += f"Category: {row.get(category_col, '')}\n"
13    recipe += f"Type: {row.get(alcoholic_col, '')}\n"
14    recipe += f"Glass: {row.get(glass_col, '')}\n"
15
16    if row.get(instructions_col):
17        recipe += f"Instructions: {row[instructions_col]}\n"
18
19    recipe += "Ingredients:\n"
20
21    # Handle new format (ingredients as string/list)
22    if 'ingredients' in row and row['ingredients']:
23        try:
24            import ast
25            ingredients_str = row['ingredients']
26
27            # Parse ingredients list
28            if ingredients_str.startswith('['):
29                ingredients = ast.literal_eval(ingredients_str)
30            else:
31                ingredients = [ingredients_str]
32
33            # Parse measures if available
34            measures = []
35            if 'ingredientMeasures' in row and row[
36                            'ingredientMeasures']:
37                measures_str = row['ingredientMeasures']
38                if measures_str.startswith('['):
39                    measures = ast.literal_eval(measures_str)
40                else:
41                    measures = [measures_str]
42
43            # Combine ingredients with measures
44            for i, ingredient in enumerate(ingredients):
45                if ingredient and str(ingredient).strip() and str(
46                    ingredient).strip() != 'None':
47                    if i < len(measures) and measures[i] and str(
48                        measures[i]).strip() != 'None':
49                        recipe += f"- {measures[i]} {ingredient}\n"
50                    else:
51                        recipe += f"- {ingredient}\n"
```

```

48
49     except Exception as e:
50         # Fallback: treat as simple string
51         recipe += f"- {row['ingredients']}\n"
52
53     else:
54         # Handle old format (strIngredient1, strIngredient2, etc.)
55         for i in range(1, 16): # Assuming max 15 ingredients
56             ingredient = row.get(f'strIngredient{i}')
57             measure = row.get(f'strMeasure{i}')
58             if ingredient and str(ingredient).strip() and str(
59                 ingredient).strip() != 'nan':
60                 if measure and str(measure).strip() and str(measure)
61                     .strip() != 'nan':
62                     recipe += f"- {measure} {ingredient}\n"
63             else:
64                 recipe += f"- {ingredient}\n"
65
66     return recipe

```

Trích xuất danh sách nguyên liệu

Hàm lấy nguyên liệu để user mix

```

1 def get_ingredients_list(self, row):
2     """Extract ingredients as a comma-separated string"""
3     ingredients = []
4
5     # Handle new format (ingredients as string/list)
6     if 'ingredients' in row and row['ingredients']:
7         try:
8             import ast
9             ingredients_str = row['ingredients']
10
11             # Parse ingredients list
12             if ingredients_str.startswith('['):
13                 ingredients_list = ast.literal_eval(ingredients_str)
14                 for ingredient in ingredients_list:
15                     if ingredient and str(ingredient).strip() and
16                         str(ingredient).strip() != 'None':
17                         ingredients.append(str(ingredient).strip())

```

```

17     else:
18         # Single ingredient as string
19         if ingredients_str.strip():
20             ingredients.append(ingredients_str.strip())
21
22     except Exception as e:
23         # Fallback: treat as simple string
24         if row['ingredients'].strip():
25             ingredients.append(row['ingredients'].strip())
26
27     else:
28         # Handle old format (strIngredient1, strIngredient2, etc.)
29         for i in range(1, 16):
30             ingredient = row.get(f'strIngredient{i}')
31             if ingredient and str(ingredient).strip() and str(
32                 ingredient).strip() != 'nan':
33                 ingredients.append(str(ingredient).strip())
34
35     return ', '.join(ingredients)

```

Lưu cocktail đã được embedding vào database

Hàm lưu trữ cocktails

```

1 def store_cocktails(self, df):
2     """Store cocktails with embeddings in the database"""
3     try:
4         conn = self.db_setup.get_connection()
5         cursor = conn.cursor()
6
7         # Clear existing data
8         cursor.execute("DELETE FROM cocktails")
9
10        print(f"Generating embeddings for {len(df)} cocktails...")
11        # Generate all embeddings at once (much more efficient)
12        all_embeddings = self.generate_embeddings(df['combined_text']
13                                         ].tolist())
14
15        print("Storing cocktails in database...")
16        for idx, (_, row) in enumerate(df.iterrows()):
17            # Get pre-computed embedding

```

```
17     embedding = all_embeddings[idx]
18
19     # Prepare data with auto-detected column names
20     name_col = 'name' if 'name' in row else 'strDrink'
21     category_col = 'category' if 'category' in row else ,
22                         strCategory'
23     alcoholic_col = 'alcoholic' if 'alcoholic' in row else ,
24                         strAlcoholic'
25     glass_col = 'glassType' if 'glassType' in row else ,
26                         strGlass'
27
28     name = row.get(name_col, '')
29     ingredients = self.get_ingredients_list(row)
30     recipe = self.create_recipe_text(row)
31     glass = row.get(glass_col, '')
32     category = row.get(category_col, '')
33     iba = row.get('strIBA', '') # This might not exist in
34                             new format
35     alcoholic = row.get(alcoholic_col, '')
36
37     # Insert into database
38     cursor.execute("""
39         INSERT INTO cocktails (name, ingredients, recipe,
40                               glass, category, iba, alcoholic,
41                               embedding)
42         VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
43         """, (name, ingredients, recipe, glass, category, iba,
44                alcoholic, embedding.tolist()))
45
46     if (idx + 1) % 100 == 0:
47         print(f"Stored {idx + 1} cocktails...")
48
49     conn.commit()
50     cursor.close()
51     conn.close()
52
53
54     print(f"Successfully stored {len(df)} cocktails in the
55          database")
56
57 except Exception as e:
58     print(f"Error storing cocktails: {e}")
59     if 'conn' in locals():
60         conn.rollback()
61         conn.close()
```



9.6.3 Hệ thống Recommendation

Embedding query của người dùng

Sử dụng cùng model embedding (all-MiniLM-L6-v2) như trong data processor để đảm bảo tính nhất quán. Số chiều cùng với số chiều các vector trong database là 384.

Khởi tạo model

```

1 def __init__(self):
2     self.model_name = os.getenv('MODEL_NAME', 'all-MiniLM-L6-v2')
3     self.model = SentenceTransformer(self.model_name)
4     self.db_setup = DatabaseSetup()

```

Hàm tạo embedding cho query người dùng

```

1 def get_user_preferences_embedding(self, preferences):
2     preference_text = ', '.join(preferences)
3     embedding = self.model.encode([preference_text])[0]
4     return embedding

```

Tìm kiếm cocktail tương đồng

Sử dụng toán tử \leqslant (cosine distance) của pgvector. Tính similarity = 1 - cosine_distance để được giá trị từ 0-1 (1 là giống nhất). Lọc chỉ lấy kết quả có similarity > ngưỡng chỉ định (mặc định 0.3)

Hàm tìm kiếm cocktail tương đồng

```

1 def search_similar_cocktails(self, query_embedding, limit=10,
2                               similarity_threshold=0.3):
3     """Search for similar cocktails using vector similarity"""
4     try:

```

```
4     conn = self.db_setup.get_connection()
5     cursor = conn.cursor()
6
7     # First check if we have any cocktails in the database
8     cursor.execute("SELECT COUNT(*) FROM cocktails")
9     count = cursor.fetchone()[0]
10    print(f"Database contains {count} cocktails")
11
12   if count == 0:
13       print("Warning: No cocktails found in database. Have you
14           run data_processor.py?")
15       cursor.close()
16       conn.close()
17       return []
18
19   # Convert numpy array to list and then to string format for
20   # pgvector
21   if hasattr(query_embedding, 'tolist'):
22       embedding_list = query_embedding.tolist()
23   else:
24       embedding_list = list(query_embedding)
25
26   # Use cosine similarity for search
27   cursor.execute("""
28       SELECT id, name, ingredients, recipe, glass, category,
29                   iba, alcoholic,
30                   1 - (embedding <=> %s::vector) as similarity
31       FROM cocktails
32       WHERE 1 - (embedding <=> %s::vector) > %s
33       ORDER BY similarity DESC
34       LIMIT %s
35   """ , (embedding_list, embedding_list, similarity_threshold,
36           limit))
37
38   results = cursor.fetchall()
39   print(f"Found {len(results)} cocktails with similarity > {
40         similarity_threshold}")
41   cursor.close()
42   conn.close()
43
44   return results
45
46 except Exception as e:
47     print(f"Error searching cocktails: {e}")
```

```

43     import traceback
44     traceback.print_exc()
45     return []

```

Các phương thức gợi ý theo nhu cầu

Hàm tìm kiếm cocktail tương đồng

```

1 def recommend_by_ingredients(self, ingredients, limit=10):
2     ingredients_text = f"cocktail with {' and '.join(ingredients)}"
3     query_embedding = self.get_user_preferences_embedding([
4         ingredients_text])
4     return self.search_similar_cocktails(query_embedding, limit)

```

Hàm gợi ý theo nguyên liệu

```

1 def recommend_by_ingredients(self, ingredients, limit=10):
2     ingredients_text = f"cocktail with {' and '.join(ingredients)}"
3     query_embedding = self.get_user_preferences_embedding([
4         ingredients_text])
4     return self.search_similar_cocktails(query_embedding, limit)

```

Hàm gợi ý theo phong cách

```

1 def recommend_by_style(self, style_preferences, limit=10):
2     style_text = f"cocktail that is {' and '.join(style_preferences)}
3     query_embedding = self.get_user_preferences_embedding([
4         style_text])
4     return self.search_similar_cocktails(query_embedding, limit)

```

Hàm gợi ý theo hoàn cảnh

```

1 def recommend_by_occasion(self, occasion, limit=10):
2     occasion_text = f"cocktail for {occasion}"

```

```

3     query_embedding = self.get_user_preferences_embedding([
4         occasion_text])
      return self.search_similar_cocktails(query_embedding, limit)

```

Hàm gợi ý tổng hợp

```

1 def recommend_by_mixed_preferences(self, ingredients=None, style=
2                                         None, occasion=None,
3                                         alcoholic_preference=None, limit=10
4                                         ):
5     preferences = []
6
7     if ingredients:
8         preferences.append(f"contains {' and '.join(ingredients)}")
9     if style:
10        preferences.append(f"is {' and '.join(style)}")
11    if occasion:
12        preferences.append(f"perfect for {occasion}")
13    if alcoholic_preference:
14        preferences.append(f"is {alcoholic_preference}")
15
16    if not preferences:
17        return []
18
19    query_embedding = self.get_user_preferences_embedding(
20        preferences)
21    return self.search_similar_cocktails(query_embedding, limit)

```

9.6.4 Các phương thức tìm kiếm cocktail

Hàm gợi ý tổng hợp

```

1 def get_cocktail_by_name(self, name):
2     cursor.execute("""
3         SELECT id, name, ingredients, recipe, glass, category, iba,
4             alcoholic
5             FROM cocktails
6             WHERE LOWER(name) LIKE LOWER(%s)
7             LIMIT 5

```

```
7     """", (f'#{name}%',))
```

Lấy ngẫu nhiên

```
1 def get_random_cocktails(self, limit=5):
2     cursor.execute("""
3         SELECT id, name, ingredients, recipe, glass, category, iba,
4             alcoholic
5         FROM cocktails
6         ORDER BY RANDOM()
7         LIMIT %s
8     """", (limit,))
```

Lọc theo danh mục

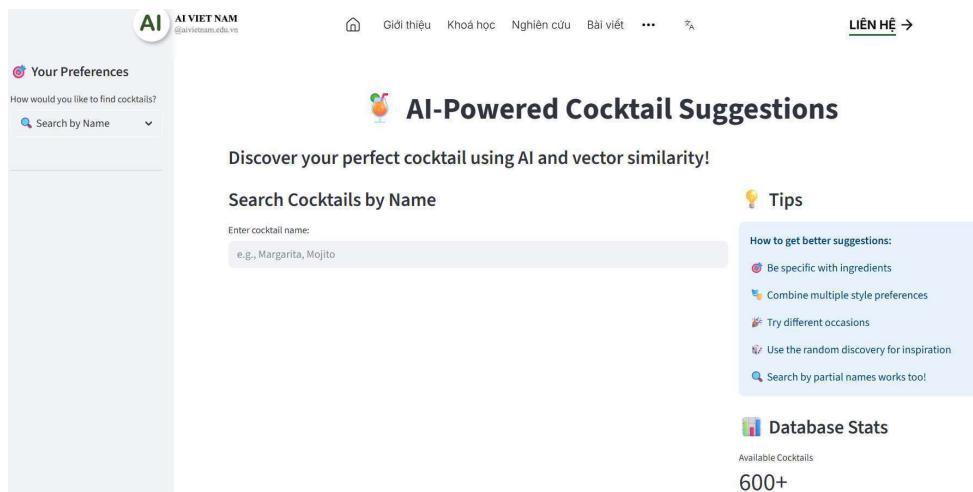
```
1 def get_cocktails_by_category(self, category, limit=10):
2     cursor.execute("""
3         SELECT id, name, ingredients, recipe, glass, category, iba,
4             alcoholic
5         FROM cocktails
6         WHERE LOWER(category) LIKE LOWER(%s)
7         ORDER BY name
8         LIMIT %s
9     """", (f'#{category}%', limit))
```

9.6.5 Build Streamlit App

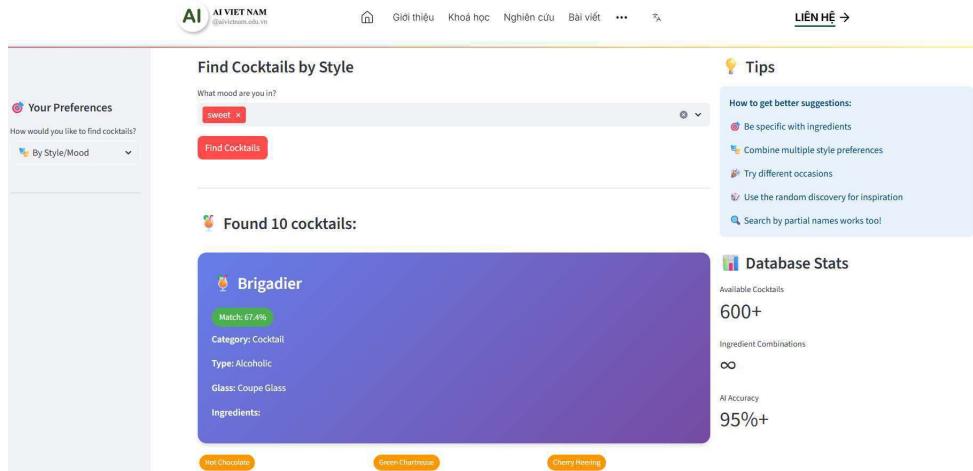
Tham khảo source code đầy đủ tại GitHub: https://github.com/ThuanNaN/aio2025_cocktail_suggestions

Repository này bao gồm:

- Mã nguồn hoàn chỉnh hệ thống gợi ý cocktail thông minh
- Tích hợp công nghệ vector embedding với Sentence-Transformers và pgvector
- Cơ sở dữ liệu PostgreSQL được tối ưu cho tìm kiếm ngữ nghĩa



Hình 13: Giao diện Streamlit Cocktail Suggestions



Hình 14: Giao diện Cocktail Suggestions tìm kiếm theo Style

9.7 Câu hỏi trắc nghiệm

1. Trong Vector Database, mục đích chính của quá trình indexing là gì?
 - (a) Nén dữ liệu để tiết kiệm dung lượng lưu trữ.
 - (b) Chuyển đổi vector về không gian thấp hơn để huấn luyện mô hình.
 - (c) Mã hóa vector thành chuỗi nhị phân.
 - (d) Tăng tốc độ truy vấn tìm kiếm vector tương tự.
2. Faiss chủ yếu được sử dụng cho mục đích nào trong các hệ thống AI?
 - (a) Tăng tốc huấn luyện mô hình học sâu.
 - (b) Giảm kích thước mô hình để triển khai trên thiết bị di động.
 - (c) Tìm kiếm vector tương tự trong không gian nhiều chiều.
 - (d) Mã hóa ảnh sang dạng nhị phân.
3. Chọn phát biểu đúng về chỉ mục IndexFlatL2 trong Faiss:
 - (a) Sử dụng thuật toán cây để tăng tốc độ tìm kiếm.
 - (b) Không cần huấn luyện trước và dùng khoảng cách Euclidean để so sánh vector.
 - (c) Là chỉ mục nén dữ liệu theo kỹ thuật PQ (Product Quantization).
 - (d) Hỗ trợ tìm kiếm theo cosine similarity.
4. Ở phần lưu image thành vector raw pixel, đâu là distance loss function được áp dụng cho search similarity?
 - (a) L1.
 - (b) L2.
 - (c) Dot Product (Inner Product).
 - (d) Cosine Similarity.
5. Dữ liệu tiên huấn luyện của mô hình pretrained InceptionResnetV1 là?
 - (a) ImageNet.

- (b) vggface2.
- (c) CIFAR-100.
- (d) MS-Celeb-1M.
6. Ở phần lưu image thành feature map, đâu là distance loss function được áp dụng cho search similarity?
- (a) L1.
- (b) L2.
- (c) Dot Product (Inner Product).
- (d) Cosine Similarity.
7. Trong PostgreSQL với extension pgvector, chỉ mục ivfflat được dùng để làm gì?
- (a) Lưu trữ vector theo định dạng nén để tiết kiệm dung lượng.
- (b) Tăng tốc tìm kiếm vector bằng cách phân vùng không gian vector thành nhiều danh sách (lists).
- (c) Mã hóa vector để tăng tính bảo mật khi lưu trữ.
- (d) Chuyển đổi vector từ không gian Euclidean sang không gian cosine.
8. Trong hàm `get_ingredients_list`, tại sao cần sử dụng `ast.literal_eval()` để xử lý chuỗi nguyên liệu?
- (a) Để biến chuỗi dạng list (ví dụ "[‘vodka’, ‘lemon’]") thành danh sách (lists).
- (b) Để chuẩn hóa các tên nguyên liệu thành chữ thường (lowercase).
- (c) Để mã hóa nguyên liệu thành embedding.
- (d) Để loại bỏ tất cả các giá trị rỗng và null khỏi chuỗi nguyên liệu.
9. Trong hàm `store_cocktails`, đoạn mã `conn.rollback()` trong khối `except` có mục đích gì?
- (a) Đóng kết nối cơ sở dữ liệu sau khi lưu thành công.
- (b) Xóa toàn bộ dữ liệu của bảng cocktails để chuẩn bị lưu lại từ đầu.

- (c) Hoàn tác (undo) toàn bộ thay đổi chưa commit nếu có lỗi xảy ra trong quá trình lưu dữ liệu.
- (d) Khôi phục kết nối nếu bị timeout khi lưu dữ liệu.
10. Trong truy vấn SQL, biểu thức 1 - (`embedding <=> %s::vector`) được sử dụng để tính độ tương đồng giữa vector query và vector trong database. Mục đích của phép tính đó là gì?
- (a) Chuyển đổi kết quả khoảng cách cosine thành độ tương đồng cosine (giá trị càng gần 1 càng giống).
- (b) Chuẩn hóa embedding về độ dài bằng 1 trước khi lưu vào cơ sở dữ liệu.
- (c) Tính khoảng cách Euclidean thay vì cosine.
- (d) Đảm bảo rằng tất cả các vector có cùng kích thước trước khi so sánh.

- Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

2. Kiến thức cần cho project:

Để hoàn thành và hiểu rõ phần project một cách hiệu quả nhất, các học viên cần nắm rõ các kiến thức được trang bị

- Kiến thức lập trình Python và xử lý hình ảnh
- Hiểu về cơ sở dữ liệu vector (vector database) và các thuật toán indexing như IVF, PQ
- Cách thao tác với các thư viện như FAISS, PyTorch, Streamlit
- Kiến thức về embedding và các mô hình trích xuất đặc trưng như InceptionResnetV1
- Các độ đo khoảng cách trong không gian vector như L2, cosine similarity

3. Rubric:

Mục	Kiến Thức	Đánh Giá
III.	<ul style="list-style-type: none"> - Lý thuyết về vector database - Các loại indexing trong vector database: Flat, IVF, PQ - Các phương pháp embedding văn bản/hình ảnh - Các kỹ thuật indexing trong vector database (Flat, IVF, PQ) 	<ul style="list-style-type: none"> - Hiểu và giải thích được khái niệm vector database - Hiểu được khái niệm embedding - Lựa chọn và triển khai được phương pháp indexing tối ưu

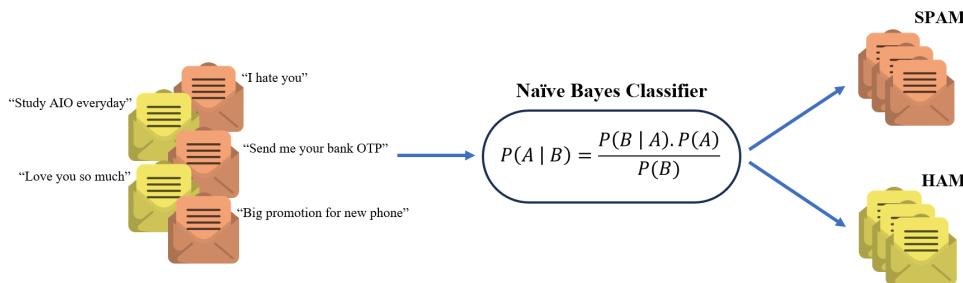
IV.	<ul style="list-style-type: none"> - Lý thuyết về Distance Metrics and Loss Functions - Công thức L2 Distance - Công thức L1 Distance - Đặc điểm Cosine Similarity và Cosine Distance - Giải thích Dot Product 	<ul style="list-style-type: none"> - Phân biệt được ứng dụng của từng loại khoảng cách - Lựa chọn được metric phù hợp cho từng bài toán
V.	<ul style="list-style-type: none"> - Face-Based Employee Check-in System - Chuẩn bị tập dữ liệu khuôn mặt - Vector hóa ảnh thành raw pixel - Vector hóa ảnh thành feature maps - Xây dựng hệ thống tìm kiếm và hiển thị 	<ul style="list-style-type: none"> - Biết cách dùng indexing trong FAISS. - Nắm được ý tưởng embedding ảnh thành vector feature. - Đọc hiểu, triển khai được pipeline vector hóa ảnh - Xây dựng được hệ thống nhận diện hoàn chỉnh
VI.	<ul style="list-style-type: none"> - Cocktail Suggestion System - Thiết lập kết nối PostgreSQL với pgvector - Vector hóa ảnh thành raw pixel - Vector hóa ảnh thành feature maps - Xây dựng hệ thống tìm kiếm và hiển thị 	<ul style="list-style-type: none"> - Nắm được cách cấu hình PostgreSQL với pgvector - Cách sử dụng pgvector để indexing - Hiểu ý tưởng sử dụng mô hình all-MiniLM-L6-v2 (SentenceTransformer) để embedding câu hỏi truy vấn - Cách tìm kiếm bằng cosine similarity trong pgvector.

Chương 10

Project 2: Đánh giá cảm xúc khách hàng dùng Vector database

10.1 Giới thiệu

Text Classification (Tạm dịch: Phân loại văn bản) là một trong những bài toán phổ biến trong lĩnh vực Machine Learning và Natural Language Processing. Trong đó, nhiệm vụ của chúng ta là xây dựng một chương trình có khả năng phân loại văn bản vào các phân lớp do chúng ta quy định. Các ứng dụng phổ biến liên quan đến loại chương trình này có thể kể đến phát hiện các bình luận tiêu cực trên không gian mạng, các đánh giá tích cực của sản phẩm...



Hình 10.1: Minh họa ứng dụng phân loại tin nhắn rác (spam) và tin nhắn thường (ham).

Trong project này, chúng ta sẽ xây dựng một chương trình Text Classification liên quan đến việc phân loại một đoạn tin nhắn là tin nhắn spam hay không. Dự án sẽ khám phá và triển khai hai phương pháp tiếp cận chính để giải quyết bài toán này:

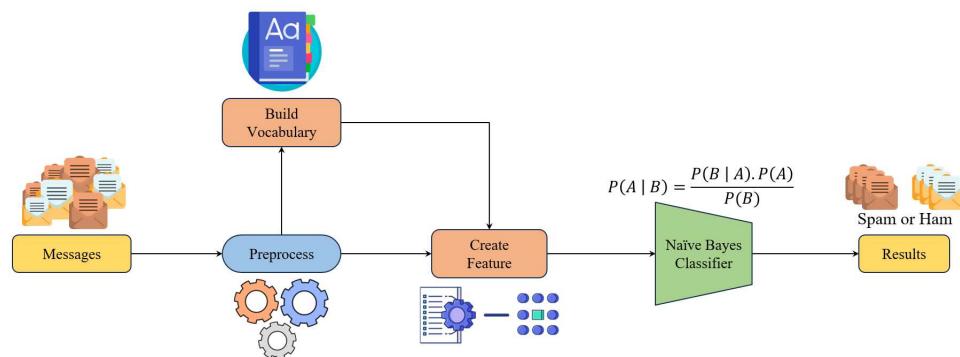
- **Naive Bayes Classifier:** Một thuật toán học máy truyền thống dựa trên định lý Bayes với giả định độc lập có điều kiện giữa các đặc trưng, thường hiệu quả cho các bài toán phân loại văn bản.
- **Phân loại sử dụng Cơ sở dữ liệu Vector (Vector Database) và Sentence Embeddings:** Một phương pháp hiện đại hơn, trong đó các

tin nhắn được biểu diễn dưới dạng vector ngữ nghĩa (embeddings) và việc phân loại dựa trên việc tìm kiếm sự tương đồng trong một cơ sở dữ liệu vector tốc độ cao như FAISS.

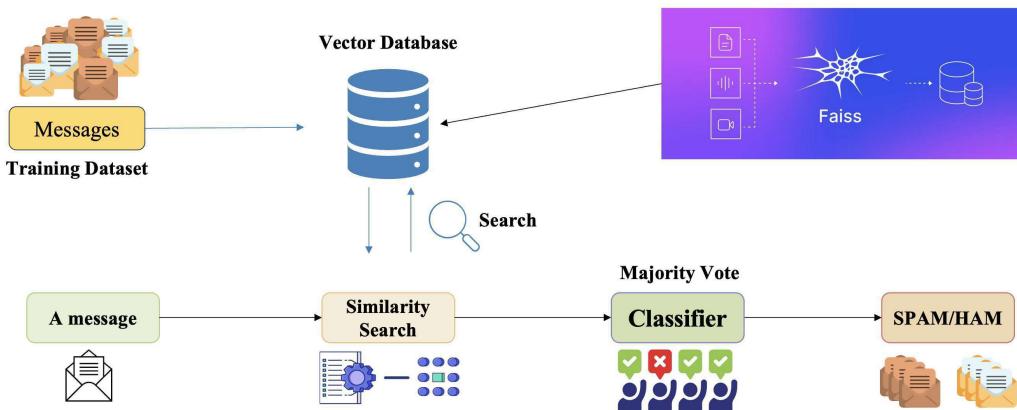
Theo đó, Input/Output chung của chương trình sẽ bao gồm:

- **Input:** Một đoạn tin nhắn (text).
- **Output:** Có là tin nhắn spam hay không (bool).

Dựa vào các thông tin trên, ta sẽ xây dựng được 2 luồng xử lý (pipeline) cho bài toán này như sau:



Hình 10.2: Pipeline tổng quan của chương trình phân loại tin nhắn với Naive Bayes.



Hình 10.3: Pipeline tổng quan của chương trình phân loại tin nhắn với Cơ sở dữ liệu Vector.

Theo đó, với bộ dữ liệu có nhãn về tin nhắn spam hoặc không spam, cả hai phương pháp đều trải qua các bước chuẩn bị dữ liệu cần thiết, bao gồm tiền xử lý và trích xuất đặc trưng. Sau khi dữ liệu được chuẩn bị, tùy thuộc vào phương pháp lựa chọn, chúng ta sẽ tiến hành xây dựng mô hình Naive Bayes Classifier hoặc tạo cơ sở dữ liệu vector để lưu trữ các biểu diễn ngữ nghĩa của tin nhắn. Cuối cùng, sử dụng mô hình hoặc cơ sở dữ liệu đã xây dựng, chương trình có thể dự đoán một tin nhắn bất kỳ có phải là spam hay không. Như vậy, các chương trình trong project của chúng ta đã hoàn tất.

10.2 Xây dựng chương trình phân loại tin nhắn Spam với Naive Bayes

Trong phần này, ta sẽ tiến hành cài đặt chương trình phân loại tin nhắn rác hay không với Naive Bayes. Chương trình được cài đặt trên Google Colab.

10.2.1 Tải và khám phá bộ dữ liệu

Đầu tiên, chúng ta cùng nhìn qua bộ dataset sẽ được sử dụng trong bài này thông qua bảng sau:

Category	Message
ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got a...
ham	Ok lar... Joking wif u oni...
spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entr...
spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it s...
...	...

Quan sát thấy bộ dữ liệu sẽ gồm có 2 cột:

1. **Category:** gồm 2 nhãn là *Ham* và *Spam*, với ý nghĩa như sau:

- *Ham:* Là những tin nhắn bình thường, không có mục đích quảng cáo hoặc lừa đảo hoặc nói cách khác là người nhận mong muốn nhận được.
- *Spam:* Là những tin nhắn không mong muốn, thường có mục đích quảng cáo sản phẩm, dịch vụ, hoặc lừa đảo.

2. **Message:** là những nội dung bên trong một Messages.

Nhiệm vụ của chúng ta là dựa vào nội dung Message để phân loại nhị phân với Naive Bayes, để xem xét rằng, liệu với nội dung như thế này thì Message đó là *Spam* hay *Ham*. Để huấn luyện mô hình Naive Bayes giải quyết bài toán này, ta cần tải bộ dữ liệu này về máy. Chúng ta có thể tải tại [đây](#) hoặc

trực tiếp tại trang Kaggle của dataset này tại [đây](#). Trong python, với đường dẫn google drive của bộ dữ liệu, ta có thể dùng lệnh sau đây để tải về một cách tự động về máy:

```
1 # https://drive.google.com/file/d/1N7rk-kfnDFIGMeXOROVThKh71gcgx-7R/
   view?usp=sharing
2 !gdown --id 1N7rk-kfnDFIGMeXOROVThKh71gcgx-7R
```

10.2.2 Cài đặt và Import thư viện

Trước tiên, chúng ta cùng điểm qua một số thư viện chính được sử dụng trong bài:

- **string**: Cung cấp các hàm cơ bản để thao tác với chuỗi ký tự.
- **nltk (Natural Language Toolkit)**: Một trong những thư viện xử lý ngôn ngữ tự nhiên phổ biến nhất trong Python.
- **pandas**: Cung cấp các cấu trúc dữ liệu hiệu quả và các công cụ để làm việc với dữ liệu.
- **numpy**: Cung cấp các đối tượng mảng đa chiều và các hàm toán học để làm việc với các mảng này.
- **scikit-learn**: Thư viện học máy phổ biến, giúp xây dựng và triển khai các mô hình học máy phức tạp một cách nhanh chóng.



Hình 10.4: Logo của một số thư viện được sử dụng trong project.

Trong môi trường code, các bạn thực thi đoạn code sau:

```

1 import string
2 import nltk
3 nltk.download("stopwords")
4 nltk.download("punkt")
5 import pandas as pd
6 import numpy as np
7 import matplotlib.pyplot as plt
8
9 from sklearn.model_selection import train_test_split
10 from sklearn.naive_bayes import GaussianNB
11 from sklearn.metrics import accuracy_score
12 from sklearn.preprocessing import LabelEncoder

```

10.2.3 Đọc và tách dữ liệu

Để đọc bộ dữ liệu có dạng file ‘.csv’, chúng ta sẽ dùng thư viện **pandas**. Để tách riêng biệt phần đặc trưng và nhãn, sau khi có dataframe, chúng ta đọc và lưu trữ dữ liệu của từng cột vào 2 biến tương ứng **messages** và **labels**:

```

1 DATASET_PATH = "/content/2cls_spam_text_cls.csv"
2 df = pd.read_csv(DATASET_PATH)
3
4 messages = df[["Message"]].values.tolist()
5 labels = df[["Category"]].values.tolist()

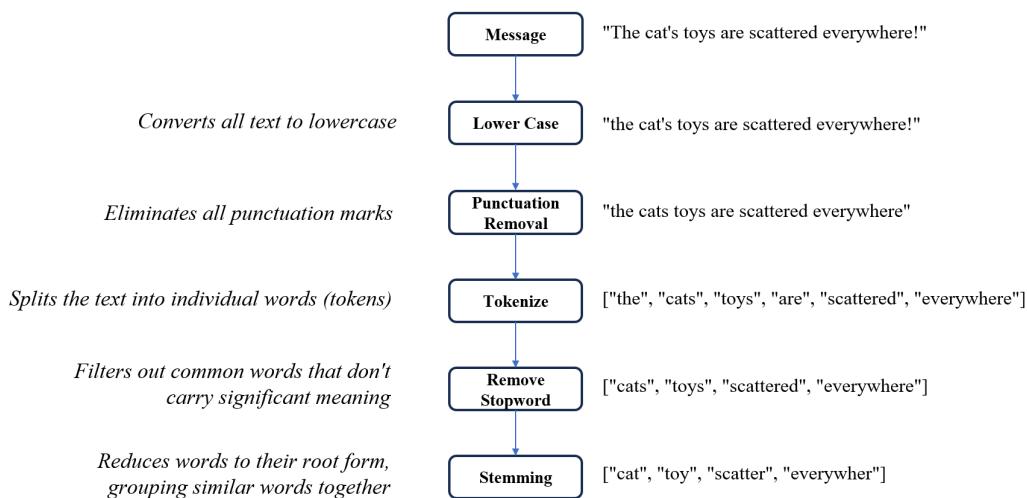
```

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...

Hình 10.5: Hiển thị một vài mẫu dữ liệu đầu tiên từ file csv.

10.2.4 Tiền xử lý dữ liệu

Tiền xử lý dữ liệu đặc trưng: Nội dung của các tin nhắn vô cùng đa dạng. Chúng có thể chứa từ viết tắt, dấu câu, các biến thể của từ, v.v. Vì vậy, bước tiền xử lý dữ liệu là vô cùng quan trọng. Chúng ta sẽ thực hiện một số bước xử lý cơ bản như sau:



Hình 10.6: Minh họa các bước tiền xử lý văn bản.

Tương ứng với đoạn code sau:

```

1 def lowercase(text):
2     return # Your code here
3
4 def punctuation_removal(text):
5     translator = # Your code here
6
7     return text.translate(translator)
8
9 def tokenize(text):
10    return # Your code here
11
12 def remove_stopwords(tokens):
13     stop_words = # Your code here
14

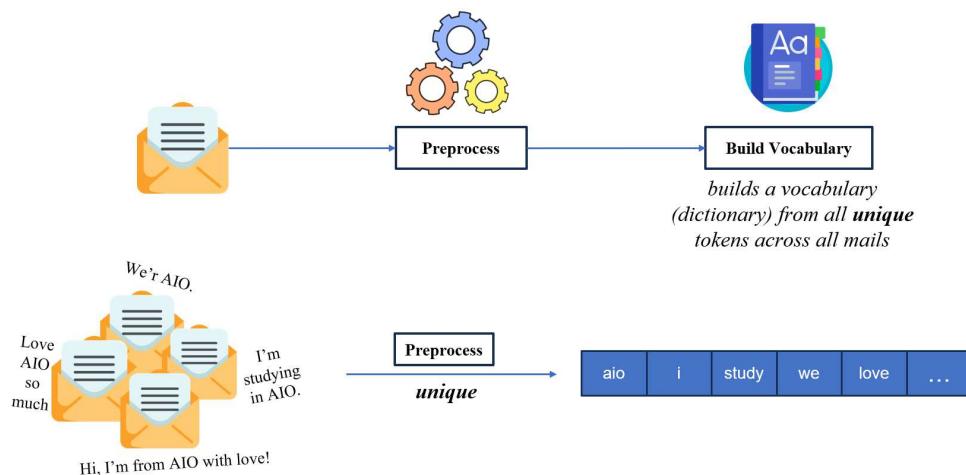
```

```

15     return [token for token in tokens if token not in stop_words]
16
17 def stemming(tokens):
18     stemmer = # Your code here
19
20     return [stemmer.stem(token) for token in tokens]
21
22 def preprocess_text(text):
23     # Your code here
24
25     return tokens
26
27 messages = [preprocess_text(message) for message in messages]

```

Tiếp theo, chúng ta cần tạo một bộ từ điển (Dictionary), chứa tất cả các từ có trong toàn bộ tin nhắn sau khi đã tiền xử lý (không tính trùng lặp).



Hình 10.7: Minh họa quá trình xây dựng từ điển từ các tin nhắn đã xử lý.

```

1 def create_dictionary(messages):
2     dictionary = []
3     for tokens in messages:
4         # Your code here
5     return dictionary

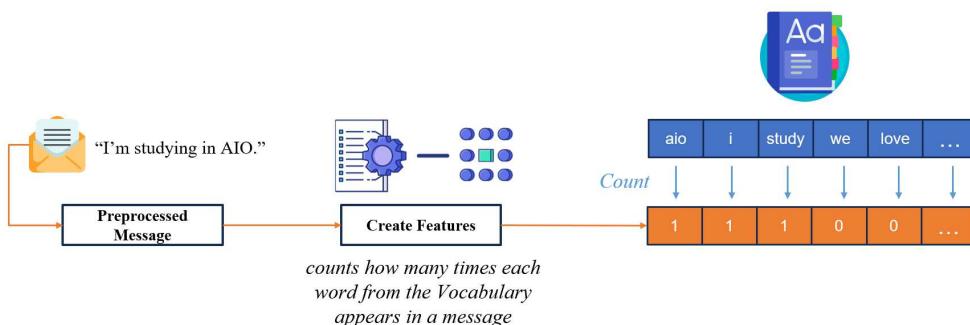
```

```

6
7 dictionary = create_dictionary(messages)

```

Kế đến, chúng ta cần tạo ra những đặc trưng đại diện cho thông tin (là các từ) của các tin nhắn. Một trong những cách đơn giản nhất là dựa vào tần suất xuất hiện của từ (Bag-of-Words). Với mỗi tin nhắn, vector đại diện sẽ có kích thước bằng với số lượng từ có trong từ điển.



Hình 10.8: Minh họa quá trình tạo vector đặc trưng dựa trên tần suất từ.

```

1 def create_features(tokens, dictionary):
2     features = np.zeros(len(dictionary))
3     for token in tokens:
4         if token in dictionary:
5             features[dictionary.index(token)] += 1
6     return features
7
8 X = np.array([create_features(tokens, dictionary) for tokens in
               messages])

```

Tiền xử lý dữ liệu nhãm: Đối với nhãm, chúng ta sẽ chuyển 2 nhãn *ham* và *spam* thành các con số 0 và 1 để máy tính có thể hiểu.

```

1 le = LabelEncoder()
2 y = le.fit_transform(labels)
3 print(f"Classes: {le.classes_}")

```

```
4 print(f"Encoded labels: {y}")
5 # >> Classes: ["ham" "spam"]
6 # >> Encoded labels: [0 0 1 ... 0 0 0]
```

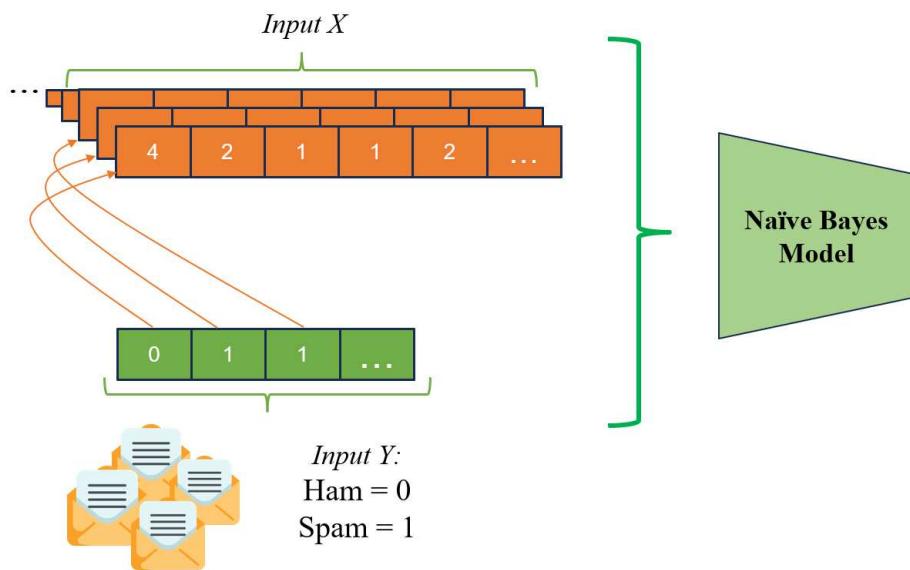
10.2.5 Chia bộ dữ liệu

Chúng ta sẽ chia bộ dữ liệu thành 3 phần: Train, Validation và Test theo tỉ lệ lần lượt là 70%, 20% và 10%.

```
1 VAL_SIZE = 0.2
2 TEST_SIZE = 0.125 # 0.1 / (1-0.2)
3 SEED = 0
4
5 X_train, X_val, y_train, y_val = train_test_split(X, y,
6                                     test_size=VAL_SIZE
7                                     ,
8                                     shuffle=True,
9                                     random_state=SEED)
10
11 X_train, X_test, y_train, y_test = train_test_split(X_train, y_train
12                                     ,
13                                     test_size=
14                                     TEST_SIZE,
15                                     shuffle=True,
16                                     random_state=
17                                     SEED)
```

10.2.6 Huấn luyện mô hình

Sau các bước trên, chúng ta chỉ cần truyền dữ liệu đã xử lý vào mô hình Gaussian Naive Bayes và tiến hành huấn luyện.



Hình 10.9: Đầu vào cho mô hình Naive Bayes.

Thực thi đoạn code sau để khởi tạo và huấn luyện mô hình:

```

1 model = GaussianNB()
2 print("Start training...")
3 model = # Your code here
4 print("Training completed!")
5
6 # >> Start training...
7 # >> Training completed!

```

10.2.7 Đánh giá mô hình

Sau khi huấn luyện, chúng ta đánh giá hiệu suất của mô hình trên tập Validation và Test bằng độ đo Accuracy.

```

1 y_val_pred = model.predict(X_val)
2 y_test_pred = model.predict(X_test)
3

```

```
4 val_accuracy = accuracy_score(y_val, y_val_pred)
5 test_accuracy = accuracy_score(y_test, y_test_pred)
6
7 print(f"Val accuracy: {val_accuracy:.4f}")
8 print(f"Test accuracy: {test_accuracy:.4f}")
9
10 # >> Val accuracy: 0.8816
11 # >> Test accuracy: 0.8602
```

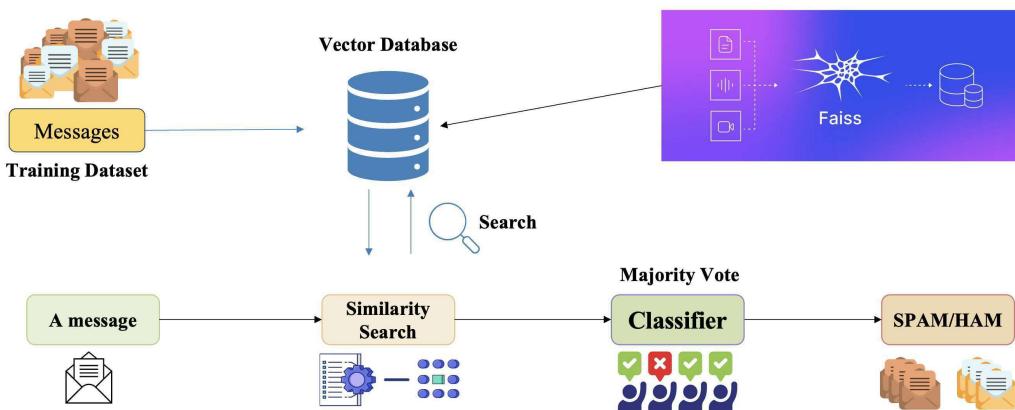
10.2.8 Thực hiện dự đoán

Cuối cùng, để sử dụng mô hình cho các tin nhắn mới, chúng ta sẽ phải thực hiện lại các công đoạn tiền xử lý, tạo đặc trưng và truyền vào mô hình để dự đoán.

```
1 def predict(text, model, dictionary, label_encoder):
2     processed_text = preprocess_text(text)
3     features = create_features(processed_text, dictionary)
4     features = np.array(features).reshape(1, -1)
5     prediction = model.predict(features)
6     prediction_cls = label_encoder.inverse_transform(prediction)[0]
7     return prediction_cls
8
9 test_input = "I am actually thinking a way of doing something useful"
10 prediction_cls = predict(test_input, model, dictionary, le)
11 print(f"Prediction: {prediction_cls}")
12
13 # >> Prediction: ham
```

10.3 Xây dựng chương trình phân loại tin nhắn Spam với Cơ sở dữ liệu Vector

Trong phần này, ta sẽ tiến hành xây dựng một chương trình phân loại tin nhắn spam (tin rác) và ham (tin thường) bằng một phương pháp hiện đại: sử dụng sentence embeddings và cơ sở dữ liệu vector. Thay vì dùng các mô hình học máy truyền thống như Naive Bayes, chúng ta sẽ chuyển đổi mỗi tin nhắn thành một vector số đại diện cho ngữ nghĩa của nó. Sau đó, việc phân loại một tin nhắn mới sẽ dựa trên việc tìm kiếm các tin nhắn tương tự nhất trong cơ sở dữ liệu đã có và lấy nhãn của chúng. Phương pháp này rất mạnh mẽ và hiệu quả, đặc biệt với sự hỗ trợ của thư viện FAISS từ Meta AI.



Hình 10.10: Chương trình phân loại tin nhắn Spam với Cơ sở dữ liệu Vector.

10.3.1 Thiết lập môi trường và Tải dữ liệu

Bước đầu tiên là chuẩn bị môi trường làm việc. Thao tác này bao gồm việc tải bộ dữ liệu từ Google Drive bằng lệnh `gdown` và import tất cả các thư viện cần thiết. Các thư viện chính bao gồm:

- **pandas, numpy:** Dành cho việc xử lý và thao tác dữ liệu.
- **torch, transformers:** Để tải và sử dụng mô hình embedding ngôn ngữ từ Hugging Face.

- **faiss:** Thư viện cho việc tìm kiếm tương đồng trên vector một cách hiệu quả.
- **sklearn:** Dùng để chia dữ liệu và mã hóa nhãn.
- **tqdm:** Để hiển thị thanh tiến trình (progress bar) khi xử lý các tác vụ tốn thời gian.



Hình 10.11: Facebook AI Similarity Search (FAISS)

```

1 !pip install -qq faiss-cpu
2 !pip install -qq transformers
3 !pip install -qq pandas
4 !pip install -qq numpy
5 !pip install -qq scikit-learn
6 !pip install -qq tqdm
7
8 # Tải dữ liệu từ Google Drive
9 # https://drive.google.com/file/d/1N7rk-kfnDFIGMeXOROVThjKh71gcgx-7R/
   view?usp=sharing
10 !gdown --id 1N7rk-kfnDFIGMeXOROVThjKh71gcgx-7R
11
12 # 1. Import các thư viện cần thiết
13 import pandas as pd
14 import numpy as np
15 import torch
16 import torch.nn.functional as F
17 import faiss
18 from transformers import AutoTokenizer, AutoModel
19 from sklearn.model_selection import train_test_split
20 from sklearn.preprocessing import LabelEncoder
21 from tqdm import tqdm

```

10.3.2 Đọc và Chuẩn bị Dữ liệu

Sau khi tải file về môi trường, chúng ta sử dụng thư viện pandas để đọc file .csv vào một cấu trúc dữ liệu gọi là DataFrame. Từ DataFrame này, chúng ta sẽ tách riêng cột chứa nội dung tin nhắn ("Message") và cột chứa nhãn phân loại ("Category") ra thành hai danh sách (list) riêng biệt để thuận tiện cho các bước xử lý và vector hóa sau này.

```

1 # 2. Đọc bộ dữ liệu
2 DATASET_PATH = "/content/2cls_spam_text_cls.csv"
3 df = pd.read_csv(DATASET_PATH)
4
5 # Tách tin nhắn và nhãn vào các list
6 messages = df["Message"].values.tolist()
7 labels = df["Category"].values.tolist()
```

10.3.3 Chuẩn bị Mô hình Embedding

Đây là bước cốt lõi để chuyển đổi văn bản thành các vector (embedding). Chúng ta tải mô hình ngôn ngữ intfloat/multilingual-e5-base từ Hugging Face. Đây là một mô hình mạnh mẽ, được huấn luyện để tạo ra các vector (embedding) đại diện ngữ nghĩa cho văn bản. Đoạn code cũng tự động phát hiện và ưu tiên sử dụng GPU (cuda) nếu có để tăng tốc tính toán. Hàm average_pool được định nghĩa để tính toán vector đại diện chung cho cả câu từ các vector output của mô hình transformer, một kỹ thuật phổ biến để có được sentence embedding chất lượng.

```

1 # 3.1. Load mô hình embedding
2 MODEL_NAME = "intfloat/multilingual-e5-base"
3 tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
4 model = AutoModel.from_pretrained(MODEL_NAME)
5
6 # Set device
7 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
8 model = model.to(device)
9 model.eval()
10
```

```

11 # Hàm để trích xuất embedding từ output của model
12 def average_pool(last_hidden_states, attention_mask):
13     last_hidden = last_hidden_states.masked_fill(
14         ~attention_mask[..., None].bool(), 0.0
15     )
16     return last_hidden.sum(dim=1) / attention_mask.sum(dim=1)[..., None]

```

10.3.4 Vector hóa Dữ liệu và Tạo Metadata

Ở bước này, chúng ta định nghĩa hàm `get_embeddings` để thực hiện quá trình vector hóa toàn bộ dữ liệu văn bản. Hàm này xử lý các tin nhắn theo từng lô (batch) để tối ưu hiệu năng. Một kỹ thuật quan trọng được áp dụng là thêm tiền tố "passage: " vào mỗi tin nhắn, giúp cải thiện chất lượng embedding cho mô hình E5. Sau khi nhận được các vector, chúng ta chuẩn hóa (normalize) chúng.

Song song đó, các nhãn dạng chữ ('ham'/'spam') được mã hóa thành số ('0'/'1') bằng `LabelEncoder`. Một bộ metadata (dữ liệu mô tả) cũng được tạo ra để lưu trữ thông tin gốc của mỗi tin nhắn, giúp liên kết các vector trở lại với dữ liệu ban đầu một cách dễ dàng.

```

1 # 3.2. Tạo sentence embeddings
2 def get_embeddings(texts, model, tokenizer, device, batch_size=32):
3     """Tạo embeddings cho một danh sách các văn bản"""
4     embeddings = []
5     for i in tqdm(range(0, len(texts), batch_size), desc="Generating
6         embeddings"):
6         batch_texts = texts[i:i+batch_size]
7         batch_texts_with_prefix = [f"passage: {text}" for text in
8             batch_texts]
9
9         batch_dict = tokenizer(batch_texts_with_prefix, max_length=
10             512,
11             padding=True, truncation=True,
12             return_tensors="pt")
13         batch_dict = {k: v.to(device) for k, v in batch_dict.items()
14             }

```

```

13     with torch.no_grad():
14         outputs = model(**batch_dict)
15         batch_embeddings = average_pool(outputs.
16                                         last_hidden_state, batch_dict["  

17                                         attention_mask"])
18         batch_embeddings = F.normalize(batch_embeddings, p=2,
19                                         dim=1)
20         embeddings.append(batch_embeddings.cpu().numpy())
21
22     # Chuẩn bị nhãn
23     le = LabelEncoder()
24     y = le.fit_transform(labels)
25
26     # Tạo embeddings cho tất cả tin nhắn
27     X_embeddings = get_embeddings(messages, model, tokenizer, device)
28
29     # Tạo metadata cho mỗi tài liệu
30     metadata = [{"index": i, "message": message, "label": label, "  

31                 "label_encoded": y[i]}  

32                 for i, (message, label) in enumerate(zip(messages,  

33                                               labels))]
```

10.3.5 Xây dựng Cơ sở dữ liệu Vector và Chia Dữ liệu

Tại đây, chúng ta chia bộ dữ liệu đã được vector hóa thành hai tập: 90% cho huấn luyện (training) và 10% cho kiểm thử (testing) bằng cách sử dụng các chỉ số (indices). Việc chia theo `stratify=y` đảm bảo tỉ lệ spam/ham trong hai tập là tương đương nhau, tránh sai lệch.

Sau đó, chúng ta tạo một cơ sở dữ liệu vector hiệu năng cao bằng thư viện FAISS. Chúng ta khởi tạo một chỉ mục `IndexFlatIP`, sử dụng phép toán Tích Vô hướng (Inner Product) để đo độ tương đồng. Chỉ các vector của tập huấn luyện được thêm vào cơ sở dữ liệu này để phục vụ cho việc tìm kiếm và phân loại sau này.

```

1 # 3.3. Tạo FAISS index và chia dữ liệu
2 TEST_SIZE = 0.1
3 SEED = 42
```

```

4
5 train_indices, test_indices = train_test_split(
6     range(len(messages)), test_size=TEST_SIZE, stratify=y,
7         random_state=SEED
8 )
9 # Tách embeddings và metadata theo chỉ số đã chia
10 X_train_emb = X_embeddings[train_indices]
11 X_test_emb = X_embeddings[test_indices]
12 train_metadata = [metadata[i] for i in train_indices]
13 test_metadata = [metadata[i] for i in test_indices]
14
15 # Tạo FAISS index
16 embedding_dim = X_train_emb.shape[1]
17 index = faiss.IndexFlatIP(embedding_dim)
18 index.add(X_train_emb.astype("float32"))

```

10.3.6 Xây dựng Logic Phân loại và Đánh giá

Đoạn code này định nghĩa hai hàm cốt lõi cho việc phân loại và đánh giá:

1. **classify_with_knn:** Đây là hàm thực hiện việc phân loại. Nó nhận một văn bản mới (query), tạo embedding cho nó (với tiền tố "query:"), sau đó dùng chỉ mục FAISS để tìm kiếm k tin nhắn giống nhất trong tập huấn luyện. Nhãn của tin nhắn mới được quyết định dựa trên nhãn chiếm đa số (majority vote) của k "hàng xóm" này.
2. **evaluate_knn_accuracy:** Hàm này được dùng để đánh giá độ chính xác của phương pháp trên tập test. Nó duyệt qua từng mẫu trong tập test, phân loại chúng, so sánh với nhãn thật và tính toán độ chính xác tổng thể.

```

1 # 4. Triển khai phân loại với embedding similarity
2 def classify_with_knn(query_text, model, tokenizer, device, index,
3                         train_metadata, k=1):
4     """Classify text using k-nearest neighbors with embeddings"""
5
6     # Get query embedding
7     query_with_prefix = f"query: {query_text}"

```

```
7     batch_dict = tokenizer([query_with_prefix],
8                           max_length=512,
9                           padding=True,
10                          truncation=True,
11                          return_tensors="pt")
12
13    batch_dict = {k: v.to(device) for k, v in batch_dict.items()}
14
15    with torch.no_grad():
16        outputs = model(**batch_dict)
17        query_embedding = average_pool(outputs.last_hidden_state,
18                                        batch_dict["attention_mask"])
18        query_embedding = F.normalize(query_embedding, p=2, dim=1)
19        query_embedding = query_embedding.cpu().numpy().astype("float32")
20
21    # Search in FAISS index
22    scores, indices = index.search(query_embedding, k)
23
24    # Get predictions from top-k neighbors
25    predictions = []
26    neighbor_info = []
27
28    for i in range(k):
29        neighbor_idx = indices[0][i]
30        neighbor_score = scores[0][i]
31        neighbor_label = train_metadata[neighbor_idx]["label"]
32        neighbor_message = train_metadata[neighbor_idx]["message"]
33
34        predictions.append(neighbor_label)
35        neighbor_info.append({
36            "score": float(neighbor_score),
37            "label": neighbor_label,
38            "message": neighbor_message[:100] + "..." if len(
39                                neighbor_message) > 100 else
40                                neighbor_message
41        })
42
43    # Majority vote for final prediction
44    unique_labels, counts = np.unique(predictions, return_counts=
45                                      True)
46    final_prediction = unique_labels[np.argmax(counts)]
47
48    return final_prediction, neighbor_info
```

```
46
47 def evaluate_knn_accuracy(test_embeddings, test_labels,
48                             test_metadata, index,
49                             train_metadata, k_values=[1, 3, 5
50 ]):
51     """Evaluate accuracy for different k values using precomputed
52         embeddings"""
53     results = []
54     all_errors = []
55
56     for k in k_values:
57         correct = 0
58         total = len(test_embeddings)
59         errors = []
60
61         for i in tqdm(range(total), desc=f"Evaluating k={k}"):
62             query_embedding = test_embeddings[i:i+1].astype("float32"
63                                                 )
64             true_label = test_metadata[i] ["label"]
65             true_message = test_metadata[i] ["message"]
66
67             # Search in FAISS index
68             scores, indices = index.search(query_embedding, k)
69
70             # Get predictions from top-k neighbors
71             predictions = []
72             neighbor_details = []
73             for j in range(k):
74                 neighbor_idx = indices[0] [j]
75                 neighbor_label = train_metadata[neighbor_idx] ["label"
76                                         ]
77                 neighbor_message = train_metadata[neighbor_idx] [
78                     "message"]
79                 neighbor_score = float(scores[0] [j])
80
81                 predictions.append(neighbor_label)
82                 neighbor_details.append({
83                     "label": neighbor_label,
84                     "message": neighbor_message,
85                     "score": neighbor_score
86                 })
87
88             # Majority vote
89             unique_labels, counts = np.unique(predictions,
```

```

83         return_counts=True)
84     predicted_label = unique_labels[np.argmax(counts)]
85
86     if predicted_label == true_label:
87         correct += 1
88     else:
89         # Collect error information
90         error_info = {
91             "index": i,
92             "original_index": test_metadata[i]["index"],
93             "message": true_message,
94             "true_label": true_label,
95             "predicted_label": predicted_label,
96             "neighbors": neighbor_details,
97             "label_distribution": {label: int(count) for
98                 label, count in zip(unique_labels,
99                 counts)}
100        }
101        errors.append(error_info)
102
103    accuracy = correct / total
104    error_count = total - correct
105
106    results[k] = accuracy
107    all_errors[k] = errors
108
109    print(f"Accuracy with k={k}: {accuracy:.4f}")
110    print(f"Number of errors with k={k}: {error_count}/{total} ("
111          f"({error_count/total}*100:.2f)%)")
112
113 return results, all_errors

```

10.3.7 Đánh giá Mô hình trên Tập Test

Bây giờ, chúng ta áp dụng hàm `evaluate_knn_accuracy` đã xây dựng lên tập dữ liệu test. Việc này nhằm mục đích đo lường hiệu năng thực tế của phương pháp phân loại trên những dữ liệu mà mô hình chưa từng thấy. Chúng ta thử nghiệm với các giá trị k khác nhau (1, 3, và 5) để quan sát xem việc thay đổi số lượng "hàng xóm" ảnh hưởng đến độ chính xác như thế nào. Kết quả cuối cùng sẽ được in ra màn hình và một file JSON chi tiết về các trường hợp dự đoán sai cũng được lưu lại để có thể phân tích sâu hơn.

```
1 # 5. Dánh giá accuracy trên test set
2 %%time
3 print("Evaluating accuracy on test set...")
4 accuracy_results, error_results = evaluate_knn_accuracy(
5     X_test_emb,
6     y_test,
7     test_metadata,
8     index,
9     train_metadata,
10    k_values=[1, 3, 5]
11 )
12
13 # Hiển thị kết quả
14 print("\n" + "="*50)
15 print("ACCURACY RESULTS")
16 print("="*50)
17 for k, accuracy in accuracy_results.items():
18     print(f"Top-{k} accuracy: {accuracy:.4f} ({accuracy*100:.2f}%)")
19 print("="*50)
20
21 # Lưu phân tích lỗi ra file
22 import json
23 from datetime import datetime
24
25 error_analysis = {
26     "timestamp": datetime.now().isoformat(),
27     "model": MODEL_NAME,
28     "test_size": len(X_test_emb),
29     "accuracy_results": accuracy_results,
30     "errors_by_k": {}
31 }
32
33 for k, errors in error_results.items():
34     error_analysis["errors_by_k"][f"k_{k}"] = {
35         "total_errors": len(errors),
36         "error_rate": len(errors) / len(X_test_emb),
37         "errors": errors
38     }
39
40 # Lưu JSON file ghi lỗi
41 output_file = "error_analysis.json"
42 with open(output_file, "w", encoding="utf-8") as f:
43     json.dump(error_analysis, f, ensure_ascii=False, indent=2)
44
```

```

45 print(f"\n***Error analysis saved to: {output_file}***")
46 print()
47 print(f"***Summary:")
48 for k, errors in error_results.items():
49     print(f"    k={k}: {len(errors)} errors out of {len(X_test_emb)}"
          samples")

```

10.3.8 Xây dựng Pipeline Phân loại Hoàn chỉnh

Để thuận tiện cho việc sử dụng và tái sử dụng, chúng ta xây dựng hàm `spam_classifier_pipeline`. Hàm này đóng gói toàn bộ quy trình, từ việc nhận một văn bản đầu vào của người dùng, tạo embedding, tìm kiếm trong FAISS, cho đến việc đưa ra dự đoán cuối cùng và hiển thị kết quả. Pipeline này không chỉ trả về nhãn dự đoán mà còn cung cấp thông tin về các "hàng xóm" gần nhất đã được dùng để đưa ra quyết định đó, giúp tăng tính minh bạch và dễ hiểu cho người dùng.

```

1 # 6. Pipeline classification cho user input
2 def spam_classifier_pipeline(user_input, k=3):
3     """
4         Complete pipeline for spam classification
5
6     Args:
7         user_input (str): Text to classify
8         k (int): Number of nearest neighbors to consider
9
10    Returns:
11        dict: Classification results with details
12    """
13
14    print()
15    print(f"***Classifying: '{user_input}'")
16    print()
17    print(f"***Using top-{k} nearest neighbors")
18    print()
19
20    # Get prediction and neighbors
21    prediction, neighbors = classify_with_knn(

```

```

22         user_input, model, tokenizer, device, index, train_metadata,
23                         k=k
24     )
25
26     # Display results
27     print(f"***Prediction: {prediction.upper()}")
28     print()
29
30     print("***Top neighbors:")
31     for i, neighbor in enumerate(neighbors, 1):
32         print(f"{i}. Label: {neighbor['label']} | Score: {neighbor['"
33                                     score"]:.4f}")
34         print(f"    Message: {neighbor['message']}")
35         print()
36
37     # Count label distribution
38     labels = [n["label"] for n in neighbors]
39     label_counts = {label: labels.count(label) for label in set(
40                           labels)}
41
42
43     return {
44         "prediction": prediction,
45         "neighbors": neighbors,
46         "label_distribution": label_counts
47     }

```

10.3.9 Kiểm thử Pipeline

Đây là bước cuối cùng để xác nhận rằng pipeline của chúng ta hoạt động chính xác. Chúng ta kiểm tra nó với một danh sách các ví dụ có sẵn, bao gồm cả tin nhắn thường và tin nhắn spam điển hình. Ngoài ra, một khối code "tương tác" được cung cấp, cho phép người dùng có thể dễ dàng thay đổi nội dung tin nhắn trong biến `user_text` và giá trị `k` trong biến `k_value` để tự mình thử nghiệm với các trường hợp khác nhau.

```

1 # 7. Test pipeline với các ví dụ khác nhau
2 test_examples = [
3     "I am actually thinking a way of doing something useful",
4     "FREE!! Click here to win \$1000 NOW! Limited time offer!",
5 ]

```

```
6 for i, example in enumerate(test_examples, 1):
7     print(f"\n--- Example {i}: '{example}' ---")
8     result = spam_classifier_pipeline(example, k=3)
9
10 # Interactive testing - người dùng có thể thay đổi text và k value
11 print("\n--- Interactive Testing ---")
12 user_text = "Win a free iPhone! Click here now!"
13 k_value = 5
14 result = spam_classifier_pipeline(user_text, k=k_value)
```

10.4 Câu hỏi trắc nghiệm

1. Khi sử dụng `faiss.IndexFlatIP` với các embedding đã được chuẩn hóa L2-normalized, phép đo độ tương đồng "Inner Product" thực chất tương đương với phép đo nào sau đây?
 - (A) Khoảng cách Euclidean.
 - (B) Khoảng cách Manhattan.
 - (C) Độ tương đồng Cosine.
 - (D) Khoảng cách Chebyshev.
2. Mục đích chính của việc thêm tiền tố "`passage:`" cho tin nhắn trong `get_embeddings` và "`query:`" cho đầu vào trong `classify_with_knn` là gì?
 - (A) Để tăng độ dài của chuỗi đầu vào, giúp mô hình học tốt hơn.
 - (B) Để hướng dẫn mô hình E5 tạo ra embedding phù hợp với ngữ cảnh tìm kiếm và truy vấn.
 - (C) Để phân biệt rõ ràng giữa tin nhắn gốc và tin nhắn được tạo ra trong quá trình xử lý.
 - (D) Để giảm thiểu số lượng token và tăng tốc độ xử lý của tokenizer.
3. Tại sao `stratify=y` lại quan trọng khi chia dữ liệu với `train_test_split` trong dự án phân loại Spam này?
 - (A) Đảm bảo dữ liệu được chia đều theo thứ tự alphabet của nhãn.
 - (B) Giúp tăng tốc quá trình chia dữ liệu.
 - (C) Để chia ngẫu nhiên dữ liệu một cách tối ưu.
 - (D) Để duy trì tỉ lệ các lớp (spam/ham) giống nhau giữa tập huấn luyện và tập kiểm thử.
4. Nếu một tin nhắn vượt quá `max_length=512` trong quá trình token hóa và được `truncation=True`, điều gì có thể xảy ra?
 - (A) Mô hình sẽ tự động điều chỉnh để xử lý toàn bộ tin nhắn.
 - (B) Tin nhắn sẽ bị loại bỏ hoàn toàn khỏi bộ dữ liệu.

- (C) Thông tin ngữ cảnh quan trọng ở cuối tin nhắn có thể bị mất, ảnh hưởng đến chất lượng embedding.
- (D) Mô hình sẽ báo lỗi vì không thể xử lý chuỗi quá dài.
5. Hàm `average_pool` sau khi lấy `last_hidden_states` sử dụng `masked_fill` với `attention_mask[..., None].bool()`. Mục đích của việc này là gì?
- (A) Để đảm bảo rằng chỉ các token quan trọng nhất được giữ lại.
- (B) Để loại bỏ các token không có ý nghĩa trong câu.
- (C) Để thêm các giá trị ngẫu nhiên vào các vị trí bị mask.
- (D) Để gán giá trị 0.0 cho các vị trí bị padding (mask) trước khi tính tổng, tránh ảnh hưởng đến average.
6. Việc tạo ra `metadata` (bao gồm `index`, `message`, `label`, `label_encoded`) trong dự án có ý nghĩa quan trọng nhất nào đối với khả năng phân tích và debug?
- (A) Giúp giảm kích thước bộ nhớ lưu trữ embeddings.
- (B) Tăng tốc độ tìm kiếm trong FAISS.
- (C) Cho phép dễ dàng liên kết lại các embedding với tin nhắn và nhãn gốc, đặc biệt khi phân tích lỗi.
- (D) Là yêu cầu bắt buộc của thư viện FAISS.
7. Tại sao việc tăng giá trị `k` (số lượng hàng xóm) trong K-NN từ 1 lên 3 hoặc 5 đôi khi có thể cải thiện độ chính xác của phân loại?
- (A) Vì mô hình có nhiều dữ liệu hơn để học.
- (B) Vì việc lấy ý kiến đa số từ nhiều hàng xóm giúp giảm ảnh hưởng của nhiễu hoặc sai lệch từ một vài điểm dữ liệu đơn lẻ.
- (C) Vì nó làm tăng tốc độ tìm kiếm trong cơ sở dữ liệu vector.
- (D) Vì nó đơn giản hóa ranh giới quyết định giữa các lớp.
8. Trong hàm `classify_with_knn`, tại sao `query_embedding` sau khi được tạo phải được chuyển đổi thành `astype("float32")` trước khi gọi `index.search` của FAISS?

- (A) `float32` yêu cầu ít bộ nhớ hơn, giúp tiết kiệm tài nguyên.
- (B) FAISS yêu cầu kiểu dữ liệu `float32` cho các vector đầu vào để hoạt động hiệu quả.
- (C) Để giảm độ chính xác của embedding, giúp phân loại nhanh hơn.
- (D) Đây là một bước tùy chọn, không ảnh hưởng đến kết quả.
9. Giả sử một tin nhắn Spam thực tế được phân loại nhầm thành "Ham".
Dựa vào `error_analysis.json` được tạo ra, thông tin nào sau đây sẽ hữu ích nhất để hiểu nguyên nhân gây ra lỗi này?
- (A) `timestamp` và `total_errors`.
- (B) `model` và `test_size`.
- (C) `accuracy_results` cho các giá trị k khác nhau.
- (D) `neighbors` và `label_distribution` của các hàng xóm gần nhất.
10. Mặc dù mạnh mẽ, một nhược điểm tiềm tàng của hệ thống phân loại Spam dựa trên tìm kiếm tương đồng vector (k-NN) là gì, đặc biệt khi dữ liệu huấn luyện rất lớn?
- (A) Khó khăn trong việc hiểu ngữ nghĩa của tin nhắn.
- (B) Không thể phát hiện tin nhắn spam mới.
- (C) Chi phí tính toán và bộ nhớ cao cho việc tìm kiếm k-NN trong cơ sở dữ liệu lớn (mặc dù FAISS giúp giảm thiểu).
- (D) Không thể áp dụng cho các ngôn ngữ khác ngoài tiếng Anh.

1. **Hint:** Các file code gợi ý và dữ liệu được lưu trong thư mục có thể được tải [tại đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. Kiến thức cần cho project:

Để hoàn thành và hiểu rõ phần project một cách hiệu quả nhất, các học viên cần nắm rõ các kiến thức được trang bị

- Kiến thức lập trình Python và xử lý chuỗi
- Hiểu về cơ sở dữ liệu vector (vector database) ở phần project 2.1
- Cách thao tác và xử lý với numpy, xác suất thống kê, đại số tuyến tính như độ tương đồng cosine, ...

4. Đề xuất phương pháp cải tiến project:

Project tiếp cận bài toán phân loại email theo hướng sử dụng các phương pháp đơn giản để biểu diễn vector và naive bayes hoặc sử dụng vector database. Để tiếp tục cải tiến, tối ưu hệ thống; nhóm tác biên soạn gợi ý một số phương pháp như sau:

- Tiềm xử lý nâng cao: Sử dụng các kỹ thuật tiềm xử lý nâng cao hơn như loại bỏ từ ngữ ít xuất hiện (rare words), chuẩn hóa cú pháp (normalization), phát hiện và xử lý các từ phủ định (negation handling) để cải thiện độ chính xác phân loại
- Kỹ thuật biểu diễn văn bản khác: Thử nghiệm phương pháp TF-IDF thay cho Bag of Words để giảm trọng số các từ phổ biến và nhấn mạnh các từ quan trọng trong ngữ cảnh cảm xúc
- Tăng cường dữ liệu (Data Augmentation): Áp dụng kỹ thuật tăng cường dữ liệu bằng cách thay thế từ đồng nghĩa, đảo vị trí câu, hoặc sinh dữ liệu mới để làm phong phú tập huấn luyện
- Đánh giá mô hình đa chỉ số: Áp dụng thêm các chỉ số đánh giá như F1-score, ROC-AUC, confusion matrix để có cái nhìn toàn diện hơn về hiệu quả mô hình, thay vì chỉ dựa vào accuracy

- Tri thức cho truy vấn: Bổ sung thêm những dạng tri thức biểu diễn khác cho văn bản hoặc kết hợp thêm các phương pháp tìm kiếm khác nhau như Bm25 kết hợp cosine similarity,...

Trên đây là một số gợi ý, giúp học viện có thể cải tiến thêm hệ thống để đạt hiểu quả tốt hơn về cả tốc độ xử lý và độ chính xác.

5. Rubric:

Mục	Kiến Thức	Đánh Giá
II.	<ul style="list-style-type: none"> - Lý thuyết về bài toán Text Classification trong Machine Learning. - Lý thuyết về mô hình Naive Bayes. - Kiến thức về lập trình python cơ bản và cách sử dụng các thư viện liên quan đến machine learning cơ bản như: pandas, scikit-learn. - Các bước tiền xử lý dữ liệu văn bản trong Natural Language Processing. - Quy trình cơ bản của một chương trình huấn luyện mô hình Machine Learning. 	<ul style="list-style-type: none"> - Nắm được khái niệm về bài toán Text Classification trong Machine Learning. - Hiểu được cách mô hình Naive Bayes có thể giải quyết bài toán phân loại văn bản. - Có khả năng sử dụng Python để xây dựng một mô hình machine learning cơ bản nhằm giải quyết bài toán phân loại. - Nắm được các kỹ thuật tiền xử lý văn bản cơ bản trong Natural Language Processing, phục vụ cho bài toán phân loại văn bản. - Hiểu được các bước làm cơ bản để huấn luyện một mô hình machine learning.
III.	<ul style="list-style-type: none"> - Áp dụng vector database để giải quyết bài toán trên. 	<ul style="list-style-type: none"> - Thực thi thông qua thư viện FAISS cho vector database.

Phần III

Module 3: Máy học cơ bản

Chương 11

Trực quan hóa và phân tích dữ liệu dùng Pandas

11.1 Giới thiệu

Pandas là một thư viện trong Python với ưu điểm là nhanh, mạnh, linh động, dễ sử dụng, mã nguồn mở, công cụ dùng để phân tích và thao tác dữ liệu. Pandas được xây dựng trên thư viện NumPy và có nhiều hàm hỗ trợ làm sạch, phân tích, và mô hình hoá dữ liệu, có thể giúp chúng ta tìm ra những đặc trưng của các tập dữ liệu. Pandas rất hiệu quả khi sử dụng trên dữ liệu bảng, như SQL table hoặc Excel spreadsheets.



Hình 11.1: Thư viện pandas.

Đặc điểm của Pandas:

- Hỗ trợ thao tác dữ liệu nhanh chóng và hiệu quả
- Làm việc dễ dàng với dữ liệu dạng bảng (giống Excel, SQL)
- Xử lý dữ liệu thiếu linh hoạt (fillna, dropna,...)
- Tích hợp chặt chẽ với NumPy và các thư viện phân tích dữ liệu khác
- Hỗ trợ nhiều định dạng dữ liệu: CSV, Excel, JSON, SQL, clipboard,...
- Tốc độ cao, tối ưu cho dữ liệu lớn
- Cung cấp nhiều hàm thao tác dữ liệu: lọc, nhóm, thống kê, nối,...

Cấu trúc dữ liệu chính của pandas:

- Series: Mảng một chiều có nhãn (index), giống một cột trong bảng dữ liệu
- DataFrame: Bảng hai chiều gồm nhiều dòng và cột, mỗi cột là một Series, là cấu trúc dữ liệu quan trọng và được dùng nhiều nhất.

Khởi tạo cấu trúc dữ liệu trong pandas

```

1 import pandas as pd
2
3 # Series
4 s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
5
6 # Dataframe
7 data = {
8     "Tên": ["An", "Bình", "Chi"],
9     "Tuổi": [20, 21, 22]
10 }
11 df = pd.DataFrame(data)

```

Ứng dụng của pandas:

- Khoa học dữ liệu (Data Science) → Tiền xử lý dữ liệu, khám phá dữ liệu (EDA), trích xuất đặc trưng
- Machine Learning (ML) → Chuẩn hóa, biến đổi dữ liệu đầu vào cho mô hình học máy
- Phân tích kinh doanh và tài chính → Phân tích doanh thu, chi phí, lợi nhuận, phân khúc khách hàng
- Xử lý dữ liệu lớn / dữ liệu log → Làm sạch, lọc, chuyển đổi dữ liệu từ hệ thống lớn
- Trực quan hóa dữ liệu → Kết hợp với matplotlib, seaborn để vẽ biểu đồ, biểu diễn số liệu
- Tự động hóa xử lý file Excel / CSV → Đọc, cập nhật, ghi báo cáo định kỳ, tự động hóa công việc văn phòng.

Trong phần này, chúng ta sẽ sử dụng pandas để thực hiện một số kỹ thuật phân tích trên hai bộ dữ liệu về text và time-series. Các câu bài tập được chia thành các bước thực hiện trong bài toán, dựa trên nội dung các bước, hình minh họa kết quả và mô tả code để hoàn thiện lần lượt các phần code nếu còn thiếu.

11.2 Tìm hiểu về Pandas

11.2.1 Cài đặt và Import thư viện

Pandas là thư viện độc lập nên cần được cài đặt trước khi sử dụng trong môi trường Python. Đây là một thư viện nằm ngoài thư viện tiêu chuẩn của Python, vì vậy bạn cần cài thủ công thông qua pip.

Cài đặt bằng pip:

```
1 pip install pandas
```

Sau khi cài đặt, bạn cần import thư viện vào chương trình bằng cú pháp sau:

```
1 import pandas as pd
```

Theo quy ước phổ biến trong cộng đồng Python, thư viện pandas thường được viết tắt là pd. Việc viết tắt này giúp giảm độ dài cú pháp khi gọi các hàm như:

```
1 pd.read_csv()  
2 pd.DataFrame()
```

Để kiểm tra phiên bản Pandas hiện tại đang sử dụng, bạn có thể dùng câu lệnh sau:

```
1 print(pd.__version__)
```

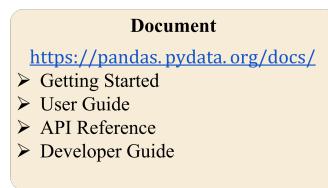
Tài liệu chính thức: Pandas được duy trì và cập nhật thường xuyên. Tài liệu đầy đủ, rõ ràng và rất chi tiết có thể được truy cập tại: <https://pandas.pydata.org/docs/>

```

!pip install pandas
import pandas as pd
pd.__version__
>>> 2.3.1

```

Hình 11.2: Cài đặt và import pandas.



Hình 11.3: Tài liệu chính thức pandas.

11.2.2 Làm việc với dữ liệu cơ bản

Đọc và ghi dữ liệu

Một trong những điểm mạnh của Pandas là khả năng tương tác với nhiều định dạng dữ liệu phổ biến. Việc đọc (import) và ghi (export) dữ liệu rất dễ dàng và linh hoạt, hỗ trợ các định dạng như CSV, JSON, Excel, SQL, clipboard,...

Đọc dữ liệu từ tệp CSV – định dạng phổ biến nhất trong phân tích dữ liệu:

```

1 df = pd.read_csv('data.csv')

```

Ngoài ra, có thể tùy chỉnh các tham số như:

- **sep**: ký tự phân tách, mặc định là dấu phẩy.
- **encoding**: mã hóa tệp (ví dụ utf-8, utf-16).
- **header, names**: kiểm soát dòng tiêu đề.

Đọc dữ liệu từ tệp JSON:

```

1 df = pd.read_json('data.json')

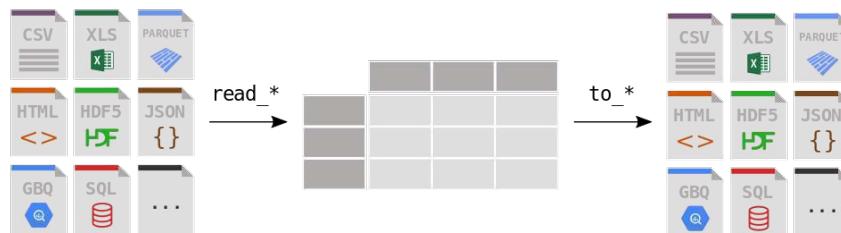
```

Ghi dữ liệu ra file:

```
1 df.to_csv('output.csv', index=False)
2 df.to_json('output.json', orient='records')
```

Ghi chú: `index=False` giúp bỏ cột chỉ số mặc định khi ghi ra file. Tuỳ chọn `orient` trong JSON điều khiển cấu trúc dữ liệu đầu ra.

Pandas cũng hỗ trợ các định dạng khác như Excel (`read_excel()`, `to_excel()`), SQL (`read_sql()`, `to_sql()`) và clipboard (`read_clipboard()`).



Hình 11.4: Đọc dữ liệu Pandas.

Cấu trúc dữ liệu chính

Pandas xây dựng trên hai cấu trúc dữ liệu cốt lõi: `Series` và `DataFrame`. Đây là nền tảng của mọi thao tác phân tích trong Pandas.

Trong Pandas, `Series` là một cấu trúc dữ liệu một chiều, hoạt động như một mảng (array) có gán nhãn. Mỗi phần tử trong Series bao gồm một giá trị và một chỉ mục (index). Mặc định, nếu không chỉ định index, Pandas sẽ tự động tạo chỉ số dạng số nguyên bắt đầu từ 0.

Ví dụ tạo một Series với index mặc định:

```
1 s = pd.Series([3, -5, 7, 4])
```

Trong trường hợp này, các phần tử sẽ có index là 0, 1, 2, 3 tương ứng với các giá trị. Đây là cách đơn giản nhất để khởi tạo một Series, tương tự như một mảng có đánh số dòng.

Index	Data
0	3
1	-5
2	7
3	4

Hình 11.5: Minh họa cấu trúc Series mặc định.

Tuy nhiên, Pandas cho phép tùy chỉnh index bằng cách truyền thêm tham số `index=[...]` khi khởi tạo Series. Điều này hữu ích khi muốn gán nhãn rõ ràng cho dữ liệu thay vì chỉ số số nguyên mặc định.

Ví dụ tạo Series với index được gán thủ công:

```
1 s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

Việc gán index tùy chỉnh cho phép truy xuất dữ liệu dễ dàng hơn bằng cách dùng nhãn thay vì số, ví dụ: `s['b']` sẽ trả về giá trị -5.

Index	Data
a	3
b	-5
c	7
d	4

Hình 11.6: Minh họa cấu trúc Series khi tùy chỉnh index.

DataFrame là cấu trúc bảng hai chiều, gồm nhiều Series xếp thành cột, với nhãn dòng (index) và nhãn cột (column).

```

1 data = {
2     'Name': ['Anh', 'Thai', 'Hoa'],
3     'Country': ['Brazil', 'Vietnam', 'Vietnam']
4 }
5
6 df = pd.DataFrame(data, columns=['Name', 'Country'])
7 df

```

Mỗi cột trong DataFrame có thể mang kiểu dữ liệu khác nhau. Một số kiểu dữ liệu phổ biến gồm:

- `int64`: số nguyên
- `float64`: số thực
- `object`: chuỗi ký tự (str)
- `bool`: giá trị đúng/sai
- `datetime64[ns]`: thời gian
- `category`: dữ liệu phân loại

DataFrame hỗ trợ các phép toán, lọc điều kiện, thống kê, nhóm, nối bảng và các thao tác phân tích mạnh mẽ khác.

Index	Columns	
	Name	Country
0	Anh	Brazil
1	Thai	Vietnam
2	Hoa	Vietnam

Hình 11.7: Minh họa cấu trúc DataFrame.

Xem và khám phá dữ liệu

Để hiểu tổng quan về tập dữ liệu, Pandas cung cấp nhiều hàm tiện ích giúp kiểm tra nhanh cấu trúc, kích thước, kiểu dữ liệu, tính phân bố, và tình trạng thiếu dữ liệu. Đây là bước quan trọng trước khi phân tích sâu hơn.

Một số hàm cơ bản thường dùng:

```
1 df.head()
2 df.tail(3)
3 len(df)
4 df.shape
5 df.index
6 df.columns
7 df.dtypes
8 df.count()
9 df.sample(3)
10 df['Country'].unique()
11 df['Country'].value_counts()
12 df.isnull().sum()
```

Giải thích các hàm:

- `df.head()`: Hiển thị 5 dòng đầu tiên của DataFrame. Thường dùng để kiểm tra nhanh cấu trúc và dữ liệu đầu bảng.
- `df.tail(3)`: Hiển thị 3 dòng cuối cùng. Hữu ích khi muốn kiểm tra các bản ghi ở cuối tập dữ liệu.
- `len(df)`: Trả về số lượng dòng trong DataFrame.
- `df.shape`: Trả về bộ số dạng (số dòng, số cột).
- `df.index`: Hiển thị đối tượng chỉ mục (index), thường là số nguyên hoặc datetime index.
- `df.columns`: Trả về danh sách tên các cột trong DataFrame.

- `df.dtypes`: Kiểu dữ liệu của từng cột, như `int64`, `object`, `float64`, v.v.
- `df.count()`: Đếm số giá trị không bị thiếu theo từng cột.
- `df.sample(3)`: Lấy ngẫu nhiên 3 dòng từ DataFrame.
- `df['Country'].unique()`: Trả về các giá trị duy nhất trong cột `Country`.
- `df['Country'].value_counts()`: Trả về tần suất xuất hiện của từng giá trị trong cột `Country`.
- `df.isnull().sum()`: Kiểm tra và thống kê số lượng giá trị bị thiếu theo từng cột.

`df.info()`

Cung cấp thông tin tổng quan về DataFrame: số dòng, tên cột, kiểu dữ liệu, số non-null, lượng bộ nhớ sử dụng. Đây là một trong những hàm cần gọi đầu tiên sau khi đọc dữ liệu.

```
1 df.info()
```

`df.describe()`

Thống kê mô tả các cột dạng số: gồm số lượng, giá trị trung bình, độ lệch chuẩn, min, max và các phân vị (25%, 50%, 75%).

```
1 df.describe()
```

Summary data

Index		Columns		
		Name	Country	Grade
0		Anh	Brazil	7.0
1		Thai	Vietnam	8.0
2		Hoa	Vietnam	9.0

	Grade
count	3.0
mean	8.0
std	1.0
min	7.0
25%	7.5
50%	8.0
75%	8.5
max	9.0

Hình 11.8: một số hàm cơ bản để xem nhanh cấu trúc và thông tin dữ liệu.

11.2.3 Truy xuất và xử lý dữ liệu

Truy xuất bằng index và slicing

Đối với `Series`, có thể truy xuất dữ liệu theo chỉ số (index) tương tự như từ điển:

```
1 s['b']
```

Ví dụ trên trả về giá trị tương ứng với nhãn chỉ mục 'b'. Đây là cách đơn giản và trực quan khi bạn làm việc với dữ liệu được gán nhãn như một danh sách có tên.

Đối với `DataFrame`, có thể dùng kỹ thuật slicing tương tự như với danh sách để chọn nhiều dòng:

```
1 df[1:]
```

Câu lệnh trên sẽ trả về toàn bộ các dòng từ chỉ số 1 trở về sau, tức là bỏ qua dòng đầu tiên. Đây là cú pháp nhanh gọn để loại bỏ dòng tiêu đề, header lõi, hoặc phục vụ các thao tác như chuẩn hóa dữ liệu đầu vào.

Index	Data		Index	Columns	
a	3	<code>1 s['b']</code> ✓ 0.0s	0	Name	<code>1 df[1:]</code> ✓ 0.0s
b	-5	-5	1	Country	Name Country
c	7		2	Vietnam	1 Thai Vietnam 2 Hoa Vietnam
d	4				

Hình 11.9: Truy xuất đơn giản với pandas.

Truy xuất với `.loc[]`, `.iloc[]`, `.at[]`, `.iat[]`

Pandas cung cấp 4 phương thức phổ biến để truy xuất dữ liệu từ `DataFrame` hoặc `Series`, dựa trên hai nguyên tắc: truy xuất theo nhãn (label) hoặc theo vị trí (index). Các phương thức này giúp lấy dữ liệu chính xác và hiệu quả trong từng ngữ cảnh cụ thể.

`.loc[]` – Truy xuất theo nhãn (label)

`.loc[]` dùng để truy xuất dữ liệu dựa trên nhãn của hàng và cột. Nhãn có thể là chuỗi hoặc giá trị trong chỉ mục được định nghĩa.

```
1 df.loc['row1', 'column1']
2 df.loc[0:3, ['name', 'age']]
```

Dòng đầu tiên truy xuất giá trị tại hàng có nhãn là ‘row1’ và cột có nhãn là ‘column1’. Dòng thứ hai lấy các dòng từ 0 đến 3 (bao gồm) với hai cột là ‘name’ và ‘age’.

`.iloc[]` – Truy xuất theo vị trí (chỉ số số nguyên)

`.iloc[]` sử dụng chỉ số số nguyên để truy xuất. Thứ tự bắt đầu từ 0 và hoạt động giống slicing trong Python.

```
1 df.iloc[0, 1]
2 df.iloc[0:3, 0:2]
```

Dòng đầu tiên truy xuất giá trị tại dòng thứ nhất và cột thứ hai (tính từ 0).
Dòng thứ hai lấy tất cả các dòng từ 0 đến 2 và các cột từ 0 đến 1.

.at[] – Truy xuất nhanh một giá trị theo nhãn

.at[] được tối ưu hóa để truy xuất một phần tử duy nhất theo nhãn của hàng và cột.

```
1 df.at['row1', 'column1']
```

Lệnh trên truy xuất trực tiếp giá trị tại hàng ‘row1’ và cột ‘column1’. Khi chỉ lấy một giá trị, at[] có hiệu suất cao hơn loc[].

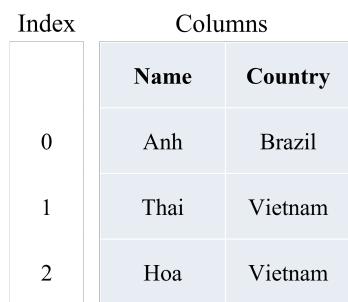
.iat[] – Truy xuất nhanh một giá trị theo vị trí

.iat[] là phiên bản hiệu suất cao của iloc[] dùng cho truy xuất một phần tử theo chỉ số nguyên.

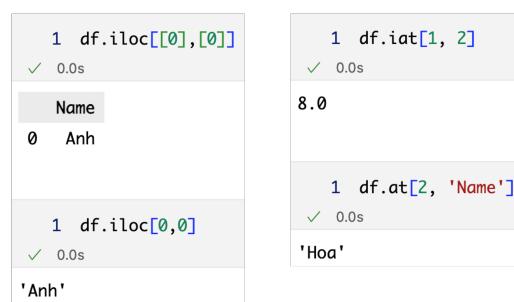
```
1 df.iat[0, 1]
```

Lệnh này truy xuất giá trị tại dòng thứ nhất và cột thứ hai theo vị trí số nguyên, rất hữu ích khi cần thao tác nhanh trên một ô dữ liệu duy nhất.

Tóm lại, loc[] và iloc[] được dùng để lấy nhiều hàng hoặc cột, trong khi at[] và iat[] được thiết kế riêng cho việc truy xuất một ô duy nhất với hiệu suất cao hơn.



Index	Columns	
	Name	Country
0	Anh	Brazil
1	Thai	Vietnam
2	Hoa	Vietnam



```

1 df.iloc[0,0]
✓ 0.0s
Name
0 Anh

1 df.iat[1, 2]
✓ 0.0s
8.0

1 df.at[2, 'Name']
✓ 0.0s
'Hoa'

```

Hình 11.10: Truy xuất với .loc, .iloc, .at, .iat

Nhóm dữ liệu

Trong phân tích dữ liệu, việc nhóm các bản ghi theo một hoặc nhiều tiêu chí là thao tác phổ biến để tính toán thống kê cho từng nhóm riêng biệt. Pandas cung cấp phương thức `groupby()` để thực hiện điều này một cách hiệu quả và linh hoạt.

`groupby` hoạt động theo quy trình: *Split* dữ liệu thành từng nhóm, *Apply* hàm xử lý (tính trung bình, tổng, đếm, v.v.), và *Combine* kết quả lại.

Ví dụ cơ bản:

```
1 df.groupby('department')['salary'].mean()
```

Lệnh trên nhóm dữ liệu theo cột `department`, sau đó tính lương trung bình theo từng phòng ban.

Có thể nhóm theo nhiều cột cùng lúc:

```
1 df.groupby(['department', 'gender'])['salary'].sum()
```

Câu lệnh này sẽ nhóm dữ liệu theo cặp (`department`, `gender`) và tính tổng lương của từng nhóm.

Kết hợp `groupby()` với các hàm thống kê như:

- `.mean()`: trung bình
- `.sum()`: tổng
- `.count()`: đếm số phần tử
- `.min()`, `.max()`: giá trị nhỏ nhất/lớn nhất
- `.agg()`: áp dụng nhiều hàm cùng lúc

`groupby()` là công cụ cực kỳ mạnh khi cần phân tích dữ liệu tổng hợp, so sánh nhóm hoặc chuẩn bị dữ liệu cho biểu đồ trực quan hóa.

Sắp xếp dữ liệu

Để sắp xếp dữ liệu trong `DataFrame` theo một hoặc nhiều cột, ta sử dụng phương thức `sort_values()`. Có thể sắp xếp tăng dần hoặc giảm dần bằng cách điều chỉnh tham số.

Ví dụ: sắp xếp theo cột `salary` tăng dần:

```
1 df.sort_values('salary')
```

Sắp xếp theo cột `salary` giảm dần:

```
1 df.sort_values('salary', ascending=False)
```

Sắp xếp theo nhiều cột:

```
1 df.sort_values(['department', 'salary'], ascending=[True, False])
```

Câu lệnh trên sắp xếp dữ liệu theo `department` tăng dần và trong từng phòng ban thì sắp xếp `salary` giảm dần.

Áp dụng hàm tùy chỉnh

`apply()` là phương thức linh hoạt cho phép áp dụng một hàm bất kỳ lên từng hàng hoặc từng cột của `DataFrame` hoặc `Series`. Rất hữu ích khi cần xử lý theo quy tắc phức tạp không có sẵn trong Pandas.

Ví dụ: chuẩn hóa dữ liệu trong cột `score` về thang điểm 10:

```
1 df['score'].apply(lambda x: x / 100 * 10)
```

Áp dụng một hàm xử lý cho từng dòng:

```
1 df.apply(lambda row: row['math'] + row['english'], axis=1)
```

Tham số `axis=1` cho biết áp dụng theo từng dòng. Nếu `axis=0`, hàm sẽ áp dụng theo từng cột.

Ngoài `lambda`, bạn có thể sử dụng bất kỳ hàm tùy chỉnh nào đã định nghĩa sẵn để truyền vào `apply()`.

	Columns		
	Name	Country	Grade
0	Anh	Brazil	7.0
1	Thai	Vietnam	8.0
2	Hoa	Vietnam	9.0

```
1 f = lambda x: x**2
2 df['Grade'].apply(f)
✓ 0.0s
```

```
0 14.0
1 16.0
2 18.0
Name: Grade, dtype: float64
```

Hình 11.11: Minh họa áp dụng hàm trong pandas.

Rolling

Phương thức `rolling()` trong Pandas tạo ra một đối tượng “cửa sổ trượt” (rolling window) cho phép áp dụng các hàm thống kê (như `mean`, `sum`, `std`, v.v.) lên các tập giá trị con liên tiếp theo chiều dọc của một cột. Đây là công cụ đặc biệt hữu ích trong các bài toán phân tích chuỗi thời gian, nơi cần quan sát xu hướng cục bộ hoặc tính toán chỉ số động.

- Cho phép áp dụng các phép biến đổi tổng hợp trên một phạm vi giới hạn (window) trượt theo từng dòng.
- Hữu ích để làm mượt dữ liệu, phát hiện xu hướng ngắn hạn, hoặc tính toán trung bình động.
- Có thể tùy chỉnh kích thước cửa sổ, lựa chọn căn chỉnh tâm cửa sổ, xử lý cạnh và các thông số khác để phù hợp với yêu cầu phân tích cụ thể.

Index	Columns		
	Name	Country	Grade
0	Anh	Brazil	7.0
1	Thai	Vietnam	8.0
2	Hoa	Vietnam	9.0

```
1 df['Grade'].rolling(3).sum()
✓ 0.0s
```

0	NaN
1	NaN
2	24.0
Name: Grade, dtype: float64	

Hình 11.12: Minh họa áp dụng rolling.

Giải thích code áp dụng rolling:

- Tính tổng của mỗi cửa sổ 3 giá trị liên tiếp trong cột `Grade`.
- Kết quả trả về là một Series mới, trong đó phần tử thứ ba chứa tổng của 3 giá trị đầu tiên.
- Những vị trí chưa đủ dữ liệu để tính toán sẽ nhận giá trị `NaN`.

Thao tác với chỉ mục

Hai phương thức `set_index()` và `reset_index()` cho phép kiểm soát và cấu trúc lại chỉ mục (index) của bảng dữ liệu. Chỉ mục đóng vai trò như khoá truy xuất nhanh và thường được sử dụng trong phân tích dữ liệu để phân nhóm, sắp xếp, hoặc truy vấn hiệu quả hơn.

- `set_index()` dùng để chuyển một (hoặc nhiều) cột dữ liệu thành chỉ mục, giúp định danh các hàng theo thuộc tính cụ thể.
- Việc đặt chỉ mục có thể hỗ trợ trong việc lọc, phân nhóm, hay kết nối dữ liệu (join/merge) hiệu quả hơn.
- `reset_index()` dùng để hoàn nguyên DataFrame về dạng chỉ mục mặc định, chuyển cột chỉ mục hiện tại trở lại làm dữ liệu thông thường.
- Hữu ích trong các bước tiền xử lý và hậu xử lý dữ liệu, đặc biệt khi cần sắp xếp lại cấu trúc bảng sau thao tác nhóm hoặc tổng hợp.

Index	Columns		
	Name	Country	Grade
0	Anh	Brazil	7.0
1	Thai	Vietnam	8.0
2	Hoa	Vietnam	9.0

Hình 11.13: Minh họa thao tác với chỉ mục.

11.2.4 Làm sạch dữ liệu

Dữ liệu thu thập từ thực tế hiếm khi hoàn chỉnh. Chúng thường có những vấn đề phổ biến như: ô trống (missing values), định dạng sai (wrong format), trùng lặp (duplicates), hay dữ liệu bất hợp lý (invalid entries). Việc xử lý các vấn đề này là bước thiết yếu trước khi phân tích hay xây dựng mô hình.

Xử lý dữ liệu NaN

Các ô trống trong Pandas thường được biểu diễn bằng giá trị NaN. Có thể loại bỏ hoặc thay thế chúng bằng các phương thức phù hợp tùy theo hoàn cảnh.

```
1 df.dropna()  
2 df.dropna(axis=1)  
3 df.fillna(0)  
4 df['age'].fillna(df['age'].mean())
```

- `df.dropna()`: Loại bỏ các dòng chứa ít nhất một giá trị thiếu, thường dùng khi số lượng dòng đủ lớn và không muốn sử dụng dữ liệu không hoàn chỉnh.
- `df.dropna(axis=1)`: Xoá toàn bộ cột có bất kỳ giá trị thiếu nào. Dùng khi muốn giữ lại các biến không có thiếu sót.
- `df.fillna(0)`: Thay toàn bộ giá trị thiếu bằng số 0, thường dùng cho các biến định lượng khi số 0 có ý nghĩa mặc định (như không có).
- `df['age'].fillna(df['age'].mean())`: Điền giá trị thiếu trong một cột bằng trung bình của cột đó. Phương pháp thường dùng trong phân tích thống kê khi muốn giữ lại dữ liệu nhưng không tạo thiên lệch quá lớn.

Sửa định dạng dữ liệu

Định dạng dữ liệu không đúng có thể gây lỗi khi tính toán hoặc trực quan hóa. Pandas hỗ trợ ép kiểu (type conversion) để đảm bảo tính chính xác của dữ liệu.

```
1 df['age'] = pd.to_numeric(df['age'], errors='coerce')
2 df['join_date'] = pd.to_datetime(df['join_date'])
3 df['is_active'] = df['is_active'].astype(bool)
```

- `pd.to_numeric(...)`: Ép cột `age` về kiểu số. Nếu gặp giá trị không thể chuyển đổi (như chuỗi chữ), giá trị đó sẽ trở thành `NaN`. Điều này cho phép xử lý lỗi mà không bị ngắt dòng.
- `pd.to_datetime(...)`: Chuyển định dạng ngày tháng không chuẩn về kiểu `datetime`, giúp thao tác dễ dàng hơn trong các phân tích theo thời gian.
- `astype(bool)`: Ép kiểu cột về giá trị logic (`True/False`), cần thiết trong các bài toán lọc hoặc điều kiện logic.

Xoá dữ liệu trùng lặp

Các bản ghi trùng lặp thường làm sai lệch thống kê hoặc gây thiên lệch trong mô hình. Pandas hỗ trợ nhiều phương pháp phát hiện và loại bỏ trùng lặp linh hoạt.

```
1 df.drop_duplicates()
2 df.drop_duplicates(subset='email')
3 df.drop_duplicates(keep='last')
```

- `drop_duplicates()`: Xoá các dòng trùng lặp hoàn toàn (mọi giá trị giống nhau). Dùng trong các bảng dữ liệu tổng hợp hoặc xuất từ hệ thống lỗi.

- `subset='email'`: Chỉ xét trùng theo một cột cụ thể. Ví dụ, nếu hai bản ghi có cùng địa chỉ email thì coi là trùng.
- `keep='last'`: Giữ lại dòng trùng cuối cùng thay vì dòng đầu tiên. Phù hợp khi dữ liệu mới hơn nằm sau.

Ngoài việc xoá trực tiếp, bạn cũng có thể lọc ra các dòng trùng để kiểm tra trước:

```
1 df[df.duplicated()]
```

- Trả về các dòng được đánh dấu là trùng lặp. Thường dùng để kiểm tra trước khi quyết định loại bỏ hoặc xử lý.

11.3 IMDB Movie Dataset

IMDB Movie dataset là một bộ dữ liệu đánh giá phim, dùng để phân tích mức độ quan tâm của phim theo một số tiêu chí như: đạo diễn, diễn viên, tên phim... nhằm đưa ra những góc nhìn hỗ trợ dự đoán trong tương lai.

Các bước cần thực hiện trong bài toán:

- Read data
- View the data
- Understand some basic information about the data
- Data Selection – Indexing and Slicing data
- Data Selection – Based on Conditional filtering
- Groupby operations
- Sorting operation
- View missing values
- Deal with missing values - Deleting
- Deal with missing values - Filling
- Apply() functions

11.3.1 Read data

Để đọc một file .csv trong pandas, ta có thể dùng hàm `read_csv()` như sau:

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4
5 # Download dataset
6 !gdown 1I3GGJLbpS1Y1dNbIDIPKfiuZUdiSa2hq
7
8 dataset_path = 'IMDB-Movie-Data.csv'
```

```
9 # Read data from .csv file
10 data = pd.read_csv(dataset_path)
11 data
```

Ngoài ra, ta có thể đọc đồng thời chỉ định cột làm chỉ mục cho bảng dữ liệu (mặc định pandas sẽ tự tạo một cột chỉ mục riêng). Ở đây, ta có thể chọn cột **Title** làm cột chỉ mục như sau (cột chỉ mục không được chứa giá trị trùng lặp):

```
1 # Read data with specified explicit index.
2 # We will use this later in our analysis
3 data_indexed = pd.read_csv(dataset_path, index_col="Title")
4 data_indexed
```

11.3.2 View data

Sử dụng đoạn code sau để xem thông tin 5 hàng đầu tiên của bảng dữ liệu.

```
1 # Preview top 5 rows using head()
2 data.head()
```

Rank	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)
0 1	Guardians of the Galaxy	Action, Adventure, Sci-Fi	A group of intergalactic criminals are forced ...	James Gunn	Chris Pratt, Vin Diesel, Bradley Cooper, Zoe S...	2014	121	8.1	757074	333.13
1 2	Prometheus	Adventure, Mystery, Sci-Fi	Following clues to the origin of mankind, a te...	Ridley Scott	Noomi Rapace, Logan Marshall-Green, Michael F...	2012	124	7.0	485820	126.46
2 3	Split	Horror, Thriller	Three girls are kidnapped by a man with a diag...	M. Night Shyamalan	James McAvoy, Anya Taylor-Joy, Haley Lu Richardson	2016	117	7.3	157606	138.12
3 4	Sing	Animation, Comedy, Family	In a city of humanoid animals, a hustling thea...	Christophe Lourdelet	Matthew McConaughey, Reese Witherspoon, Seth MacFarlane	2016	108	7.2	60545	270.32
4 5	Suicide Squad	Action, Adventure, Fantasy	A secret government agency recruits some of th...	David Ayer	Will Smith, Jared Leto, Margot Robbie, Viola Davis	2016	123	6.2	393727	325.02

Hình 11.14: Một số mẫu dữ liệu đầu tiên của bộ dữ liệu

11.3.3 Understand some basic information about the data

Lệnh sau cung cấp một số thông tin cơ bản về bộ dữ liệu như kiểu dữ liệu, số hành non-null,...

```

1 #Let's first understand the basic information about this data
2 data.info()

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Rank              1000 non-null    int64  
 1   Title             1000 non-null    object  
 2   Genre             1000 non-null    object  
 3   Description       1000 non-null    object  
 4   Director          1000 non-null    object  
 5   Actors            1000 non-null    object  
 6   Year              1000 non-null    int64  
 7   Runtime (Minutes) 1000 non-null    int64  
 8   Rating            1000 non-null    float64 
 9   Votes              1000 non-null    int64  
 10  Revenue (Millions) 872 non-null    float64 
 11  Metascore         936 non-null    float64 
dtypes: float64(3), int64(4), object(5)
memory usage: 93.9+ KB

```

Hình 11.15: Thông tin cơ bản về bảng dữ liệu

```

1 # the statistical information about this data
2 data.describe()

```

Ở đây ta có thể thấy:

- Giá trị min và max của Year, tức dataset chứa các bộ phim từ **2006** tới **2016**.
- Rating trung bình cho các bộ phim là **6.7**, thấp nhất là **1.9**, cao nhất là **9.0**.
- Doanh thu cao nhất đạt được là **936.6** triệu dollar.

	Rank	Year	Runtime (Minutes)	Rating	Votes	Revenue (Millions)	Metascore
count	1000.000000	1000.000000	1000.000000	1000.000000	1.000000e+03	872.000000	936.000000
mean	500.500000	2012.783000	113.172000	6.723200	1.698083e+05	82.956376	58.985043
std	288.819436	3.205962	18.810908	0.945429	1.887626e+05	103.253540	17.194757
min	1.000000	2006.000000	66.000000	1.900000	6.100000e+01	0.000000	11.000000
25%	250.750000	2010.000000	100.000000	6.200000	3.630900e+04	13.270000	47.000000
50%	500.500000	2014.000000	111.000000	6.800000	1.107990e+05	47.985000	59.500000
75%	750.250000	2016.000000	123.000000	7.400000	2.399098e+05	113.715000	72.000000
max	1000.000000	2016.000000	191.000000	9.000000	1.791916e+06	936.630000	100.000000

Hình 11.16: Tổng quan thống kê dữ liệu từ dataset

11.3.4 Data Selection – Indexing and Slicing data

Từ bảng dữ liệu, ta có thể tách bất kì cột nào trong bảng dữ liệu để trở thành một Series hoặc một DataFrame, tùy vào phương thức tách ta sử dụng. Ở đây, ta sẽ tách một số cột trong **data** sử dụng kỹ thuật **Indexing**. Để tách cột thành Series, ta thực hiện:

```
1 # Extract data as series
2 genre = data["Genre"]
3 genre
```

```
0      Action,Adventure,Sci-Fi
1      Adventure,Mystery,Sci-Fi
2          Horror,Thriller
3      Animation,Comedy,Family
4      Action,Adventure,Fantasy
...
995     Crime,Drama,Mystery
996         Horror
997     Drama,Music,Romance
998     Adventure,Comedy
999     Comedy,Family,Fantasy
Name: Genre, Length: 1000, dtype: object
```

Hình 11.17: Tách cột **Genre** thành một Series

Để tách cột thành DataFrame, ta thực hiện:

```
1 # Extract data as dataframe
2 genre = data[["Genre"]]
3 genre
```

Genre
0 Action,Adventure,Sci-Fi
1 Adventure,Mystery,Sci-Fi
2 Horror,Thriller
3 Animation,Comedy,Family
4 Action,Adventure,Fantasy
...
995 Crime,Drama,Mystery
996 Horror
997 Drama,Music,Romance
998 Adventure,Comedy
999 Comedy,Family,Fantasy

1000 rows × 1 columns

Hình 11.18: Tách cột **Genre** thành một Dataframe

Ta có thể chọn và tách cùng một lúc nhiều cột với nhau, tạo thành một DataFrame mới:

```
1 columns = data[["Title", "Genre", "Actors", "Director", "Rating"]]
2 columns
```

Đối với việc tách hàng, ta có thể tách ra một số lượng hàng nhất định, từ chỉ mục X đến chỉ mục Y trong bảng dữ liệu, gọi là **Slicing**. Ví dụ, để tách các hàng thứ 10 đến thứ 15 và dựa vào hình kết quả dưới để hoàn thiện đoạn code sau:

```
1 get_columns = # Your code here
2 get_columns
```

	Title	Rating	Revenue (Millions)
10	Fantastic Beasts and Where to Find Them	7.5	234.02
11	Hidden Figures	7.8	169.27
12	Rogue One	7.9	532.17
13	Moana	7.7	248.75
14	Colossal	6.4	2.87

Hình 11.19: Tách một số cột tạo thành một DataFrame mới

11.3.5 Data Selection – Based on Conditional filtering

Ta có lấy các hàng trong bảng dữ liệu dựa trên một số điều kiện cần tuân theo. Ví dụ, ta mong muốn lấy các bộ phim từ 2010 tới 2015, với rating nhỏ hơn 6.0 nhưng lại có doanh thu thuộc top 5% trên toàn bộ dataset. Theo đó, ta có thể triển khai code như sau và dựa vào hình kết quả dưới để hoàn thiện đoạn code sau:

```
1 get_data = # Your code here
2 get_data
```

Rank	Title	Genre	Description	Director	Actors	Year	Runtime (Minutes)	Rating	Votes
941	The Twilight Saga: Eclipse	Adventure,Drama,Fantasy	As a string of mysterious killings grips Seatt...	David Slade	Kristen Stewart, Robert Pattinson, Taylor Laut...	2010	124	4.9	192740

Hình 11.20: Phim với doanh thu cao trong giai đoạn năm 2010-2015

11.3.6 Groupby Operations

Groupby là một phép gom nhóm dữ liệu dựa trên một hoặc nhiều biến (ở đây là cột dữ liệu trong bảng). Ví dụ, ta có thể tìm số rating trung bình mà các đạo diễn đạt được bằng cách gom nhóm các chỉ số Rating của các bộ phim theo Director.

```
1 data.groupby("Director")[["Rating"]].mean().head()
```

Rating	
Director	
Aamir Khan	8.5
Abdellatif Kechiche	7.8
Adam Leon	6.5
Adam McKay	7.0
Adam Shankman	6.3

Hình 11.21: Sử dụng groupby để tìm số rating trung bình đạt được của các đạo diễn trong bộ dữ liệu.

11.3.7 Sorting Operations

Sorting cho phép ta sắp xếp các hàng trong bảng dữ liệu theo thứ tự tăng/giảm dần dựa theo giá trị của cột nào đó trong bảng dữ liệu. Ví dụ, dựa trên kết quả groupby phần trước, ta có thể tìm top 5 đạo diễn đạt số rating trung bình cao nhất, dựa vào kết quả trong hình dưới đây để hoàn thiện đoạn code sau:

```
1 sorted_values = # Your code here
2 sorted_values
```

Rating	
Director	
Nitesh Tiwari	8.80
Christopher Nolan	8.68
Olivier Nakache	8.60
Makoto Shinkai	8.60
Aamir Khan	8.50

Hình 11.22: 5 đạo diễn có được số Rating trung bình cao nhất.

11.3.8 View missing values

Các bộ dữ liệu thường sẽ xuất hiện tình trạng bị giá trị rỗng (missing value) trong một vài trường thông tin của một số mẫu dữ liệu. Khi xử lý dữ liệu, ta cần khắc phục vấn đề này. Vì vậy, việc đầu tiên ta cần kiểm tra xem vị trí bị mất mát dữ liệu theo cách sau:

```
1 # To check null values row-wise
2 data.isnull().sum()
```

Rank	0
Title	0
Genre	0
Description	0
Director	0
Actors	0
Year	0
Runtime (Minutes)	0
Rating	0
Votes	0
Revenue (Millions)	128
Metascore	64
dtype:	int64

Hình 11.23: Bảng tổng số lượng các giá trị null có trong từng cột của bảng dữ liệu.

Ở đây ta thấy Revenue (Millions) và Metascore là 2 cột có chứa dữ liệu null. Để xử lý vấn đề mất mát dữ liệu, có hai phương án chính: hoặc thế các vùng trống bằng một giá trị nào đó hoặc loại bỏ chúng.

11.3.9 Deal with missing values - Deleting

Đối với phương án loại bỏ, ta có thể loại bỏ toàn bộ cột chứa nhiều giá trị null (nếu có thể) hoặc chỉ loại bỏ các hàng chứa giá trị null. Đối với xóa cột, ta thực hiện:

```

1 # Use drop function to drop columns
2 data.drop("Metascore", axis=1).head()

```

Lệnh trên vẫn chưa drop data thực trên server cho tới khi ta thêm `inplace=True`.

Đối với xóa hàng, ta dùng:

```
1 data.dropna()
```

11.3.10 Dealing with missing values - Filling

Đối với phương án thế giá trị mới vào các ô trống, ta có thể sử dụng các giá trị mean, median... của cột dữ liệu tương ứng để thay thế (việc chọn giá trị để thay thế còn tùy thuộc vào tính chất của bộ dữ liệu, bài toán đang giải quyết...). Ví dụ, có một vài hàng có Revenue mang giá trị null, ta có thể gán cho nó giá trị trung bình, hoàn thiện đoạn code sau:

```

1 revenue_mean = # Your code here
2 print("The mean revenue is: ", revenue_mean)
3
4 # We can fill the null values with this mean revenue
5 data_indexed["Revenue (Millions)"].fillna(revenue_mean, inplace=True)

```

11.3.11 apply() functions

Apply functions được dùng khi ta muốn thực thi một hàm nào đó lên các hàng trong bảng dữ liệu. Sau khi thực thi, kết quả trả về từ hàm chính là giá trị mới của hàng tương ứng. Ví dụ, ta muốn phân loại phim theo ba mức độ ["Good", "Average", "Bad"] dựa trên **Rating**, ta có thể định nghĩa một hàm để làm điều này và apply nó lên DataFrame kết hợp dựa vào hình dưới để hoàn thiện code sau:

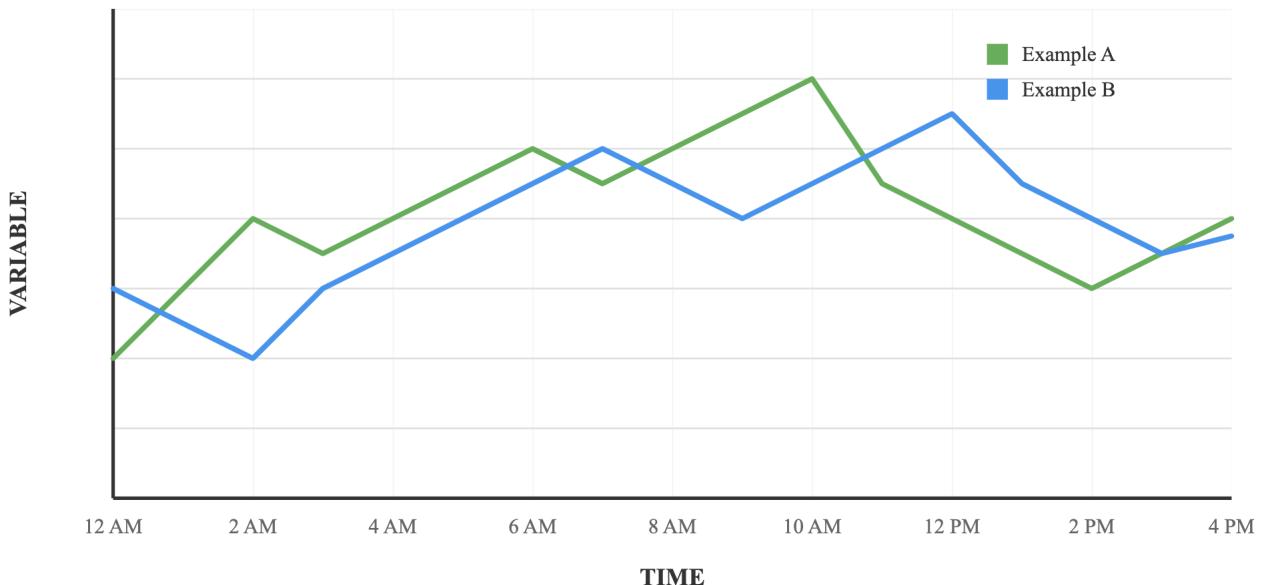
```
1 # Classify movies based on ratings
2 def rating_group(rating):
3     if rating >= 7.5:
4         return "Good"
5     elif rating >= 6.0:
6         return "Average"
7     else:
8         return "Bad"
9
10 # Lets apply this function on our movies data
11 # creating a new variable in the dataset to hold the rating category
12 data["Rating_category"] = # Your code here
13
14 data[["Title", "Director", "Rating", "Rating_category"]].head(5)
```

	Title	Director	Rating	Rating_category
0	Guardians of the Galaxy	James Gunn	8.1	Good
1	Prometheus	Ridley Scott	7.0	Average
2	Split	M. Night Shyamalan	7.3	Average
3	Sing	Christophe Lourdelet	7.2	Average
4	Suicide Squad	David Ayer	6.2	Average

Hình 11.24: DataFrame sau khi được apply hàm `rating_group()`. Kết quả trả về sau khi thực thi hàng này sẽ được đưa vào một cột mới mang tên `Rating_category`

11.4 Time Series Dataset

Time series data là một dạng dữ liệu với giá trị được đo lường tại những điểm khác nhau theo thời gian. Một số dữ liệu time series được phân bố theo tần suất nhất định, ví dụ như thời tiết trong 1 giờ, lượng truy cập website trong ngày, tổng doanh thu trong tháng... Dữ liệu time series cũng có thể phân bố với khoảng cách không đều, ví dụ như số lượng cuộc gọi khẩn cấp trong ngày hoặc nhật ký hệ thống.



Hình 11.25: Hình dạng đồ thị của dữ liệu time-series

Trong phần này, chúng ta sẽ khai thác khía cạnh sắp xếp và trực quan hóa dữ liệu cho time series. Cụ thể với dữ liệu time series cho năng lượng, ta sẽ làm quen với áp dụng của các kỹ thuật time-based indexing, resampling, và rolling. Việc này sẽ giúp ta phân tích được các khía cạnh thông tin ẩn quan trọng trong dữ liệu. Ví dụ, Rolling windows có thể giúp ta khám phá các biến thể về nhu cầu điện và cung cấp năng lượng tái tạo theo thời gian. Chúng ta dùng bộ dữ liệu daily time series của Open Power System Data (OPSD) ở Đức, gồm tổng lượng tiêu thụ điện, sản xuất điện gió và sản xuất điện mặt trời trên toàn quốc trong giai đoạn 2006-2017. Các bạn tải bộ dữ

liệu `opsd_germany_daily.csv` dựa vào đoạn code trong phần dưới đây.

Chúng ta sẽ thực hiện các vấn đề sau:

- Read dataset
- Time-based indexing
- Visualizing time series data
- Seasonality
- Frequencies
- Resampling
- Rolling windows
- Trends

Chúng ta sẽ khám phá mức tiêu thụ và sản xuất điện ở Đức thay đổi theo thời gian như thế nào, và trả lời các câu hỏi:

- Khi nào mức tiêu thụ điện thường cao nhất và thấp nhất?
- Năng lượng gió và mặt trời được sản xuất đã thay đổi theo mùa như thế nào?
- Xu hướng dài hạn trong tiêu thụ điện, năng lượng mặt trời và năng lượng gió là gì?
- So sánh tỷ lệ sản lượng năng lượng gió và mặt trời với mức tiêu thụ năng lượng gió và mặt trời, và tỷ lệ này đã thay đổi như thế nào theo thời gian?

11.4.1 Read dataset

Đầu tiên, ta vẫn dùng hàm `read_csv()` để đọc bảng dữ liệu:

```
1 # Download dataset
2 !gdown 1qW7tPUw3u3Dn4kuBEoV8XwLYpetBDVjr
3
```

```

4 # Import libs
5 import pandas as pd
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8
9 # Read data
10 dataset_path = "./opsd_germany_daily.csv"
11
12 # Read data from .csv file
13 opsd_daily = pd.read_csv(dataset_path)
14
15 print(opsd_daily.shape)
16 print(opsd_daily.dtypes)
17 opsd_daily.head(3)

```

Ta được kết quả như hình bên dưới, có thể quan sát thấy nhiều giá trị bị bỏ trống ở các cột Wind, Solar, Wind+Solar:

```
(4383, 5)
Date          object
Consumption   float64
Wind          float64
Solar          float64
Wind+Solar    float64
dtype: object
```

	Date	Consumption	Wind	Solar	Wind+Solar
0	2006-01-01	1069.184	NaN	NaN	NaN
1	2006-01-02	1380.521	NaN	NaN	NaN
2	2006-01-03	1442.533	NaN	NaN	NaN

Hình 11.26: Một số dữ liệu đầu tiên của DataFrame

Đối với dạng dữ liệu Time Series, ta có thể chọn cột **Date** làm index (vì giá trị cột này trong bộ dữ liệu luôn là duy nhất (unique)):

```

1 dataset_path = "./opsd_germany_daily.csv"
2 opsd_daily = opsd_daily.set_index("Date")
3 opsd_daily.head(3)

```

	Consumption	Wind	Solar	Wind+Solar
Date				
2006-01-01	1069.184	NaN	NaN	NaN
2006-01-02	1380.521	NaN	NaN	NaN
2006-01-03	1442.533	NaN	NaN	NaN

Hình 11.27: Bảng dữ liệu sau khi chọn cột Date làm index

Ta có thể thực hiện lại bước load file và lúc này, chỉ định cột sẽ làm chỉ mục ngay từ lúc thực hiện lời gọi hàm, đồng thời tạo thêm các số cột **Year**, **Month**, **Weekday** trích từ cột **Date** để thuận tiện cho việc xử lý một số bước về sau:

```

1 opsd_daily = pd.read_csv(dataset_path, index_col=0, parse_dates=True
                           )
2
3 opsd_daily["Year"] = opsd_daily.index.year
4 opsd_daily["Month"] = opsd_daily.index.month
5 opsd_daily["Weekday Name"] = opsd_daily.index.day_name()
6
7 opsd_daily.sample(5, random_state=0)

```

	Consumption	Wind	Solar	Wind+Solar	Year	Month	Weekday	Name
Date								
2008-08-23	1152.011	NaN	NaN	NaN	2008	8	Saturday	
2013-08-08	1291.984	79.666	93.371	173.037	2013	8	Thursday	
2009-08-27	1281.057	NaN	NaN	NaN	2009	8	Thursday	
2015-10-02	1391.050	81.229	160.641	241.870	2015	10	Friday	
2009-06-02	1201.522	NaN	NaN	NaN	2009	6	Tuesday	

Hình 11.28: DataFrame với các cột mới: Year, Month, Weekday

11.4.2 Time-based indexing

Một trong những tính năng nổi trội của pandas khi xử lý dữ liệu time series là tính năng time-based indexing, liên quan đến việc dùng dates và times để

tổ chức và truy cập dữ liệu (khá giống với Indexing ở phần trước nhưng giá trị lúc này sẽ là ngày tháng năm). Việc này cho phép ta dùng loc accessor để thực thi. Ví dụ, ta có thể truy cập dữ liệu theo một khoảng thời gian từ ngày **2014-01-20** đến ngày **2014-01-22**:

```
1 opsd_daily.loc["2014-01-20":"2014-01-22"]
```

Date	Consumption	Wind	Solar	Wind+Solar	Year	Month	Weekday	Name
2014-01-20	1590.687	78.647	6.371	85.018	2014	1	Monday	
2014-01-21	1624.806	15.643	5.835	21.478	2014	1	Tuesday	
2014-01-22	1625.155	60.259	11.992	72.251	2014	1	Wednesday	

Hình 11.29: Lấy các mẫu dữ liệu từ ngày 20/1/2014 đến 22/1/2014

Một tính năng khác của pandas là partial-string indexing, cho phép ta Slicing theo mô tả thời gian một cách chung chung, không cần cụ thể ngày tháng năm như ở phần trên. Ví dụ:

```
1 new_opsd_df = opsd_daily.loc["2012-02"]
2 new_opsd_df.head()
```

Date	Consumption	Wind	Solar	Wind+Solar	Year	Month	Weekday	Name
2012-02-01	1511.866	199.607	43.502	243.109	2012	2	Wednesday	
2012-02-02	1563.407	73.469	44.675	118.144	2012	2	Thursday	
2012-02-03	1563.631	36.352	46.510	82.862	2012	2	Friday	
2012-02-04	1372.614	20.551	45.225	65.776	2012	2	Saturday	
2012-02-05	1279.432	55.522	54.572	110.094	2012	2	Sunday	

Hình 11.30: Partial-string indexing. Với việc chỉ đặt ‘2012-02’, ta có thể lấy được toàn bộ các mẫu dữ liệu thuộc ‘2012-02’.

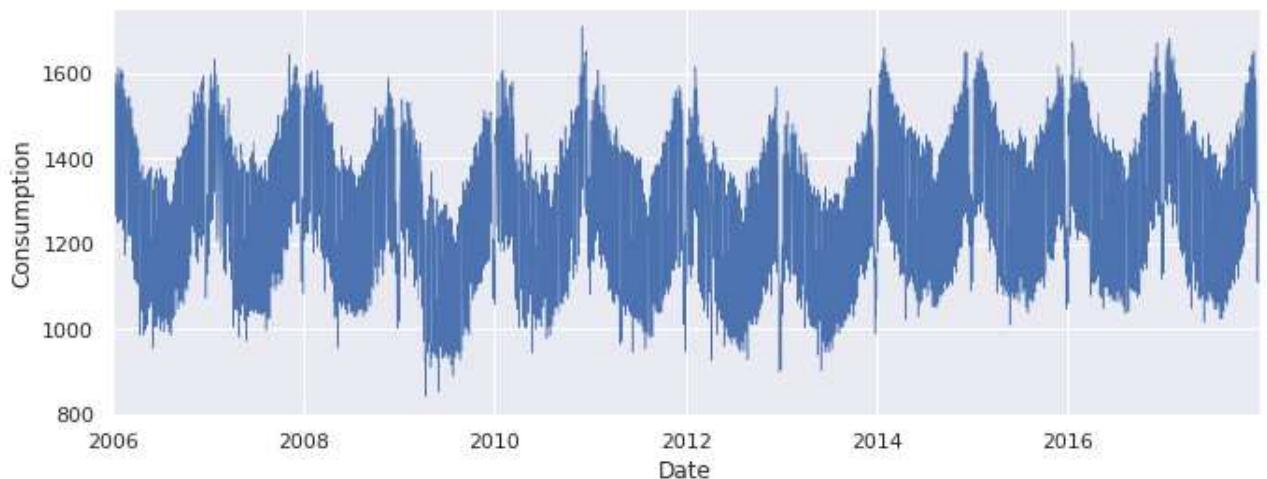
11.4.3 Visualizing time series data:

Với việc pandas có hỗ trợ trực quan hóa dữ liệu lên đồ thị, phối hợp với thư viện seaborn ta có thể dễ dàng trực quan hóa được dữ liệu time-series lên đồ thị. Ví dụ, ta trực quan (plot) dữ liệu cột **Consumption** như sau:

```

1 sns.set(rc={"figure.figsize":(11, 4)})
2 col_to_plot = "Consumption"
3 opsd_daily[col_to_plot].plot(linewidth=0.5)
4 plt.ylabel(col_to_plot)
5 plt.show()

```



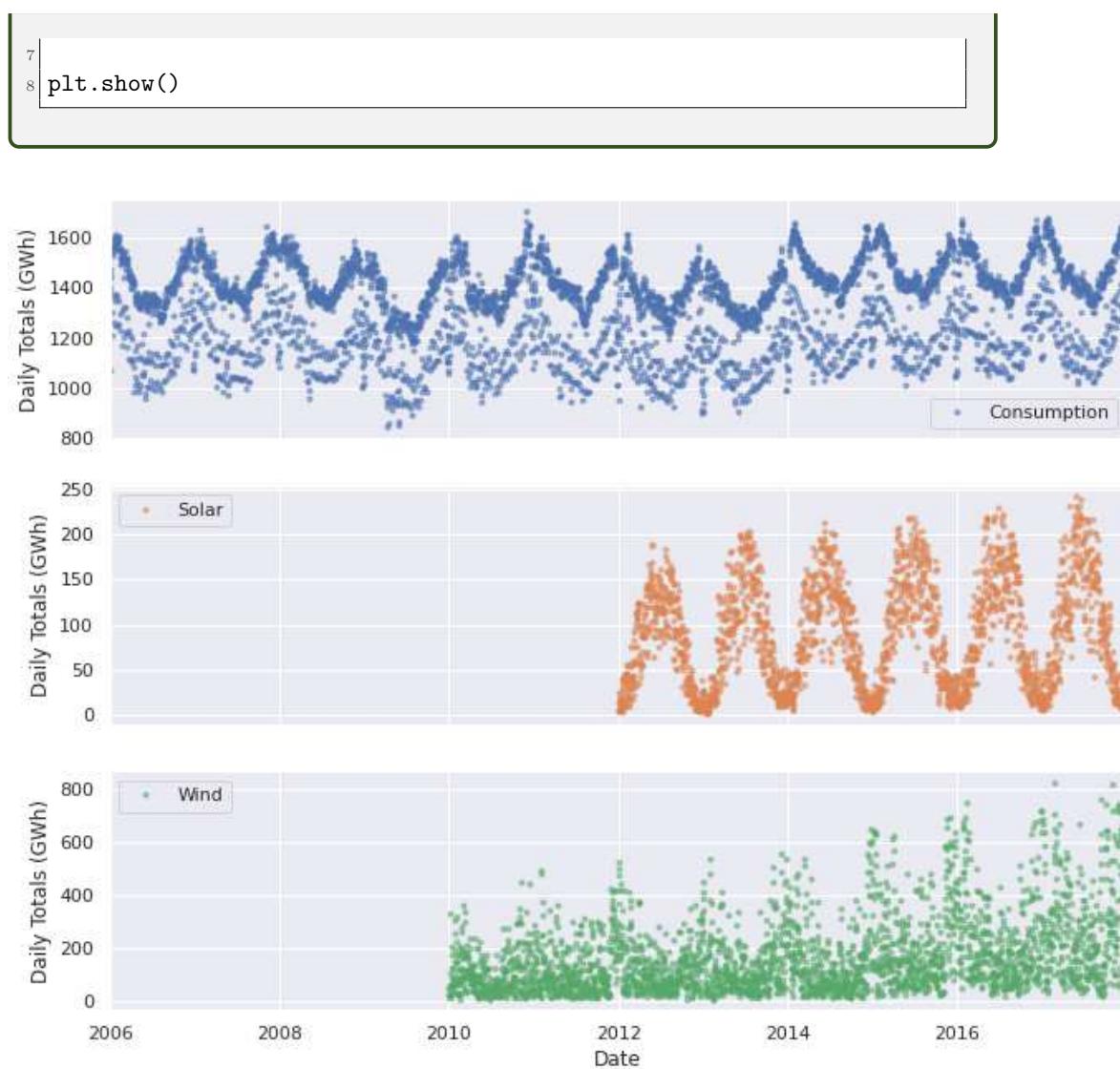
Hình 11.31: Đồ thị dữ liệu về mức tiêu thụ điện năng hằng ngày tại Đức

Ta có thể plot cùng lúc một số cột dữ liệu khác thành từng đồ thị riêng lẻ. Dựa vào hình và đoạn code sau để hoàn thiện code:

```

1 cols_plot = ["Consumption", "Solar", "Wind"]
2 axes = opsd_daily[cols_plot].plot(
3     # your code here
4 )
5 for ax in axes:
6     ax.set_ylabel("Daily Totals (GWh)")

```



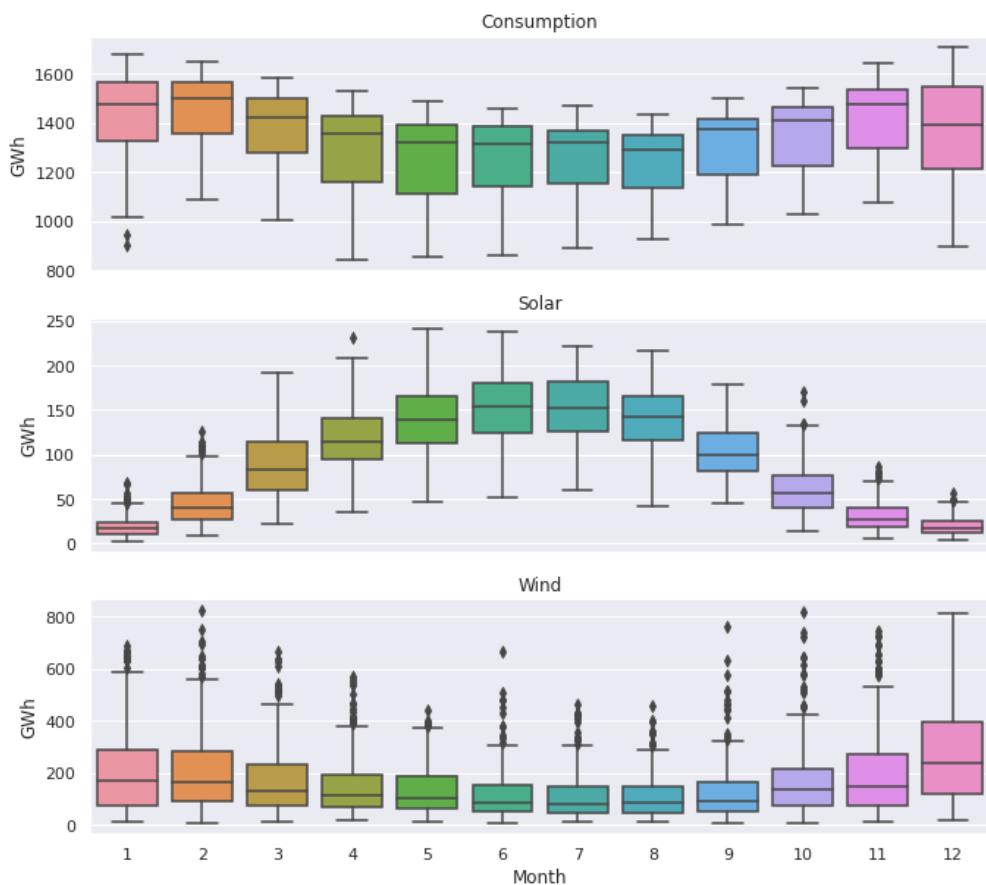
Hình 11.32: Đồ thị về mức tiêu thụ điện, sản lượng điện năng từ mặt trời và sản lượng điện năng từ gió

11.4.4 Seasonality

Tạm dịch: tính thời vụ. Chỉ số về các đặc trưng lặp đi lặp lại trong một khoảng thời gian cố định xuyên suốt các năm. Các dạng đặc trưng này thường được ảnh hưởng bởi rất nhiều yếu tố khác nhau. Ở trong dữ liệu của bài, ta có

thể khai phá tính thời vụ của dữ liệu, dùng seaborn để vẽ, và group dữ liệu thành từng nhóm như sau:

```
1 fig, axes = plt.subplots(3, 1, figsize=(11, 10), sharex=True)
2 for name, ax in zip(["Consumption", "Solar", "Wind"], axes):
3     sns.boxplot(data=opsd_daily, x="Month", y=name, ax=ax)
4     ax.set_ylabel("GWh")
5     ax.set_title(name)
6     # Remove the automatic x-axis label from all but the bottom
       subplot
7     if ax != axes[-1]:
8         ax.set_xlabel("")
```



Hình 11.33: Biểu diễn phân bố của các cột Consumption, Wind, Solar theo Month

11.4.5 Frequencies

Trong DatetimeIndex của pandas, ta có thể sử dụng các giá trị thời gian sẵn có để tạo thành một chuỗi giá trị theo tần suất. Ví dụ, với hai giá trị ‘1998-03-10’ và ‘1998-03-14’, ta có thể tạo một danh sách thời gian với tần suất theo ngày. Tức danh sách mới của chúng ta trở thành: ‘1998-03-10’, ‘1998-03-11’, ‘1998-03-12’, ‘1998-03-13’, ‘1998-03-14’. Việc này được thực hiện bằng cách cài đặt thuộc tính ‘freq’.

```
1 pd.date_range("1998-03-10", "1998-03-15", freq="D")
```

```
DatetimeIndex(['1998-03-10', '1998-03-11', '1998-03-12', '1998-03-13',
               '1998-03-14', '1998-03-15'],
              dtype='datetime64[ns]', freq='D')
```

Hình 11.34: Ví dụ về lấy tần suất theo ngày từ 10/3/1998 đến 15/3/1998

Với tính năng này của pandas, ta có thể thực hiện việc thế dữ liệu bị mất bằng kỹ thuật forward fill (ffill). Kỹ thuật này liên quan đến việc sử dụng giá trị ghi nhận được tại thời điểm trước đó làm giá trị thay thế cho toàn bộ giá trị bị mất mát sau đó trước khi gấp được mẫu dữ liệu có giá trị. Ví dụ, giả sử ta biết được giá trị Consumption của một vài ngày như sau:

```
1 # To select an arbitrary sequence of date/time values from a pandas
   time series,
2 # we need to use a DatetimeIndex, rather than simply a list of date/
   time strings
3 times_sample = pd.to_datetime(["2013-02-03", "2013-02-06", "2013-02-
   08"])
4 # Select the specified dates and just the Consumption column
5 consum_sample = opsd_daily.loc[times_sample, ["Consumption"]].copy()
6 consum_sample
```

Consumption	
2013-02-03	1109.639
2013-02-06	1451.449
2013-02-08	1433.098

Hình 11.35: Lấy dữ liệu của 3 ngày trong bộ dữ liệu gốc làm ví dụ mẫu

Dựa vào kết quả trong hình và mô tả trong code để hoàn thiện code sau:

```
1 # Convert the data to daily frequency, without filling any missings
2 consum_freq = # Your code here
3 # Create a column with missings forward filled
4 consum_freq["Consumption - Forward Fill"] = consum_sample.asfreq("D"
   , method="ffill")
5 consum_freq
```

	Consumption	Consumption - Forward Fill
2013-02-03	1109.639	1109.639
2013-02-04	NaN	1109.639
2013-02-05	NaN	1109.639
2013-02-06	1451.449	1451.449
2013-02-07	NaN	1451.449
2013-02-08	1433.098	1433.098

Hình 11.36: Thực hiện ffill vào các ngày khác trong phạm vi từ ngày 3/2/2013 đến 8/2/2013

Với giá trị tiêu thụ điện năng của 3 ngày, ta có thể thế giá trị cho các ngày còn lại trong phạm vi của 3 ngày trên sử dụng ffill.

11.4.6 Resampling

Là kỹ thuật dùng để thay đổi tần số biểu diễn của bộ dữ liệu time series, có thể gia tăng hoặc giảm đi tần số lấy mẫu. Ví dụ, ta có thể giảm tần số của bộ dữ liệu hiện tại từ ngày sang tháng. Điều này đồng nghĩa với việc bộ dữ liệu mới của chúng ta sẽ có ít mẫu dữ liệu hơn bản gốc.

Resampling thường hữu dụng với time series cho lower hoặc higher frequency. Resampling cho lower frequency (downsampling) thường liên quan tới hoạt động tổng hợp, ví dụ mức doanh thu trong tháng từ dữ liệu ngày. Resampling cho higher frequency (upsampling) ít phổ biến hơn, thường dùng trong việc nội suy. Ở đây, ta thử áp dụng downsampling cho bộ dữ liệu hiện tại như sau:

```

1 # Specify the data columns we want to include (i.e. exclude Year,
   Month, Weekday Name)
2 # Resample to weekly frequency, aggregating with mean
3 data_columns = ["Consumption", "Wind", "Solar", "Wind+Solar"]
4 opsd_weekly_mean = opsd_daily[data_columns].resample("W").mean()
5 opsd_weekly_mean.head(3)

```

Ở đoạn code trên, ta downsampling từ tần số theo ngày sang tuần. Giá trị của các cột lúc này sẽ là trung bình của 7 ngày trong tuần.

Date	Consumption	Wind	Solar	Wind+Solar
2006-01-01	1069.184000	NaN	NaN	NaN
2006-01-08	1381.300143	NaN	NaN	NaN
2006-01-15	1486.730286	NaN	NaN	NaN

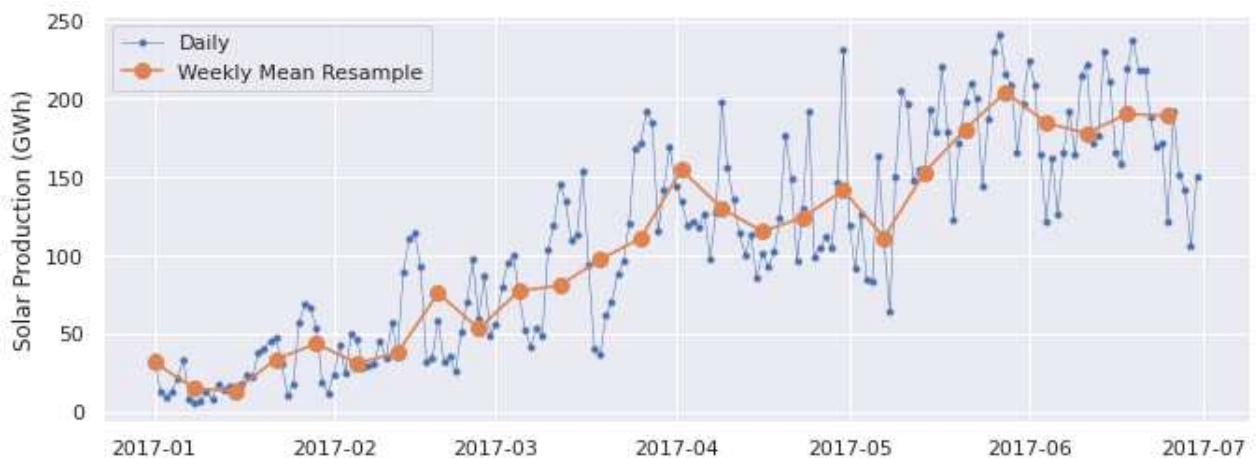
Hình 11.37: Sử dụng kỹ thuật Resampling để đổi tần số lấy mẫu của bộ dữ liệu từ ngày sang tuần

Đĩ nhiên, khi ta downsample bộ dữ liệu, số lượng mẫu dữ liệu của bảng dữ liệu mới sẽ ít hơn so với bảng gốc và ít hơn $1/7$ lần. Có thể kiểm tra bằng cách dùng thuộc tính shape của DataFrame:

```
1 print(opsd_daily.shape[0])
2 print(opsd_weekly_mean.shape[0])
```

Ta visualize daily và weekly time series của Solar trong 6 tháng như sau:

```
1 # Start and end of the date range to extract
2 start, end = "2017-01", "2017-06"
3 # Plot daily and weekly resampled time series together
4 fig, ax = plt.subplots()
5 ax.plot(opsd_daily.loc[start:end, "Solar"],
6 marker=".", linestyle="-", linewidth=0.5, label="Daily")
7 ax.plot(opsd_weekly_mean.loc[start:end, "Solar"],
8 marker="o", markersize=8, linestyle="-", label="Weekly Mean Resample
9 ")
10 ax.set_ylabel("Solar Production (GWh)")
11 ax.legend()
12 plt.show()
```



Hình 11.38: Đồ thị Time series của Solar theo ngày và theo tuần

Lưu ý rằng bảng dữ liệu gốc của chúng ta có tồn một số giá trị null. Vì vậy để đảm bảo toàn bộ các mẫu có giá trị, ta cài đặt tham số **min_count** vào để xử lý vấn đề này. Ví dụ, ta resampling bộ dữ liệu thành theo năm, để đảm bảo các ngày trong năm đều tồn tại giá trị non-null, ta có thể cài đặt **min_count=360** (các bạn có thể chọn **min_count** bằng một giá trị khác tùy vào quan sát cá nhân):

```

1 # Compute the annual sums, setting the value to NaN for any year
   which has
2 # fewer than 360 days of data
3 opsd_annual = opsd_daily[data_columns].resample("YE").sum(min_count=
   360)
4 # The default index of the resampled DataFrame is the last day of
   each year,
5 # ("2006-12-31", "2007-12-31", etc.) so to make life easier, set the
   index
6 # to the year component
7 opsd_annual = opsd_annual.set_index(opsd_annual.index.year)
8 opsd_annual.index.name = "Year"
9 # Compute the ratio of Wind+Solar to Consumption
10 opsd_annual["Wind+Solar/Consumption"] = opsd_annual["Wind+Solar"] /
   opsd_annual["Consumption"]
11 opsd_annual.tail(3)

```

	Consumption	Wind	Solar	Wind+Solar	Wind+Solar/Consumption
Year					
2015	505264.56300	77468.994	34907.138	112376.132	0.222410
2016	505927.35400	77008.126	34562.824	111570.950	0.220528
2017	504736.36939	102667.365	35882.643	138550.008	0.274500

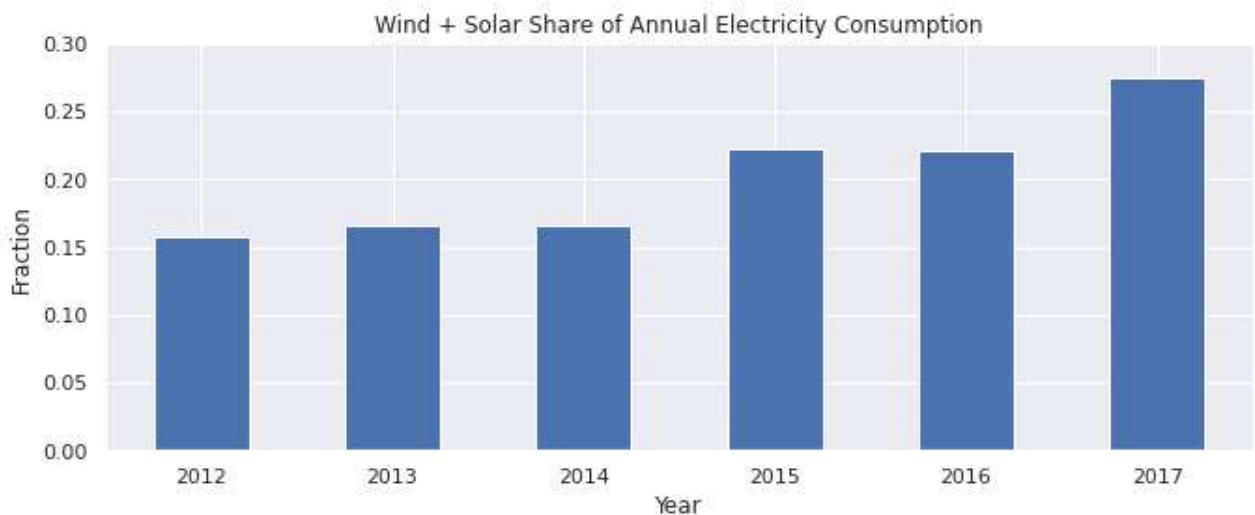
Hình 11.39: Annual resampling với bộ dữ liệu hiện tại

Ta có thể vẽ biểu đồ hiển thị sản lượng sản xuất năng lượng gió và mặt trời đóng góp vào mức độ tiêu thụ điện năng kể từ năm 2012 như sau:

```

1 # Plot from 2012 onwards, because there is no solar production data
   in earlier years
2 ax = opsd_annual.loc[2012:, "Wind+Solar/Consumption"].plot.bar(color
   ="CO")
3 ax.set_ylabel("Fraction")
4 ax.set_ylim(0, 0.3)
5 ax.set_title("Wind + Solar Share of Annual Electricity Consumption")
6 plt.xticks(rotation=0)
7 plt.show()

```



Hình 11.40: Biểu đồ cột biểu thị **Solar + Wind** đóng góp vào mức tiêu thụ điện năng

11.4.7 Rolling windows

Rolling window cũng là một hoạt động chuyển thông tin quan trọng trong dữ liệu time series. Giống downsampling, rolling windows chia dữ liệu thành các time windows (các khoảng thời gian như tuần, tháng... được trượt trên các mẫu dữ liệu theo ngày) và dữ liệu trong mỗi window đó được tổng hợp với hàm mean(), median(), sum(),... Tuy nhiên, không giống như downsampling, khi mà dữ liệu không overlap nhau và output luôn có tần số thấp hơn input, rolling windows overlap và gom thành những dữ liệu có cùng tần số, vì thế time series được chuyển có cùng tần số với time series gốc. Ta ví dụ với rolling trong 7 ngày:

```

1 # Compute the centered 7-day rolling mean
2 opsd_7d = opsd_daily[data_columns].rolling(7, center=True).mean()
3 opsd_7d.head(10)

```

Date	Consumption	Wind	Solar	Wind+Solar
2006-01-01	NaN	NaN	NaN	NaN
2006-01-02	NaN	NaN	NaN	NaN
2006-01-03	NaN	NaN	NaN	NaN
2006-01-04	1361.471429	NaN	NaN	NaN
2006-01-05	1381.300143	NaN	NaN	NaN
2006-01-06	1402.557571	NaN	NaN	NaN
2006-01-07	1421.754429	NaN	NaN	NaN
2006-01-08	1438.891429	NaN	NaN	NaN

Hình 11.41: Rolling windows với chu kỳ 7 ngày

Ở đây, 2006-01-01 đến 2006-01-07 được đánh nhãn là 2006-01-04, 2006-01-02 đến 2006-01-08 được đánh nhãn là 2006-01-05, tương tự cho các dòng khác.

11.4.8 Trends

Là một đặc trưng chỉ xu hướng của dữ liệu, có thể tăng hoặc giảm đi trong một khoảng thời gian dài. Với kỹ thuật rolling windows, ta có thể dễ dàng trực quan hóa trends của bộ dữ liệu, tại các time scales khác nhau. Ví dụ, ta tính 365-day rolling mean:

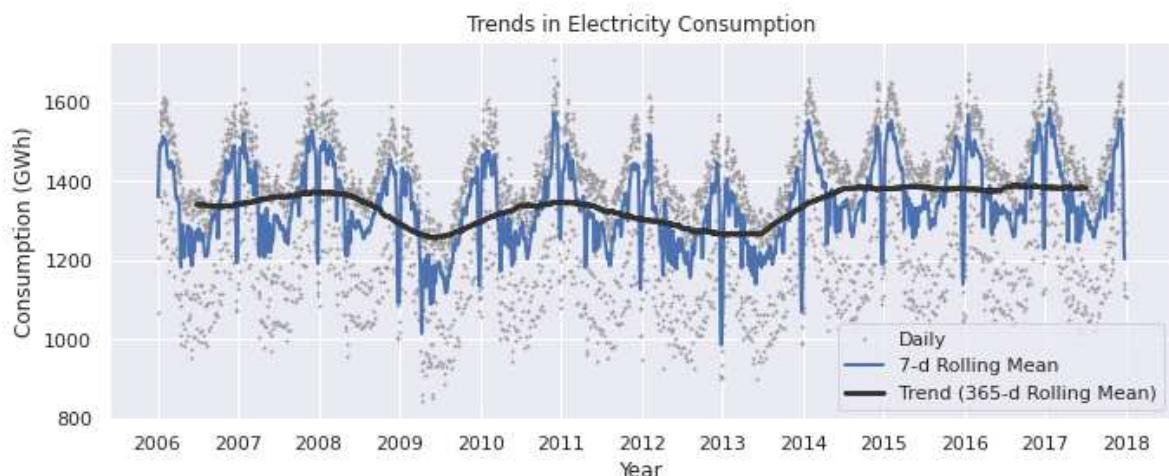
```

1 import matplotlib.dates as mdates
2
3 # The min_periods=360 argument accounts for a few isolated missing
4 # days in the
5 # wind and solar production time series
6 opsd_365d = opsd_daily[data_columns].rolling(
7     window=365,
8     center=True,
9     min_periods=360
10).mean()
11
12 # Plot daily, 7-day rolling mean, and 365-day rolling mean time
# series
13 fig, ax = plt.subplots()
```

```

13 ax.plot(opsd_daily["Consumption"], marker=".", markersize=2, color="0.6",
14         linestyle="None", label="Daily")
15 ax.plot(opsd_7d["Consumption"], linewidth=2, label="7-d Rolling Mean")
16 ax.plot(opsd_365d["Consumption"], color="0.2", linewidth=3,
17         label="Trend (365-d Rolling Mean)")
18 # Set x-ticks to yearly interval and add legend and labels
19 ax.xaxis.set_major_locator(mdates.YearLocator())
20 ax.legend()
21 ax.set_xlabel("Year")
22 ax.set_ylabel("Consumption (GWh)")
23 ax.set_title("Trends in Electricity Consumption")
24 plt.show()

```



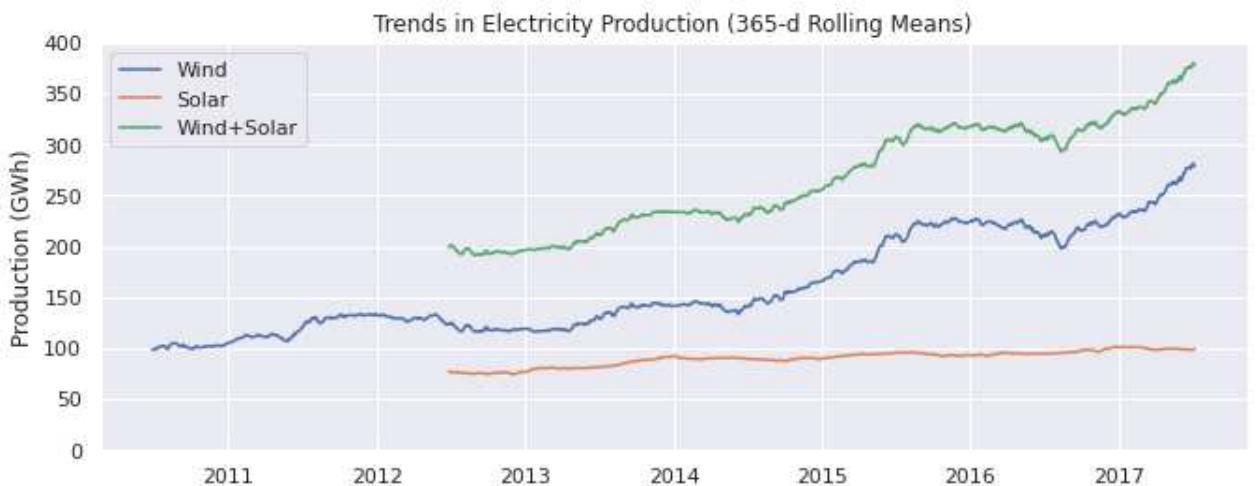
Hình 11.42: Xu hướng tiêu thụ điện, tuần và năm, tăng mạnh vào cuối năm

```

1 # Plot 365-day rolling mean time series of wind and solar power
2 fig, ax = plt.subplots()
3 for nm in ["Wind", "Solar", "Wind+Solar"]:
4     ax.plot(opsd_365d[nm], label=nm)
5     # Set x-ticks to yearly interval, adjust y-axis limits, add
      # legend and labels
6     ax.xaxis.set_major_locator(mdates.YearLocator())

```

```
7     ax.set_ylim(0, 400)
8     ax.legend()
9     ax.set_ylabel("Production (GWh)")
10    ax.set_title("Trends in Electricity Production (365-d Rolling
11    Means)")
12    plt.show()
```



Hình 11.43: Xu hướng sản xuất năng lượng điện gió và mặt trời có xu hướng tăng qua hằng năm, đặc biệt là năng lượng gió

Như vậy với một số bước thực hiện trên, ta đã xem qua cách sắp xếp, phân tích và trực quan hóa dữ liệu time series data trong pandas, dùng các kỹ thuật như time-based indexing, resampling, rolling windows. Áp dụng kỹ thuật này vào bộ dataset OPSD, thu được các thông tin chi tiết về thời điểm, các kỳ, và xu hướng trong sản xuất và tiêu thụ điện.

11.5 Câu hỏi trắc nghiệm

1. Data Analysis là gì?
 - (A) Quá trình thu thập dữ liệu.
 - (B) Quá trình tìm kiếm và khai thác dữ liệu.
 - (C) Quá trình xử lý dữ liệu.
 - (D) Các phương án trên đều đúng.
2. Cấu trúc dữ liệu nào sau đây không thuộc pandas:
 - (A) Series
 - (B) DataFrame
 - (C) Panel
 - (D) Tensor
3. Ý nghĩa của phương thức head() đối với bảng dữ liệu trong pandas là:
 - (A) Hiển thị các hàng cuối cùng
 - (B) Hiển thị các hàng đầu tiên
 - (C) Hiển thị ngẫu nhiên một số hàng
 - (D) Hiển thị tất cả các hàng
4. Ý nghĩa của phương thức describe() đối với bảng dữ liệu trong pandas là:
 - (A) Bảng thống kê của các cột dữ liệu string
 - (B) Bảng thống kê của các cột dữ liệu số
 - (C) Bảng thống kê của các cột dữ liệu list
 - (D) Bảng thống kê của các cột dữ liệu dict
5. Phương thức nào sau đây được dùng để đọc một file .csv từ bộ nhớ trong pandas?
 - (A) pd.load_csv()
 - (B) pd.read_csv()

- (C) pd.read_file()
(D) pd.load_file()
6. Ý nghĩa của phương thức groupby() đối với bảng dữ liệu trong pandas là:
(A) Lọc các hàng theo điều kiện
(B) Tổng hợp thống kê các cột dữ liệu
(C) Nối các bảng dữ liệu
(D) Gom nhóm dữ liệu theo giá trị của một hoặc nhiều cột
7. Phương thức nào sau đây dùng để kiểm tra các giá trị NaN có trong bảng dữ liệu?
(A) df.isna()
(B) df.notna()
(C) df.notnull()
(D) df.tail()
8. Phương thức nào sau đây có thể được dùng để bỏ đi một hàng có giá trị null trong bảng dữ liệu?
(A) df.drop_null()
(B) df.drop_missing()
(C) df.dropna()
(D) df.remove_null()
9. Phương thức nào sau đây trong pandas được dùng để fill các giá trị bị mất trong bảng dữ liệu sử dụng kỹ thuật forward filling?
(A) fillna(method="bfill")
(B) fillna(method="pad")
(C) fillna(method="ffill")
(D) fillna(method="forward")
10. Phương thức nào sau đây trong pandas được dùng để resample dữ liệu?

- (A) resample()
 (B) downsample()
 (C) reduce()
 (D) shrink()
11. Phương thức nào sau đây trong pandas được dùng để tính rolling windows?
 (A) rolling()
 (B) mean()
 (C) average()
 (D) smooth()
12. Phương thức nào sau đây trong pandas được dùng thực thi một hàm bất kì vào tất cả phần tử trong một Series?
 (A) pd.Series.transform()
 (B) pd.Series.applymap()
 (C) pd.Series.map()
 (D) pd.Series.apply()
13. Phương thức nào sau đây có thể được dùng để lấy toàn bộ một cột sử dụng tên của nó từ bảng dữ liệu?
 (A) df[col]
 (B) df.loc[col]
 (C) df.ix[col]
 (D) df.iloc[col]

Xem qua bảng dữ liệu sau đây (df) và trả lời các câu hỏi dưới đây:

Date	Open	High	Low	Close	Volume	Adj Close
6/29/2010	19.000000	25.000000	17.540001	23.889999	18766300	23.889999
6/30/2010	25.790001	30.420000	23.299999	23.830000	17187100	23.830000
7/1/2010	25.000000	25.920000	20.270000	21.959999	8218800	21.959999
7/2/2010	23.000000	23.100000	18.709999	19.200001	5139800	19.200001
7/6/2010	20.000000	20.000000	15.830000	16.110001	6866900	16.110001

14. Dòng lệnh nào sau đây dùng để chọn các hàng có giá trị ‘Close’ lớn hơn 25?
- (A) df[df["Close"] > 25]
 - (B) df["Close"] > 25
 - (C) df.iloc[df["Close"] > 25]
 - (D) df.Close(25)
15. Dòng lệnh nào sau đây dùng để chọn các hàng có giá trị ‘Volume’ nhỏ hơn hoặc bằng 1000000?
- (A) df.iloc[df["Volume"] <= 1000000]
 - (B) df[df["Volume"] <= 1000000]
 - (C) df.Volume <= "1000000"
 - (D) df["Volume"] <= 1000000
16. Dòng lệnh nào sau đây dùng để chọn các hàng có giá trị ‘High’ nhỏ hơn hoặc bằng ‘Low’?
- (A) df.iloc[df["High"] <= df["Low"]]
 - (B) df[df["High"] <= df["Low"]]
 - (C) df[df.High < "Low"]
 - (D) df["High"] <= df["Low"]
17. Dòng lệnh nào sau đây dùng để tìm giá trị trung bình của cột ‘Close’?
- (A) df.mean()
 - (B) df["Close"].mean
 - (C) df["Close"].sum()
 - (D) df["Close"].mean()

- 1. Hint:** Các file code gợi ý và bộ dữ liệu được lưu trong thư mục có thể được tải [tại đây](#).
- 2. Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

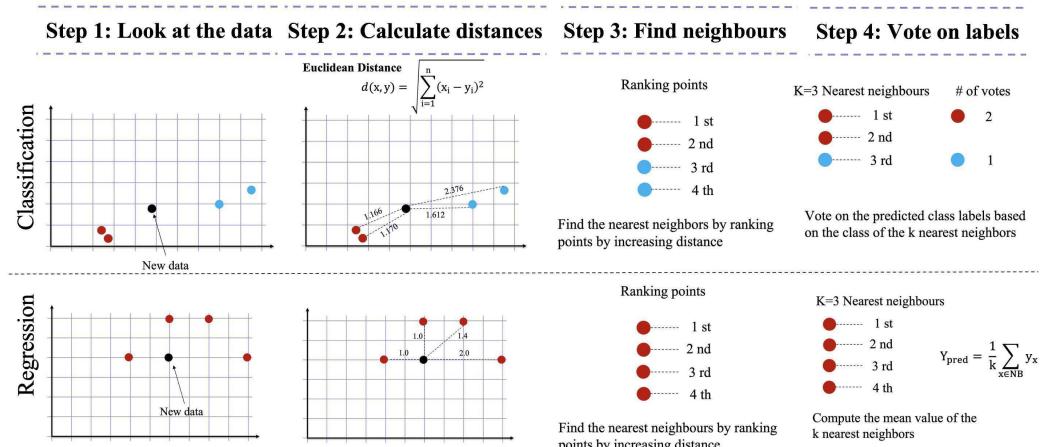
3. Rubric:

Mục	Kiến Thức	Đánh Giá
1	- Khái niệm về Data Analysis	- Hiểu được định nghĩa về Data Analysis
2	- Kiến thức về các hàm, thuộc tính cơ bản của pandas.	- Hiểu được định nghĩa của các hàm read_csv(), describe(), head(), cấu trúc dữ liệu trong pandas
3	- Kiến thức về khái niệm và hàm groupby.	- Hiểu được định nghĩa về groupby.
4	- Khái niệm về missing data. - Các hàm về xử lý, tương tác với missing data trong pandas.	- Hiểu được khái niệm về missing data. - Hiểu được cách kiểm tra các missing data có trong bộ dữ liệu. - Hiểu được cách xử lý missing data bằng cách filling hoặc dropping.
5	- Kiến thức về resampling và rolling windows.	- Khả năng thực hiện resampling và tạo rolling windows trong pandas.
6	- Kiến thức về thực thi hàm vào từng phần tử trong bảng dữ liệu trong pandas.	- Biết sử dụng hàm apply() trong pandas để thực thi một hàm bất kì vào trong bảng dữ liệu trong pandas.
7	- Kiến thức về slicing và indexing bảng dữ liệu trong pandas.	- Hiểu rõ cách thực hiện slicing và indexing trong pandas.

Chương 12

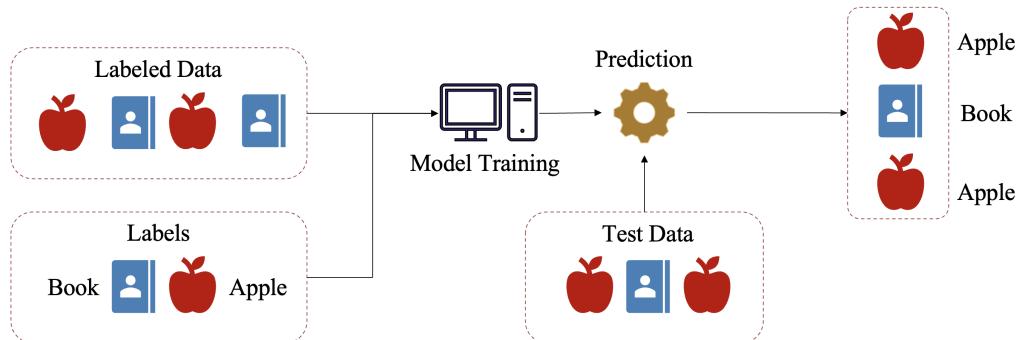
K-Nearest Neighbors và K-Means

12.1 Giới thiệu



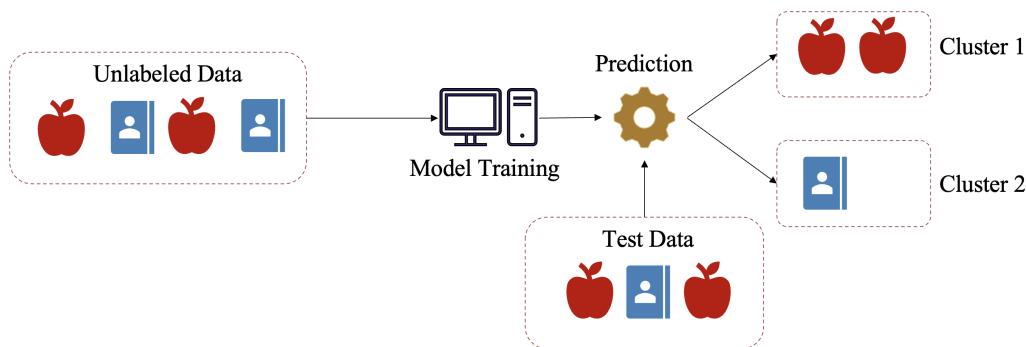
Hình 12.1: Chi tiết các bước tính toán với thuật toán KNN.

Trong lĩnh vực học máy, học có giám sát và học không giám sát là hai phương pháp quan trọng được sử dụng để giải quyết các loại bài toán khác nhau. Chúng ta sẽ tìm hiểu 2 thuật toán cơ bản: KNN - K Láng giềng gần nhất (K-Nearest Neighbors) thuộc nhóm học có giám sát và phân cụm K-Means thuộc nhóm học không giám sát.



Hình 12.2: Supervised Learning.

Học có giám sát là phương pháp trong đó dữ liệu huấn luyện đã được gán nhãn sẵn. Thuật toán sẽ học từ các cặp dữ liệu đầu vào (features) và đầu ra (labels) để dự đoán nhãn cho dữ liệu mới. Mục tiêu của học có giám sát là xây dựng được một mô hình có khả năng khái quát hóa tốt, giúp dự đoán chính xác trong các tình huống chưa gặp. Các bài toán phổ biến của học có giám sát bao gồm phân loại (classification) và hồi quy (regression). Ví dụ, hệ thống có thể dự đoán giá nhà dựa trên diện tích và vị trí, hoặc phân loại email thành thư rác và thư hợp lệ.



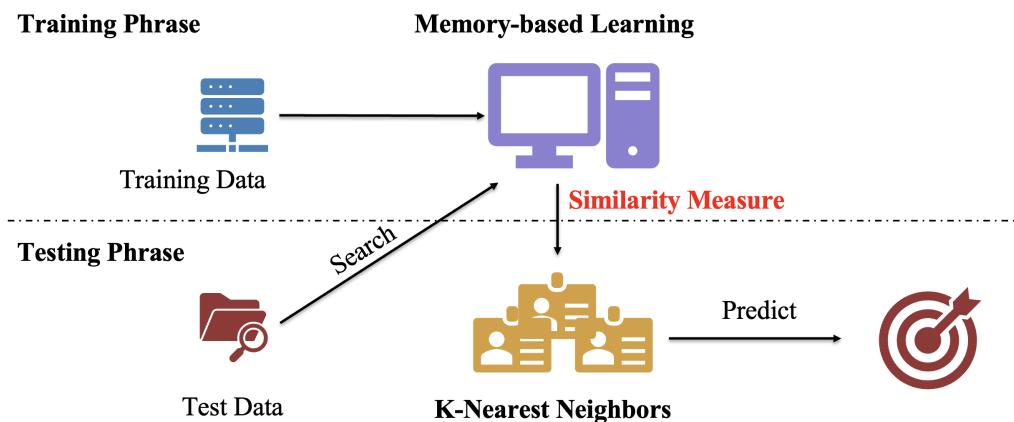
Hình 12.3: Unsupervised Learning.

Ngược lại, học không giám sát được áp dụng khi dữ liệu không có nhãn. Thay vì dự đoán, thuật toán sẽ tìm kiếm các mẫu ẩn hoặc cấu trúc tự nhiên bên trong dữ liệu. Các bài toán điển hình bao gồm phân cụm (clustering) và giảm chiều (dimensionality reduction). Chẳng hạn, doanh nghiệp có thể sử dụng thuật toán K-Means để phân nhóm khách hàng dựa trên hành vi mua sắm, từ đó xây dựng các chiến dịch tiếp thị phù hợp.

Trong phần này, chúng ta sẽ tìm hiểu 2 thuật toán cơ bản: KNN - K Láng giềng gần nhất (K-Nearest Neighbors) thuộc nhóm học có giám sát và phân cụm K-Means thuộc nhóm học không giám sát.

12.2 Thuật toán KNN

K-Nearest Neighbors (KNN) là một trong những thuật toán học máy có giám sát cơ bản. KNN còn được gọi là Lazy Learning, Memory-Based Learning,... Với bước huấn luyện mô hình, chỉ đơn giản là lưu trữ lại giá trị của dữ liệu huấn luyện, vì vậy KNN là phương pháp học máy không tham số (Non-Parametric). Ở bước dự đoán, mô hình sẽ sử dụng các độ đo khoảng cách để tìm các hàng xóm lân cận.



Hình 12.4: Thuật toán KNN.

Một số độ đo thường được sử dụng trong KNN như:

1. Euclidean
2. Chebyshev
3. Manhattan
4. Minkowski

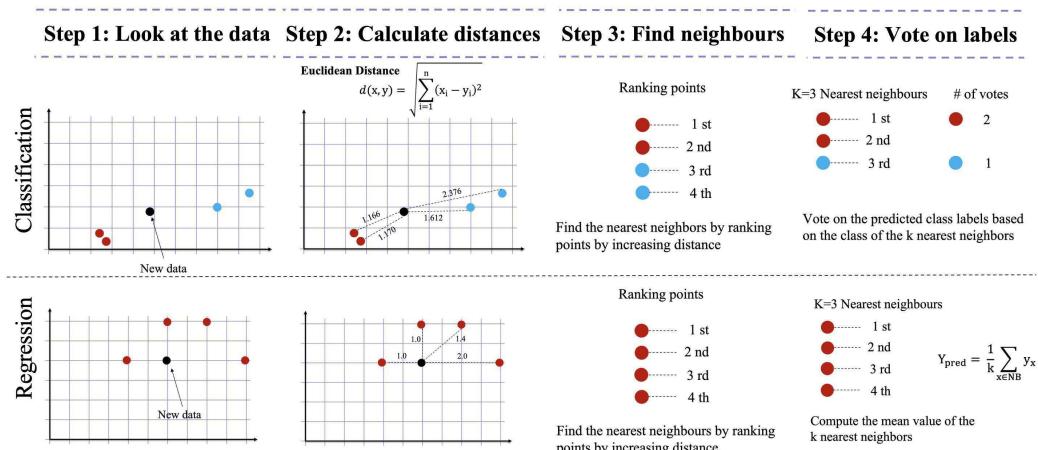
Các độ đo này tính toán dữ liệu dự đoán với các điểm dữ liệu khác được lưu trữ trong mô hình huấn luyện. Từ đó xếp hạng và tìm ra K điểm dữ liệu huấn luyện có kết quả gần với dữ liệu dự đoán nhất. Cuối cùng dựa vào phương pháp biểu quyết của các dữ liệu hàng xóm trong tập huấn luyện để đưa ra kết quả dự đoán.

Các bước hoạt động của KNN: KNN hoạt động theo 4 bước đơn giản, áp dụng cho cả bài toán phân loại và hồi quy:

- Bắt đầu với một tập dữ liệu đã được gán nhãn và một điểm dữ liệu mới cần dự đoán.
- Tính khoảng cách (Calculate distances): Tính khoảng cách từ điểm dữ liệu mới đến tất cả các điểm trong tập huấn luyện. Khoảng cách Euclidean là một trong những độ đo phổ biến nhất.
- Tìm K láng giềng (Find neighbours): Sắp xếp các khoảng cách theo thứ tự tăng dần và chọn ra K điểm dữ liệu gần nhất với điểm mới. "K" là một số nguyên dương do người dùng xác định.
- Đưa ra dự đoán (Vote on labels/Compute mean):

Với bài toán Phân loại (Classification): Điểm dữ liệu mới sẽ được gán nhãn phổ biến nhất trong số K láng giềng (phương pháp biểu quyết đa số).

Với bài toán Hồi quy (Regression): Giá trị dự đoán cho điểm mới sẽ là trung bình cộng giá trị của K láng giềng.

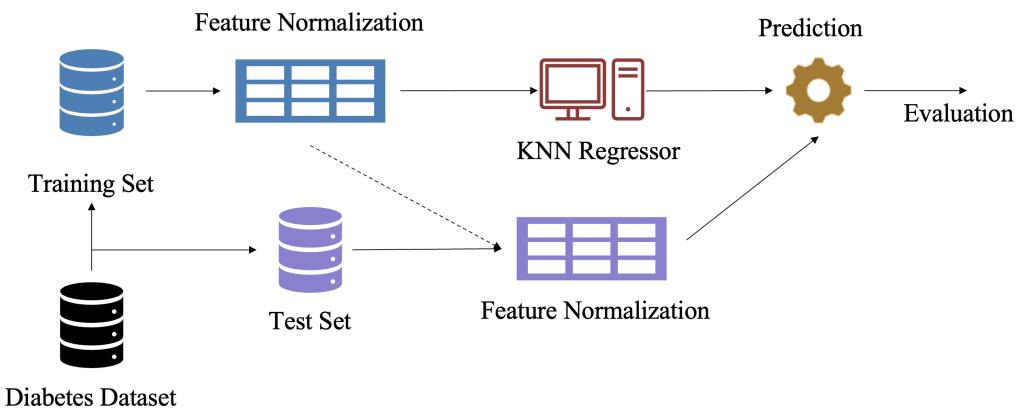


Hình 12.5: Chi tiết các bước tính toán với thuật toán KNN.

Trong phần này, ta sẽ áp dụng thuật toán KNN cho bài toán Regression (Trên bộ dữ liệu Diabetes) và hai bài toán Classification (Trên bộ dữ liệu IRIS và IMDB).

12.2.1 KNN for Diabetes Regression

Mục tiêu là dự đoán một chỉ số định lượng về sự tiến triển của bệnh tiểu đường dựa trên các đặc trưng y tế.



Hình 12.6: Quy trình áp dụng thuật toán KNN cho bài toán Regression trên bộ dữ liệu Diabetes.

Quá trình thực thi theo trình tự như sau, bổ sung thêm phần code để hoàn thiện mô hình:

- Thiết lập thư viện Đầu tiên, chúng ta cần nhập các thư viện cần thiết từ scikit-learn, numpy để xử lý dữ liệu và mô hình.

```

1 import numpy as np
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4 from sklearn.model_selection import train_test_split
5 from sklearn.metrics import mean_squared_error, r2_score
6 from sklearn.preprocessing import StandardScaler
7 from sklearn.neighbors import KNeighborsRegressor
  
```

- Tải về bộ dữ liệu

Tải bộ dữ liệu Diabetes có sẵn trong scikit-learn và chia nó thành hai phần: 80% cho huấn luyện (training) và 20% cho kiểm thử (testing). Việc này giúp đánh giá mô hình trên dữ liệu mà nó chưa từng thấy.

```

1 # Load the diabetes dataset
2 diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=
3                                     True)
4
5 # Split train:test = 8:2
6 X_train, X_test, y_train, y_test = train_test_split(
7     # You code here
8 )

```

3. Chuẩn hoá dữ liệu

Các đặc trưng trong dữ liệu có thể có các thang đo rất khác nhau. Để đảm bảo tất cả các đặc trưng đóng góp một cách công bằng vào việc tính toán khoảng cách, chúng ta cần chuẩn hóa chúng. StandardScaler sẽ biến đổi dữ liệu sao cho chúng có trung bình là 0 và độ lệch chuẩn là 1.

```

1 # Scale the features using StandardScaler
2 scaler = StandardScaler()
3 X_train = scaler.fit_transform(X_train)
4 X_test = # You code here

```

4. Huấn luyện mô hình

Bây giờ, chúng ta tạo một mô hình KNeighborsRegressor với n_neighbors=5, huấn luyện nó trên tập X_train và y_train, sau đó dự đoán trên X_test.

```

1 # Training
2 knn_regressor = KNeighborsRegressor(n_neighbors=5)
3 knn_regressor.fit(X_train, y_train)
4
5 # Testing
6 y_pred = # You code here
7 mean_squared_error(y_test, y_pred)

```

5. So sánh các giá trị k

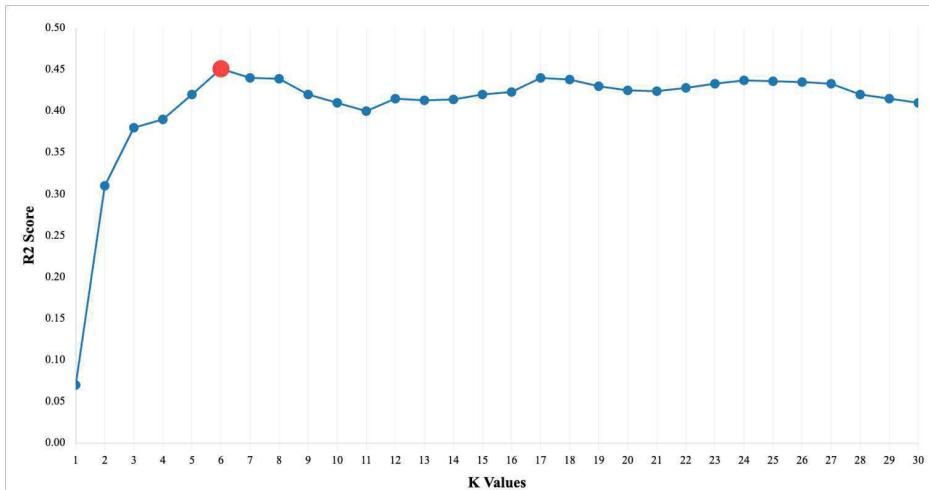
Việc chọn giá trị K tốt nhất là rất quan trọng. Chúng ta sẽ thử các giá trị K từ 1 đến 30 và xem giá trị nào cho điểm R2 Score cao nhất. R2 Score đo lường mức độ phù hợp của mô hình với dữ liệu, giá trị càng gần 1 càng tốt.

```

1 k_values = [i for i in range (1,31)]
2 scores = []
3
4 for k in k_values:
5     knn_regressor = # You code here
6     knn_regressor.fit(X_train, y_train)
7     y_pred = # You code here
8     score = r2_score(y_test, y_pred)
9     scores.append(np.mean(score))

```

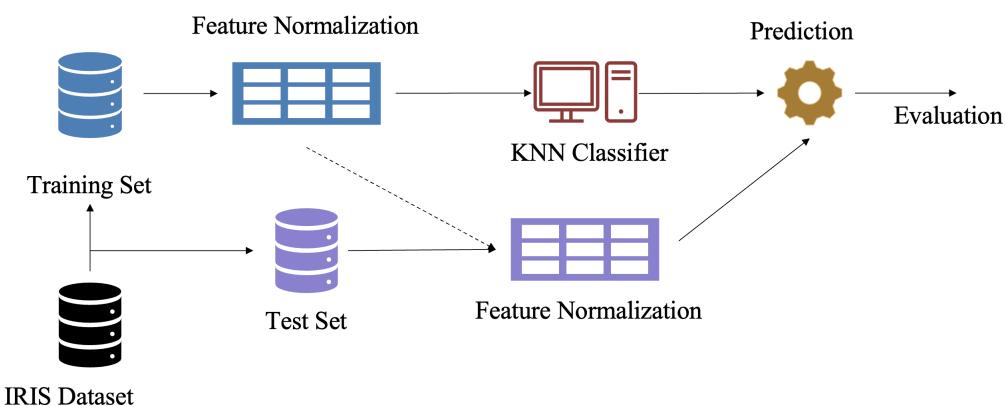
Kết quả cho thấy giá trị k=6, R2 Score đạt giá trị cao nhất trước khi bắt đầu giảm nhẹ.



Hình 12.7: Kết quả so sánh các giá trị k khác nhau trong mô hình KNN.

12.2.2 KNN for IRIS Classification

Bây giờ, chúng ta sẽ áp dụng KNN để phân loại 3 loài hoa Iris dựa trên các đặc điểm của chúng.



Hình 12.8: Quy trình áp dụng thuật toán KNN cho bài toán Classification trên bộ dữ liệu phân loại hoa IRIS.

Quá trình thực thi theo trình tự như sau, bổ sung thêm phần code để hoàn thiện mô hình:

- Thiết lập thư viện

Cài đặt các thư viện cần thiết

```

1 import numpy as np
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4 from sklearn import datasets
5 from sklearn.model_selection import train_test_split
6 from sklearn.metrics import accuracy_score
7 from sklearn.preprocessing import StandardScaler
8 from sklearn.neighbors import KNeighborsClassifier
  
```

- Tải về bộ dữ liệu

Tải về bộ dữ liệu IRIS có sẵn trong sklearn và chia tập train và tập test.

```

1 # Load the diabetes dataset
2 iris_X, iris_y = datasets.load_iris(return_X_y=True)
3
  
```

```

4 # Split train:test = 8:2
5 X_train, X_test, y_train, y_test = train_test_split(
6     iris_X, iris_y, test_size=0.2, random_state=42
7 )

```

3. Chuẩn hoá dữ liệu

Tương tự, chúng ta sử dụng StandardScaler để chuẩn hoá dữ liệu.

```

1 # Scale the features using StandardScaler
2 scaler = StandardScaler()
3 X_train = # You code here
4 X_test = scaler.transform(X_test)

```

4. Huấn luyện và đánh giá mô hình

Chúng ta huấn luyện mô hình với k=3 và đánh giá bằng accuracy_score - tỉ lệ dự đoán đúng trên tổng số dự đoán.

```

1 knn_classifier = KNeighborsClassifier(n_neighbors=3)
2 knn_classifier.fit(X_train, y_train)
3
4 y_pred = knn_classifier.predict(X_test)
5 accuracy_score(y_test, y_pred)

```

5. So sánh các giá trị k

Tương tự, chúng ta kiểm tra các giá trị K khác nhau để tìm ra K tối ưu cho độ chính xác.

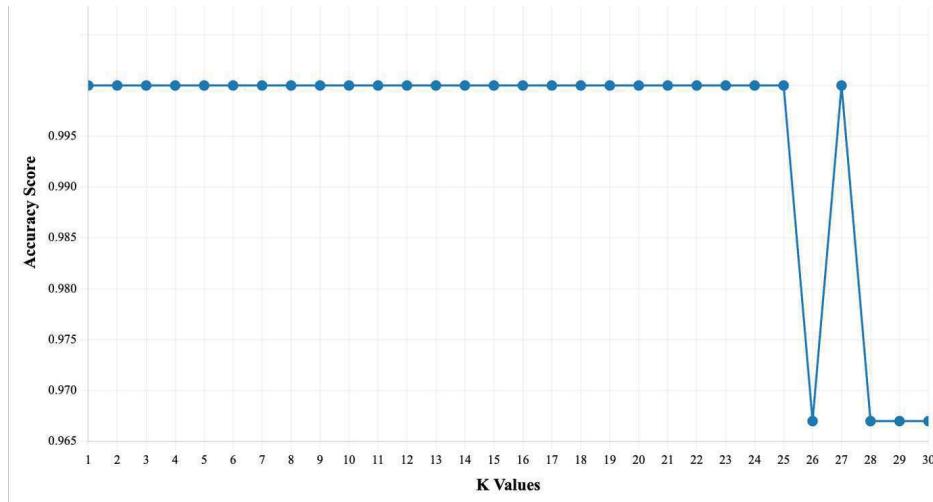
```

1 k_values = [i for i in range (1,31)]
2 scores = []
3
4 for k in k_values:
5     knn_classifier = # You code here
6     knn_classifier.fit(X_train, y_train)
7     y_pred = # You code here
8     score = accuracy_score(y_test, y_pred)

```

```
9 scores.append(np.mean(score))
```

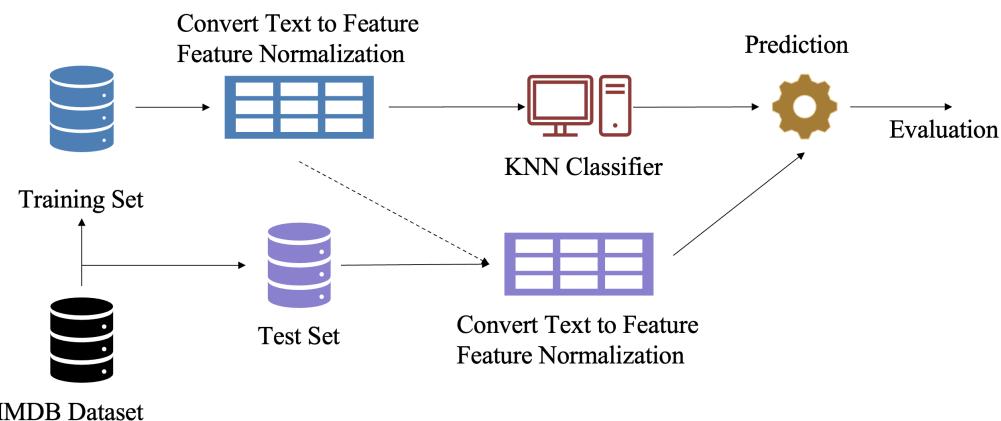
Biểu đồ cho thấy độ chính xác đạt 100% (1.0) với nhiều giá trị K, cho thấy đây là một bài toán tương đối dễ. Tuy nhiên, hiệu suất bắt đầu giảm ở các giá trị k lớn (từ 26 trở đi).



Hình 12.9: Kết quả so sánh các giá trị k khác nhau trong mô hình KNN.

12.2.3 KNN for IMDB Classification

Tiếp theo chúng ta sẽ áp dụng KNN cho bài toán phân loại văn bản trên bộ dữ liệu IMDB. Chúng ta sẽ sử dụng phương pháp Túi từ (Bag of Words - BoW) để biểu diễn mỗi đoạn văn bản (mỗi bài đánh giá) dưới dạng một vector, trong đó mỗi phần tử của vector tương ứng với số lần xuất hiện của một từ trong từ điển.



Hình 12.10: Quy trình áp dụng thuật toán KNN cho bài toán Classification trên bộ dữ liệu phân loại văn bản IMDB.

Quá trình thực thi theo trình tự như sau, bổ sung thêm phần code để hoàn thiện mô hình:

- Thiết lập thư viện

Chúng ta cần thêm các thư viện mới: datasets để tải bộ dữ liệu IMDB và CountVectorizer để thực hiện kỹ thuật Bag of Words.

```

1 !pip install datasets==4.0.0
2
3 import numpy as np
4 from datasets import load_dataset
5 from sklearn.neighbors import KNeighborsClassifier
6 from sklearn.preprocessing import StandardScaler
7 from sklearn.feature_extraction.text import CountVectorizer
8 from sklearn.metrics import accuracy_score, f1_score,
                           classification_report

```

- Tải về bộ dữ liệu

Sử dụng thư viện datasets để tải trực tiếp bộ dữ liệu IMDB, đã được chia sẵn thành tập huấn luyện và tập kiểm thử.

```

1 # Load the imdb dataset
2 imdb = load_dataset("imdb")
3 imdb_train, imdb_test = imdb["train"], imdb["test"]
4 len(imdb_train["text"]), len(imdb_test["text"])

```

3. Chuyển văn bản thành đặc trưng sử dụng phương pháp túi từ (BoW) CountVectorizer để biến đổi các bài đánh giá dạng văn bản thành các vector số.

- max_features=1000: Chúng ta chỉ giữ lại 1000 từ xuất hiện phổ biến nhất trong toàn bộ dữ liệu để giới hạn kích thước của vector, giúp mô hình chạy nhanh hơn và tránh nhiễu.
- fit_transform(): Học từ điển (tất cả các từ) từ imdb_train và biến đổi nó thành vector.
- transform(): Chỉ biến đổi imdb_test dựa trên từ điển đã học từ tập huấn luyện.

```

1 # Convert text to vector using Bow
2 vectorizer = CountVectorizer(max_features=1000)
3 X_train = # You code here
4 X_test = # You code here
5
6 y_train = np.array(imdb_train["label"])
7 y_test = np.array(imdb_test["label"])

```

4. Chuẩn hoá dữ liệu

Sử dụng StandardScaler để chuẩn hoá các vector.

```

1 # Scale the features using StandardScaler
2 scaler = StandardScaler()
3 X_train = scaler.fit_transform(X_train)
4 X_test = scaler.transform(X_test)

```

5. Huấn luyện và mô hình

Cuối cùng, chúng ta huấn luyện mô hình KNeighborsClassifier và đánh giá hiệu suất của nó bằng cả accuracy_score và f1_score. F1-score là một chỉ số hữu ích khi đánh giá các bài toán phân loại, vì nó là trung bình điều hòa của Precision và Recall.

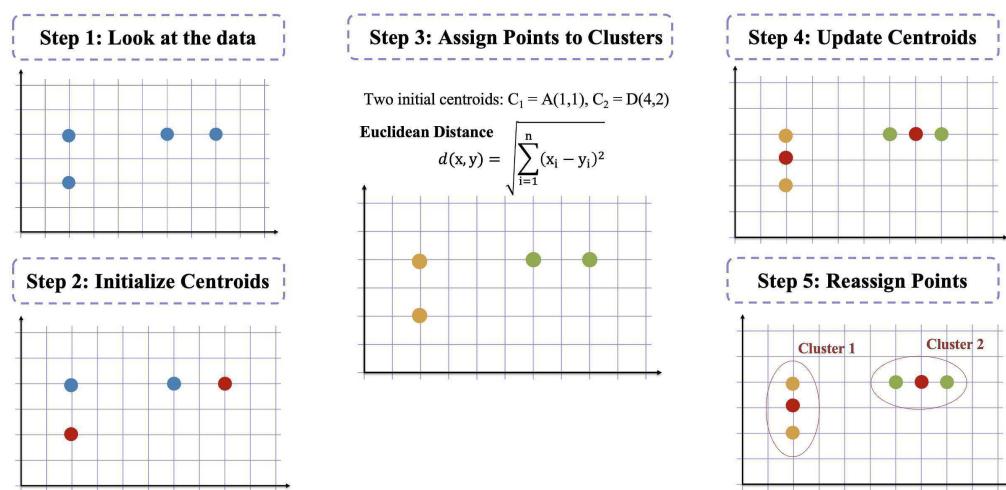
```
1 # Training
2 knn_classifier = KNeighborsClassifier(n_neighbors=3)
3 knn_classifier.fit(X_train, y_train)
4
5 # Prediction
6 y_pred = knn_classifier.predict(X_test)
7 accuracy_score(y_test, y_pred), f1_score(y_test, y_pred)
```

Kết quả huấn luyện đạt accuracy khoảng 63.43%, và F1 khoảng 66.36%.

12.3 Thuật toán phân cụm K-Means

Thuật toán **K-Means** là một thuật toán phân cụm phổ biến và được sử dụng rộng rãi trong các bài toán học máy (machine learning) và khai phá dữ liệu (data mining). Mục tiêu của K-Means là phân chia n điểm dữ liệu thành k cụm sao cho các điểm trong cùng một cụm có độ tương đồng cao nhất.

Quy trình cơ bản của K-Means bao gồm các bước sau:



Hình 12.11: Thuật toán K-Means Clustering.

- Khởi tạo:** Lựa chọn ngẫu nhiên k điểm từ tập dữ liệu làm các tâm cụm ban đầu.
- Gán nhãn:** Với mỗi điểm dữ liệu, tính khoảng cách từ điểm đó tới mỗi tâm cụm và gán nó vào cụm có tâm gần nhất. Khoảng cách thường được đo bằng khoảng cách Euclid, được tính như sau:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

trong đó, \mathbf{x} là điểm dữ liệu, \mathbf{y} là tâm cụm, và n là số chiều của dữ liệu.

- Cập nhật tâm cụm:** Sau khi đã gán nhãn cho tất cả các điểm, cập nhật lại vị trí của các tâm cụm bằng cách tính trung bình các điểm dữ

liệu trong mỗi cụm:

$$\mathbf{c}_j = \frac{1}{|\mathcal{C}_j|} \sum_{\mathbf{x}_i \in \mathcal{C}_j} \mathbf{x}_i$$

trong đó, \mathcal{C}_j là tập hợp các điểm dữ liệu thuộc cụm j .

4. **Lặp lại:** Lặp lại quá trình gán nhãn và cập nhật tâm cụm cho đến khi các tâm cụm không còn thay đổi đáng kể hoặc đạt đến số lần lặp tối đa.

Thuật toán K-Means có độ phức tạp tính toán là $O(n \times k \times t \times m)$, trong đó t là số lần lặp và m là số chiều của dữ liệu. K-Means thường được sử dụng vì tính đơn giản và hiệu quả của nó trong việc phân cụm dữ liệu lớn, mặc dù nó có thể không tìm được nghiệm tối ưu toàn cục (global optimization) do phụ thuộc vào việc khởi tạo các tâm cụm ban đầu.

Chúng ta sẽ sử dụng K-Means để phân nhóm khách hàng của một trung tâm thương mại dựa trên các thuộc tính như tuổi, thu nhập và điểm chi tiêu dựa trên bộ dữ liệu ‘Mall Customer’. Quy trình thực hiện gồm các bước như sau:

12.3.1 Cài đặt thư viện

Cài đặt các thư viện cần thiết.

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from sklearn.cluster import KMeans
6 from sklearn.preprocessing import LabelEncoder, MinMaxScaler
7 from sklearn.metrics import adjusted_rand_score, silhouette_score

```

12.3.2 Tải về bộ dữ liệu

Tải về bộ dữ liệu bằng cách chạy đoạn code sau, đọc dữ liệu sử dụng thư viện pandas.

```

1 # Download dataset
2 !gdown 16w9m7CK-vsTzrvw3XMR0dkWLias0xw68
3
4 df = pd.read_csv("Mall_Customers.csv")
5 df.drop("CustomerID", axis=1, inplace=True)
6 df.info()

```

12.3.3 Phân tích dữ liệu

Chúng ta thực hiện một số thống kê dữ liệu trên các tiêu chí.

```

1 import math
2
3 def plot_all_histograms(df, title_prefix=""):
4     num_cols = df.select_dtypes(include=[np.number]).columns
5     n_cols = 3
6     n_rows = math.ceil(len(num_cols) / n_cols)
7
8     plt.figure(figsize=(5 * n_cols, 4 * n_rows))
9
10    for i, col in enumerate(num_cols, 1):
11        plt.subplot(n_rows, n_cols, i)
12        sns.histplot(df[col], kde=True, bins=30)
13        plt.title(f"{title_prefix}{col}")
14        plt.xlabel("")
15        plt.ylabel("")
16
17    plt.tight_layout()
18    plt.show()
19
20 plot_all_histograms(df)

```

12.3.4 Chuẩn hoá dữ liệu

Dữ liệu bao gồm cả thuộc tính dạng chuỗi ("Genre"). Chúng ta mã hóa (Encode): Chuyển cột "Genre" (Giới tính) thành dạng số (ví dụ: 0 và 1) bằng LabelEncoder.

Sau đó, đưa tất cả các thuộc tính số về cùng một phạm vi (từ 0 đến 1)

bằng MinMaxScaler.

```

1 # Encode Genre
2 label_encoder = LabelEncoder()
3
4 df[ "Genre" ] = # You code here
5
6 scaler = MinMaxScaler()
7 df_scaled = scaler.fit_transform(df)

```

12.3.5 Huấn luyện và đánh giá mô hình

Không giống KNN, K-Means yêu cầu chúng ta phải tự xác định số cụm k. Hai phương pháp phổ biến để tìm k tối ưu là:

- Elbow Method (Phương pháp khuỷu tay): Vẽ biểu đồ giá trị Inertia (tổng bình phương khoảng cách từ các điểm đến tâm cụm của chúng) theo k. Điểm "khuỷu tay" (nơi đồ thị bắt đầu phẳng ra) thường là giá trị k tốt.
- Silhouette Score: Đo lường mức độ một điểm dữ liệu giống với cụm của chính nó so với các cụm khác. Giá trị càng gần +1, việc phân cụm càng tốt.

```

1 inertia = []
2 silhouette_scores = []
3
4 k_values = range(2, 11)
5 for k in k_values:
6     kmeans = # You code here
7     kmeans.fit(df_scaled)
8     inertia.append(kmeans.inertia_)
9     silhouette_scores.append(silhouette_score(df_scaled, kmeans.
10                             labels_))
11
12 # Vẽ Elbow Method
13 plt.plot(k_values, inertia, marker="o")
14 plt.xlabel("Số cụm (k)")
15 plt.ylabel("Inertia")
16 plt.title("Elbow Method")

```

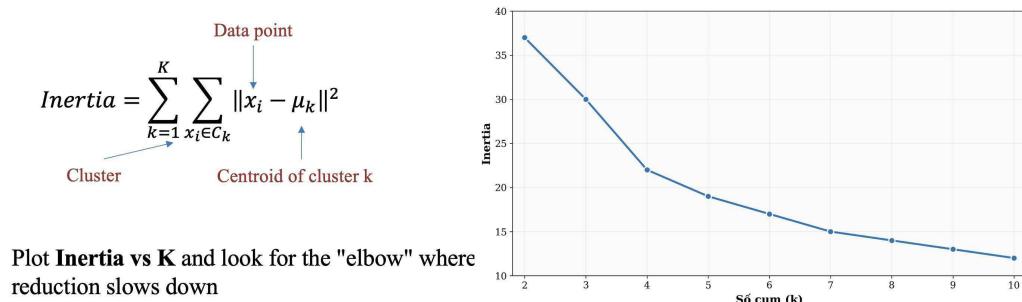
```

16 plt.show()
17
18 # Vẽ Silhouette Score
19 plt.plot(k_values, silhouette_scores, marker="o", color="green")
20 plt.xlabel("Số cụm (k)")
21 plt.ylabel("Silhouette Score")
22 plt.title("Silhouette Score cho từng k")
23 plt.show()

```

12.3.6 Tối ưu lựa chọn tham số cụm

Phương pháp Elbow để lựa chọn giá trị k được tính dựa vào khoảng cách từ điểm đến tâm cụm, được tính như hình sau:



Hình 12.12: Lựa chọn số cụm k trong thuật toán K-Means Clustering.

Dựa vào đây, chúng ta có thể lựa chọn giá trị k khoảng bằng 4, để huấn luyện lại mô hình và đánh giá mô hình.

```

1 optimal_k = k_values[np.argmax(silhouette_scores)]
2 print(f"Số cluster tối ưu theo Silhouette Score: {optimal_k}")
3 print(f"Silhouette Score cao nhất: {max(silhouette_scores):.3f}")
4
5 kmeans_final = KMeans(n_clusters=optimal_k, random_state=42, n_init=
                           10)
6 cluster_labels = kmeans_final.fit_predict(df_scaled)
7
8 silhouette_score(df_scaled, kmeans.labels_)

```

12.4 Câu hỏi trắc nghiệm

1. Hàm đánh giá nào trong các hàm sau đây không được sử dụng trong KNN?
 - (A) Manhattan
 - (B) Minkowski
 - (C) Tanimoto
 - (D) ROUGE
2. Cho hàm tính khoảng cách sau đây: $d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$ Hàm tính độ đo khoảng cách trên được gọi là:
 - (A) Manhattan
 - (B) Minkowski
 - (C) Euclidean
 - (D) Tanimoto
3. Độ đo nào không được sử dụng để đánh giá mô hình phân loại?
 - (a) F1-Score
 - (b) Accuracy
 - (c) Precision
 - (d) ROUGE
4. Phương pháp nào không được sử dụng để biểu diễn văn bản thành các giá trị vector là:
 - (A) Bag-of-Word
 - (B) TF-IDF
 - (C) One Hot Encoding
 - (D) F-Score

Chú ý: đáp án của câu hỏi trước sẽ giúp trả lời cho các câu hỏi sau trong các câu hỏi dựa vào bảng dưới đây. Dưới đây là một bộ dữ liệu nhỏ với 3 thuộc tính và 8 mẫu:

Feature 1	Feature 2	Feature 3
2.0	3.0	1.5
3.0	3.5	2.0
3.5	3.0	2.5
8.0	8.0	7.5
8.5	8.5	8.0
9.0	8.0	8.5
1.0	2.0	1.0
1.5	2.5	1.5

Bảng 12.1: Bộ dữ liệu nhỏ với 3 thuộc tính và 8 mẫu.

Giả sử bạn khởi tạo 2 centroids cho bộ dữ liệu trên. Hai điểm dữ liệu làm centroids ban đầu từ bảng trên: Cụm 1: (2.0, 3.0, 1.5) và Cụm 2: (1.0, 2.0, 1.0)

5. Tính khoảng cách Euclid giữa điểm dữ liệu (2.0, 3.0, 1.5) và centroid (8.0, 8.0, 7.5). Kết quả gần nhất với giá trị nào?
 - A 8.6
 - B 9.0
 - C 7.7
 - D 10.0
6. Với các centroids đã được khởi tạo, xác định cụm của điểm dữ liệu (3.0, 3.5, 2.0) dựa trên khoảng cách Euclid đến các centroids. Điểm này thuộc cụm nào?
 - A Cụm 1
 - B Cụm 2
7. Sau khi gán các điểm dữ liệu vào các cụm, điểm centroid mới cho cụm 1 (bao gồm các điểm dữ liệu thuộc cụm 1) sẽ là gì?
 - A Trung bình của các điểm dữ liệu (2.0, 3.0, 1.5), (3.0, 3.5, 2.0).
 - B Trung bình của các điểm dữ liệu (8.0, 8.0, 7.5), (8.5, 8.5, 8.0), (9.0, 8.0, 8.5).
 - C Trung bình của tất cả các điểm dữ liệu thuộc centroid 1.

- D** Trung bình của các điểm dữ liệu (1.0, 2.0, 1.0), (1.5, 2.5, 1.5).
8. Sau khi cập nhật các centroids, nếu chúng không thay đổi sau một lần lặp, thuật toán K-Means có dừng lại không?
- A** Có, thuật toán dừng lại khi centroids không thay đổi.
- B** Không, thuật toán sẽ tiếp tục chạy thêm một số lần lặp.
- C** Có, nhưng chỉ dừng lại nếu đạt số lần lặp tối đa.
- D** Không, thuật toán sẽ không bao giờ dừng lại.
9. Thuật toán K-Means thuộc loại thuật toán học máy nào?
- (A) Học có giám sát (Supervised Learning)
- (B) Học không giám sát (Unsupervised Learning)
- (C) Học tăng cường (Reinforcement Learning)
- (D) Học bán giám sát (Semi-supervised Learning)
10. Trong thuật toán K-Means, yếu tố nào sau đây ảnh hưởng trực tiếp đến việc hội tụ và chất lượng phân cụm?
- (A) Phương pháp khởi tạo các centroid ban đầu
- (B) Kích thước batch trong quá trình huấn luyện
- (C) Hàm đánh giá F1-Score
- (D) Số lượng epoch cố định

- 1. Hint:** Các file code gợi ý và dữ liệu được lưu trong thư mục có thể được tải [tại đây](#).
- 2. Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. Rubric:

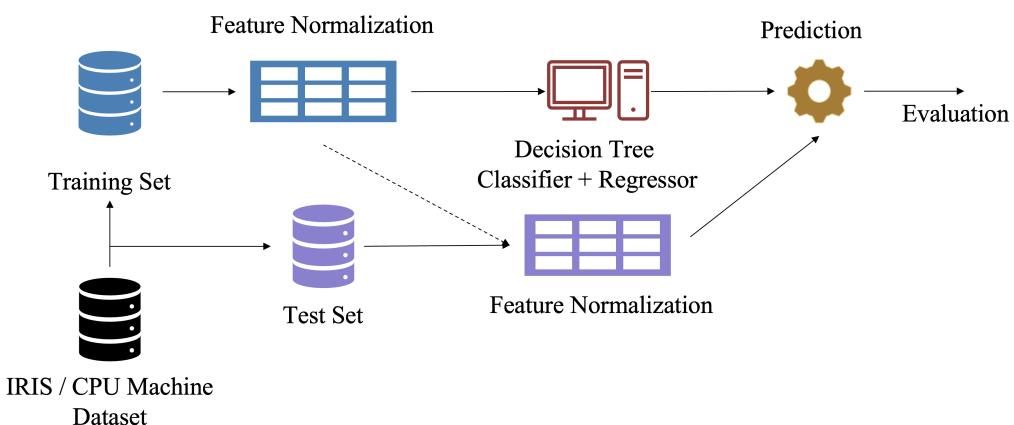
Mục	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Nắm vững kiến thức về K-Nearest Neighbors (KNN), phương pháp huấn luyện mô hình - Các độ đo thường được sử dụng trong KNN - Phương pháp lựa chọn giá trị K - Các thuật toán tìm kiếm được sử dụng trong KNN 	<ul style="list-style-type: none"> - Hoàn thiện code KNN cho bài toán phân loại hoa Iris và phân loại văn bản - Hoàn thiện code KNN cho bài toán dự đoán - Sử dụng thư viện sklearn cho mô hình KNN
2.	<ul style="list-style-type: none"> - Nắm vững kiến thức của k-means clustering - Hiểu đường luồng hoạt động của k-means được xây dựng từ cơ bản - Thực hiện xây dựng thuật toán bằng numpy 	<ul style="list-style-type: none"> - Có thể áp dụng phân cụm k-means với data - Có thể xây dựng thuật toán k-means bằng numpy

Chương 13

Decision tree cho bài toán phân loại và hồi quy

13.1 Giới thiệu

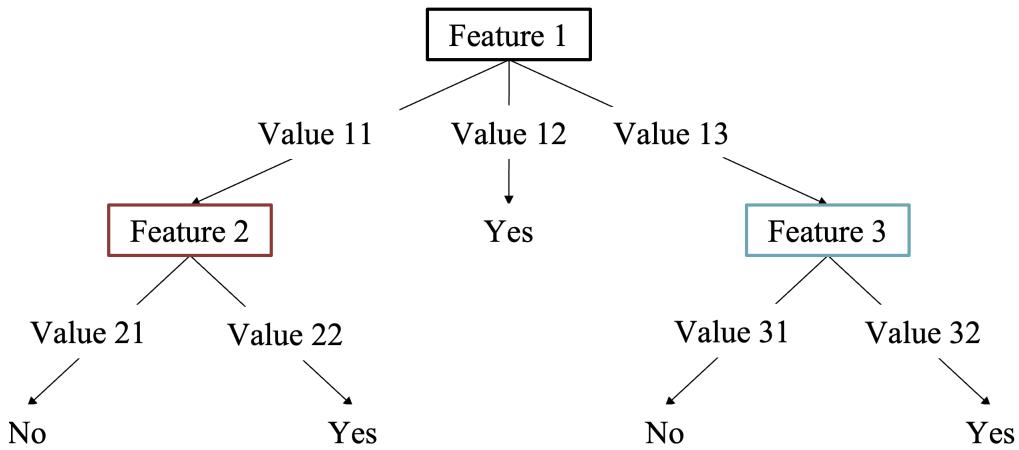
Decision Tree - Cây quyết định là phương pháp học có giám sát không tham số dựa vào ý tưởng xây dựng mô hình dạng cây để xấp xỉ dữ liệu huấn luyện.



Hình 13.1: Mô hình biểu diễn cây quyết định cho bài toán phân loại và hồi quy.

Cây quyết định có một số đặc trưng sau:

- Cây có các node là các thuộc tính của dữ liệu. Ví dụ như thuộc tính về Nhiệt độ.
- Các đường nối giữa các node là các giá trị của thuộc tính. Ví dụ thuộc tính nhiệt độ sẽ có các giá trị như: Cao, Trung Bình, Thấp.
- Các node lá đại diện cho các kết quả có thể có của mô hình. Ví dụ với bài toán phân loại nhị phân thì các node lá có thể nhận kết quả là '0' hoặc '1'.

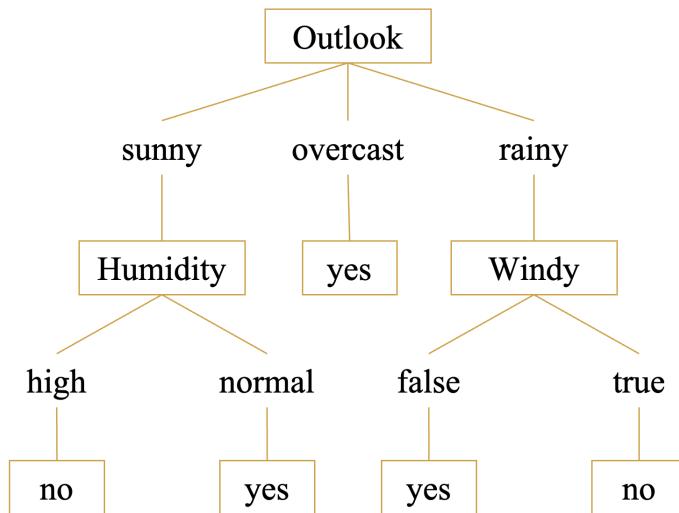


Hình 13.2: Mô hình biểu diễn cây quyết định.

Để xây dựng cây quyết định trên, chúng ta sử dụng thuật toán Iterative Dichotomiser (ID3) xây dựng theo chiều từ trên xuống dựa theo một số độ đo để xác định thuộc tính nào sẽ là node gốc và các thuộc tính ở node con là gì? Một số độ đo thường được sử dụng để đánh giá độ quan trọng của các thuộc tính như Gini Impurity hoặc Entropy / Information Gain cho bài toán classification và Variance / Sum of Squared Errors (SSE) cho bài toán regression.

Trong quá trình huấn luyện chúng ta sẽ đánh giá các độ đo và xem xét thuộc tính dựa vào các độ đo này có tốt để phân chia tập dữ liệu thành các tập con. Chúng ta sẽ tìm hiểu về các độ đo ở phần tiếp theo.

Ví dụ về cây quyết định sau khi được huấn luyện như sau:



Hình 13.3: Cây quyết định cho phân loại nhị phân.

13.2 Cây quyết định ứng dụng cho bài toán phân loại

Trong bài toán phân loại, mục tiêu của mô hình là gán mỗi điểm dữ liệu đầu vào vào một lớp cụ thể. Ví dụ: Phân loại email là spam hoặc không spam, dựa trên các đặc trưng như: có chứa từ “free”, độ dài email, số lượng dấu chấm than... Tiêu chí phân chia phổ biến trong phân loại:

- Information Gain (Entropy): Đo mức độ “nhiều loạn” của dữ liệu, chọn thuộc tính giúp giảm entropy lớn nhất.
- Gini Impurity: Ước tính xác suất một phần tử bị phân loại sai nếu được gán nhau ngẫu nhiên theo phân phối lớp trong nút.

Dộ đo Gini có thể được xác định bởi công thức:

$$Gini(D) = \frac{n_1}{n} Gini(D_1) + \frac{n_2}{n} Gini(D_2)$$

$$Gini(D_i) = 1 - \sum_{j=1}^c p_j^2$$

Tập D là tập dữ liệu ban đầu có n phần tử và dựa vào tiêu chí A có thể được phân chia thành 2 tập con là D_1 có n_1 phần tử và D_2 có n_2 phần tử. p_j là xác suất của các sample trong D_i thuộc vào class c .

Độ đo Information Gain được xác định bởi công thức:

$$Gain(D, A) = Entropy(D) - Entropy(D, A)$$

Độ đo Entropy được xác định bởi công thức:

$$Entropy(D, A) = \frac{n_1}{n} Entropy(D_1) + \frac{n_2}{n} Entropy(D_2)$$

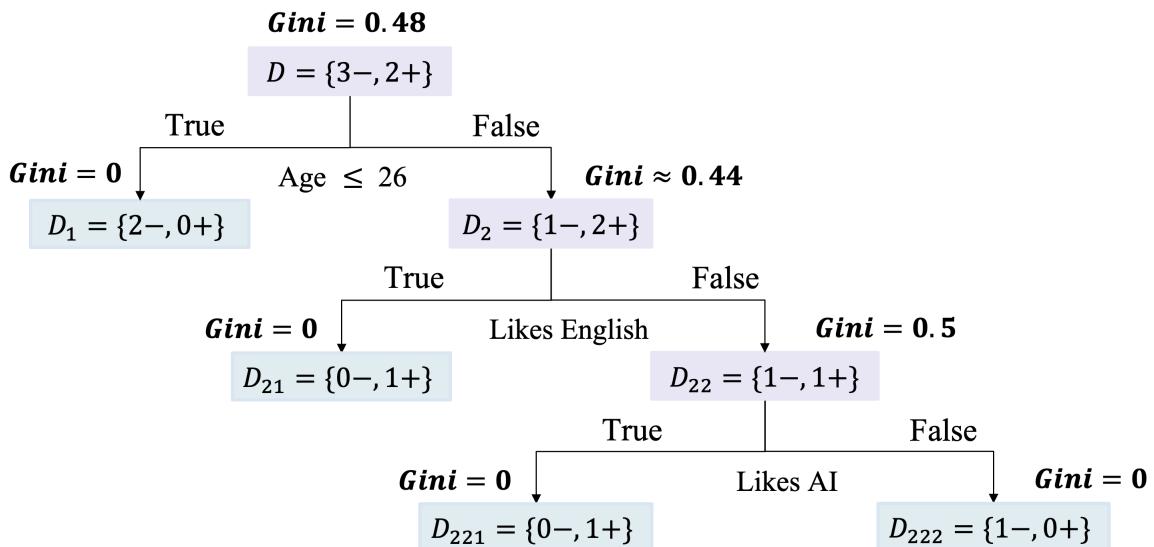
$$Entropy(D_i) = - \sum_{j=1}^c p_j \log_2 p_j$$

Để xác định được thuộc tính quan trọng với cây quyết định chúng ta có thể chọn *Entropy* hoặc *Gini* thấp nhất, *Gain* cao nhất.

Quy trình huấn luyện cây

- Tính độ quan trọng của dữ liệu tại một nút (theo các chỉ số như Gini hoặc Entropy).
- Duyệt tất cả thuộc tính và chọn thuộc tính chia tách tốt nhất.
- Tạo các nhánh con từ điều kiện chia tách.
- Áp dụng lại quá trình cho từng nhánh con (đệ quy).
- Dừng khi: Tất cả mẫu thuộc về cùng một lớp hoặc số mẫu nhỏ hơn ngưỡng tối thiểu, hoặc cây đạt chiều sâu tối đa.

Ví dụ về cây quyết định sau khi được huấn luyện như sau:



Hình 13.4: Ví dụ dây quyết định sau khi huấn luyện.

Dựa vào các gợi ý trong phần sau để hoàn thiện các đoạn code huấn luyện mô hình cây quyết định cho bài toán phân loại:

13.2.1 Cài đặt và Import thư viện

Chúng ta import các thư viện cần thiết để thực hiện các bước tiền xử lý, trích xuất đặc trưng, huấn luyện mô hình và đánh giá mô hình. Các thư viện này bao gồm:

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from sklearn import datasets
6 from sklearn.datasets import fetch_openml
7 from sklearn.model_selection import train_test_split
8 from sklearn.metrics import accuracy_score, classification_report
9 from sklearn.tree import DecisionTreeClassifier,
                                DecisionTreeRegressor, plot_tree
  
```

```
10 from sklearn.preprocessing import StandardScaler, MinMaxScaler,
   LabelEncoder
```

13.2.2 Đọc và khám phá bộ dữ liệu

Chúng ta load bộ dữ liệu này bằng thư viện `datasets` và in ra một số thông tin cơ bản về bộ dữ liệu như sau:

```
1 # Load the diabetes dataset
2 iris = datasets.load_iris()
3 iris_X, iris_y = iris.data, iris.target
4 feature_names = iris.feature_names
5 target_names = iris.target_names
6
7 print(f"Kích thước dữ liệu: {iris_X.shape}")
8 print(f"Số lượng đặc trưng: {len(feature_names)}")
9 print(f"Tên các đặc trưng: {feature_names}")
10 print(f"Số lượng lớp: {len(target_names)}")
11 print(f"Tên các lớp: {target_names}")
```

Nội dung được in ra màn hình như sau:

```
Kích thước dữ liệu: (150, 4)
Số lượng đặc trưng: 4
Tên các đặc trưng: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
Số lượng lớp: 3
Tên các lớp: ['setosa' 'versicolor' 'virginica']
```

13.2.3 Trực quan hóa dữ liệu

Chúng ta bắt đầu bằng việc tạo DataFrame từ dữ liệu gốc.

```
1 iris_df = pd.DataFrame(iris_X, columns=feature_names)
2 iris_df["target"] = iris_y
```

```

3 iris_df["species"] = [target_names[i] for i in iris_y]
4 iris_df

```

Trực quan hoá thông qua các dạng biểu đồ sau:

- **Biểu đồ Boxplot:** Giúp so sánh phân phối các đặc trưng giữa các loài. Nhờ đó, ta có thể phát hiện đặc trưng nào có khả năng phân tách tốt giữa các lớp.
- **Heatmap ma trận tương quan:** Phân tích mối quan hệ giữa các đặc trưng với nhau. Điều này giúp tránh việc đưa vào mô hình những đặc trưng bị trùng lặp thông tin.
- **Scatter plot:** Hình dung phân bố các điểm dữ liệu trong không gian 2 chiều, chẳng hạn như *Sepal Length* và *Petal Length*, từ đó quan sát được sự phân cụm giữa các lớp.

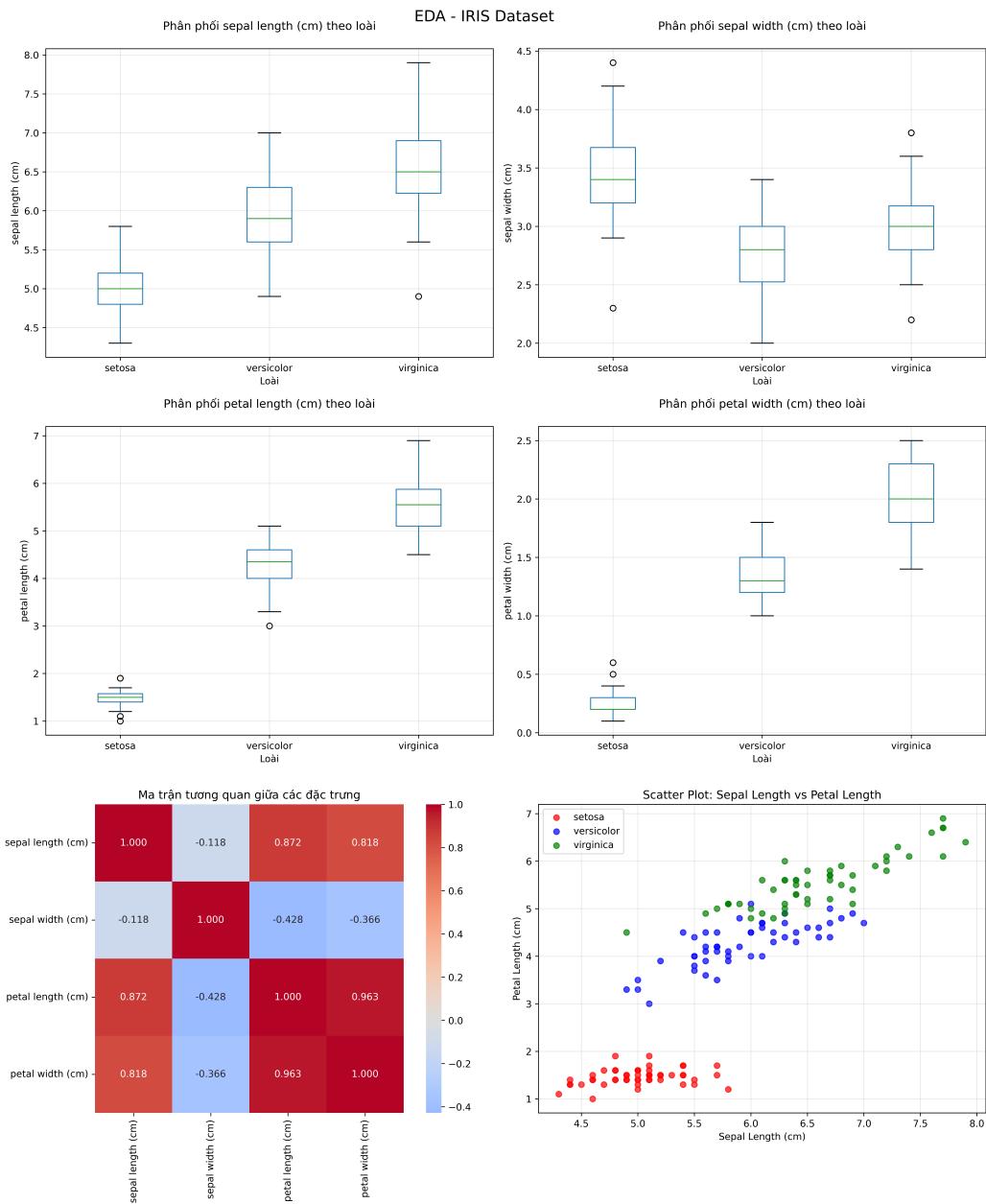
```

1 fig, axes = plt.subplots(3, 2, figsize=(15, 18))
2
3 # 1. Phân phối các đặc trưng theo loài
4 for i, feature in enumerate(feature_names):
5     ax = axes[i//2, i%2]
6     iris_df.boxplot(column=feature, by="species", ax=ax)
7     ax.set_title(f"Phân phối {feature} theo loài", fontsize=12, pad=20)
8     ax.set_xlabel("Loài", fontsize=10)
9     ax.set_ylabel(feature, fontsize=10)
10    ax.grid(True, alpha=0.3)
11
12 # 2. Ma trận tương quan
13 axes[2, 0].remove()
14 axes[2, 0] = plt.subplot(3, 2, 5)
15 correlation_matrix = iris_df[feature_names].corr()
16 sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", center=0,
17               square=True, ax=axes[2, 0], fmt=".3f")
18 axes[2, 0].set_title("Ma trận tương quan giữa các đặc trưng",
19                      fontsize=12)
20
21 # 3. Scatter plot
22 axes[2, 1].remove()

```

```
22 axes[2, 1] = plt.subplot(3, 2, 6)
23 colors = ["red", "blue", "green"]
24 for i, species in enumerate(target_names):
25     species_data = iris_df[iris_df["species"] == species]
26     axes[2, 1].scatter(species_data["sepal length (cm)"],
27                         species_data["petal length (cm)"],
28                         label=species, alpha=0.7, color=colors[i])
29 axes[2, 1].set_xlabel("Sepal Length (cm)", fontsize=10)
30 axes[2, 1].set_ylabel("Petal Length (cm)", fontsize=10)
31 axes[2, 1].set_title("Scatter Plot: Sepal Length vs Petal Length",
32                      fontsize=12)
33 axes[2, 1].legend()
34 axes[2, 1].grid(True, alpha=0.3)
35 plt.suptitle("EDA - IRIS Dataset", fontsize=16, y=0.98)
36 plt.tight_layout()
37 plt.savefig("01_EDA_IRIS.png", dpi=600, bbox_inches="tight")
38 plt.show()
```

Kết quả trực quan hóa dữ liệu về giá trị các số liệu thông kê và phân bố dữ liệu như hình sau:



Hình 13.5: Trực quan hoá dữ liệu IRIS.

13.2.4 Chia tập dữ liệu và chuẩn hoá

Dữ liệu được chia theo tỷ lệ 80-20. Sau đó, chuẩn hoá dữ liệu bằng StandardScaler giúp đưa các đặc trưng về cùng một thang đo, tránh trường hợp mô hình thiên vị các đặc trưng có giá trị lớn.

```

1 X_train_iris, X_test_iris, y_train_iris, y_test_iris =
    train_test_split(
2     # Your code here
3 )
4
5 print(f"Kích thước tập huấn luyện: {X_train_iris.shape}")
6 print(f"Kích thước tập kiểm tra: {X_test_iris.shape}")
7 print(f"Phân phối lớp trong tập huấn luyện: {np.bincount(
        y_train_iris)}")
8 print(f"Phân phối lớp trong tập kiểm tra: {np.bincount(y_test_iris)}
    ")
```

Kết quả thu được:

```

Kích thước tập huấn luyện: (120, 4)
Kích thước tập kiểm tra: (30, 4)
Phân phối lớp trong tập huấn luyện: [40 40 40]
Phân phối lớp trong tập kiểm tra: [10 10 10]
```

Chuẩn hoá dữ liệu:

```

1 # Áp dụng StandardScaler
2 scaler_iris = StandardScaler()
3 X_train_iris_scaled = # Your code here
4 X_test_iris_scaled = # Your code here
```

13.2.5 Huấn luyện và đánh giá mô hình

Sau khi xử lý xong dữ liệu, chúng ta tiến hành xây dựng mô hình cây quyết định cho phân loại. Mô hình được giới hạn max_depth=5 để tránh overfitting

và min_samples_split=10 để kiểm soát độ phân nhánh.

```
1 # Xây dựng mô hình Decision Tree
2 dt_classifier = # Your code here
3 dt_classifier.fit(X_train_iris, y_train_iris)
4
5 # Dự đoán
6 y_pred_iris = dt_classifier.predict(X_test_iris)
7
8 # Đánh giá
9 print(classification_report(y_test_iris, y_pred_iris, target_names=
10                           target_names))
accuracy = accuracy_score(y_test_iris, y_pred_iris)
11 print(f"Độ chính xác: {accuracy:.4f}") # 96\%
```

13.3 Cây quyết định ứng dụng cho bài toán hồi quy

Khác với phân loại (nơi cây đưa ra các nhãn), cây hồi quy dự đoán một giá trị số tại mỗi nút lá. Mỗi bước trong cây cố gắng chia dữ liệu sao cho giảm sai số nội tại của từng nhánh con. Bài toán hồi quy nhằm dự đoán một **giá trị liên tục** từ dữ liệu đầu vào. Một số ví dụ thực tế:

- Dự đoán giá nhà dựa vào diện tích, vị trí, số phòng.
- Dự đoán nhiệt độ theo dữ liệu thời tiết.
- Dự đoán điểm số, thời gian hoàn thành, doanh thu,...

Decision Tree Regressor là phiên bản mở rộng của cây quyết định, hoạt động hiệu quả cho bài toán hồi quy mà không giả định quan hệ tuyến tính giữa đầu vào và đầu ra. Cách hoạt động:

- Cây thực hiện chia nhỏ dữ liệu đầu vào theo điều kiện tại mỗi nút.
- Mỗi nút nhánh là một điều kiện kiểm tra trên thuộc tính (ví dụ: **Area > 120**).
- Mỗi nút lá chứa **giá trị dự đoán** là trung bình của nhãn (target) tại nút đó.
- Quá trình dự đoán được thực hiện bằng cách duyệt cây từ gốc đến lá.

Với bài toán Regression. Để chọn được thuộc tính tốt phân chia bộ dữ liệu và xây dựng cây, chúng ta sẽ sử dụng độ đo SSE. Độ đo Sum of Squared Error (SSE) được xác định bởi công thức:

$$SSE(D) = SSE(D_1) + SSE(D_2)$$

$$SSE(D_i) = \sum_{j=1}^{n_i} (x_j - \bar{x}_i)^2$$

Trong đó D là tập dữ liệu ban đầu có n phần tử và có thể được phân chia thành 2 tập con là D_1 có n_1 phần tử và D_2 có n_2 phần tử. \bar{x}_i là giá trị trung bình của các phần tử trong tập D_i .

Để xác định được thuộc tính quan trọng với cây quyết định chúng ta chọn SSE với giá trị thấp nhất.

Dựa trên các gợi ý sau đây để hoàn thiện các đoạn code:

13.3.1 Cài đặt và Import thư viện

cài đặt các thư viện cần thiết:

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from sklearn import datasets
6 from sklearn.datasets import fetch_openml
7 from sklearn.model_selection import train_test_split
8 from sklearn.metrics import mean_squared_error,
                           classification_report,
                           confusion_matrix, r2_score
9 from sklearn.tree import DecisionTreeClassifier,
                           DecisionTreeRegressor, plot_tree
10 from sklearn.preprocessing import StandardScaler, MinMaxScaler,
                           LabelEncoder
```

13.3.2 Đọc và khám phá bộ dữ liệu

Tải về bộ dữ liệu và chuyển sang DataFrame để xử lý:

```
1 machine_cpu = fetch_openml(name="machine_cpu", version=1)
2 cpu_X, cpu_y = machine_cpu.data, machine_cpu.target
3 cpu_feature_names = machine_cpu.feature_names
4
5 cpu_df = pd.DataFrame(cpu_X, columns=cpu_feature_names)
6 cpu_df["performance"] = cpu_y
7
8 print(f"Kích thước dữ liệu: {cpu_df.shape}")
9 print(f"Các đặc trưng: {cpu_feature_names}")
10 print(f"Target: Performance (CPU Relative Performance)")
```

Kích thước dữ liệu: (209, 7)

Các đặc trưng: ['MYCT', 'MMIN', 'MMAX', 'CACH', '1CHMIN', 'CHMAX']

Target: Performance (CPU Relative Performance)

13.3.3 Trực quan hóa dữ liệu

Khác với phân loại, bài toán hồi quy yêu cầu dự đoán một giá trị liên tục. Bộ dữ liệu CPU cung cấp các đặc trưng phần cứng như MYCT, MMIN, CACH... với nhãn là hiệu suất xử lý (performance). Trong bước EDA này:

- **Histograms:** Cho biết phân phối của từng đặc trưng. Một số đặc trưng bị lệch, có thể cần biến đổi (như log-transform).
- **Heatmap tương quan:** Cho thấy đặc trưng nào có mối tương quan cao với giá trị cần dự đoán.
- **Boxplot:** Dễ dàng phát hiện các điểm ngoại lai (outliers), vốn có thể ảnh hưởng lớn đến bài toán hồi quy.

```

1 fig, axes = plt.subplots(3, 3, figsize=(18, 15))
2 axes = axes.flatten()
3
4 # 1-6. Histogram của các đặc trưng
5 for i, col in enumerate(cpu_feature_names):
6     axes[i].hist(cpu_df[col], bins=20, alpha=0.7, color="lightblue",
7                   edgecolor="black")
8     axes[i].set_title(f"Phân phối {col}", fontsize=12)
9     axes[i].set_xlabel(col, fontsize=10)
10    axes[i].set_ylabel("Tần số", fontsize=10)
11    axes[i].grid(True, alpha=0.3)
12
13 # 7. Histogram của target
14 axes[6].hist(cpu_df["performance"], bins=20, alpha=0.7, color="lightcoral",
15               edgecolor="black")
16 axes[6].set_title("Phân phối Performance (Target)", fontsize=12)
17 axes[6].set_xlabel("Performance", fontsize=10)
18 axes[6].set_ylabel("Tần số", fontsize=10)
19 axes[6].grid(True, alpha=0.3)

```

```

18
19 # 8. Ma trận tương quan
20 correlation_matrix_cpu = cpu_df[cpu_feature_names + ["performance"
21 ]].corr()
22 sns.heatmap(correlation_matrix_cpu, annot=True, cmap="coolwarm",
23 center=0,
24 square=True, ax=axes[7], fmt=".3f")
25 axes[7].set_title("Ma trận tương quan", fontsize=12)
26
27 # 9. Boxplot để kiểm tra outliers
28 cpu_df.boxplot(column=cpu_feature_names, ax=axes[8])
29 axes[8].set_title("Boxplot - Kiểm tra Outliers", fontsize=12)
30 axes[8].set_ylabel("Giá trị", fontsize=10)
31 axes[8].tick_params(axis="x", rotation=45)
32
33 plt.suptitle("EDA - CPU Performance Dataset", fontsize=16, y=0.98)
34 plt.tight_layout()
35 plt.show()

```

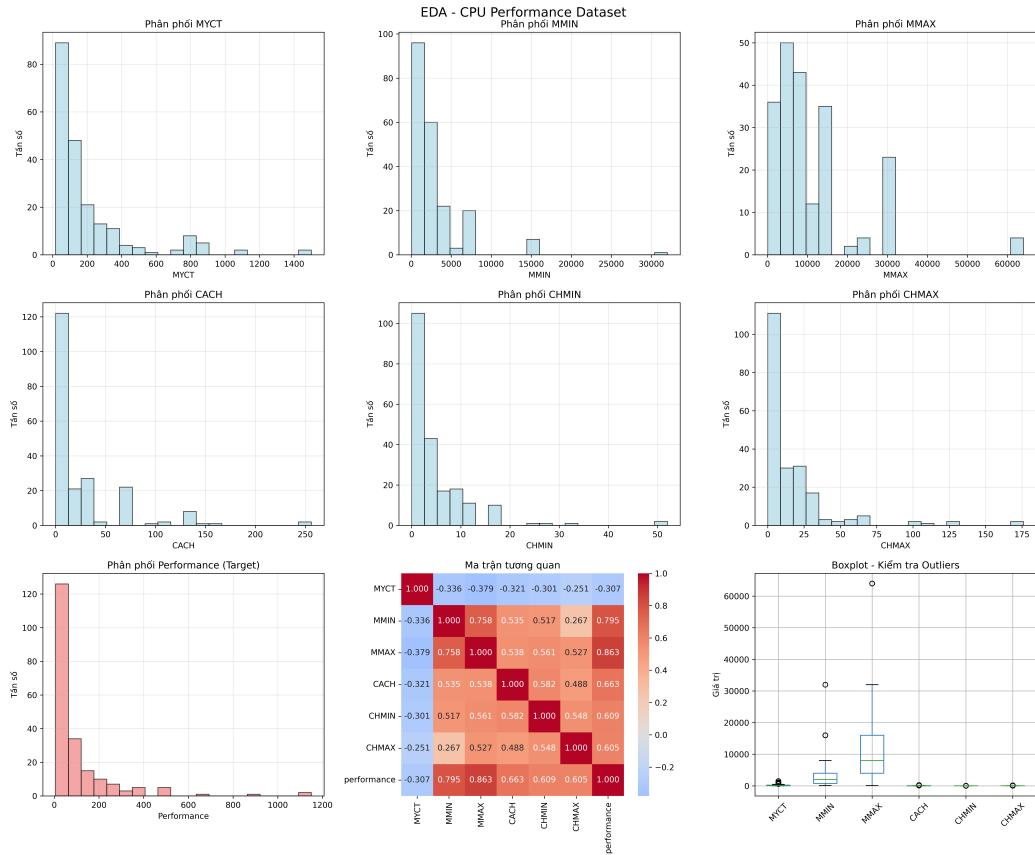
13.3.4 Chia tập dữ liệu và chuẩn hoá

Dữ liệu được chia theo tỷ lệ 80-20. Sau đó, chuẩn hoá dữ liệu bằng StandardScaler giúp đưa các đặc trưng về cùng một thang đo, tránh trường hợp mô hình thiên vị các đặc trưng có giá trị lớn.

```

1 # Chia dữ liệu
2 X_train_cpu, X_test_cpu, y_train_cpu, y_test_cpu = train_test_split(
3     # Your code here
4 )
5
6 # Chuẩn hoá
7 scaler_cpu = StandardScaler()
8 X_train_cpu_scaled = # Your code here
9 X_test_cpu_scaled = # Your code here

```



Hình 13.6: Trực quan hóa dữ liệu CPU.

13.3.5 Huấn luyện và đánh giá mô hình

Sau khi xử lý xong dữ liệu, chúng ta tiến hành xây dựng mô hình cây quyết định cho bài toán hồi quy. Mô hình được giới hạn `max_depth=5` để tránh overfitting.

```

1 # Xây dựng mô hình Decision Tree Regression
2 dt_regressor = # Your code here
3 dt_regressor.fit(X_train_cpu, y_train_cpu)
4
5 # Dự đoán
6 y_pred_cpu = dt_regressor.predict(X_test_cpu)

```

```
7 # Đánh giá
8 mse = mean_squared_error(y_test_cpu, y_pred_cpu)
9 rmse = np.sqrt(mse)
10 mae = np.mean(np.abs(y_test_cpu - y_pred_cpu))
11 r2 = r2_score(y_test_cpu, y_pred_cpu)
12
13
14 print(f"MSE: {mse:.4f}")
15 print(f"RMSE: {rmse:.4f}")
16 print(f"MAE: {mae:.4f}")
17 print(f"R2: {r2:.4f}")
```

Kết quả thu được:

```
MSE: 8545.5913
RMSE: 92.4424
MAE: 38.9844
R2: 0.8321
```

13.4 Câu hỏi trắc nghiệm

Age	Likes English	Likes AI	Raise Salary
23	0	0	0
25	1	1	0
27	1	0	1
29	0	1	1
29	0	0	0

Hình 13.7: Bảng dữ liệu cho bài toán Classification.

Dựa vào công thức tính và bảng dữ liệu gồm các thuộc tính Age, Likes English, Likes AI và cột nhãn Raise Salary trả lời các câu hỏi sau đây:

1. Giả sử cho bài toán bộ dữ liệu D gồm các ví dụ được phân loại thành 2 lớp. Độ đo Entropy bằng 1 khi nào?
 - (a) Số dữ liệu trong hai lớp bằng nhau
 - (b) Số dữ liệu trong lớp 1 bằng 3 lần số dữ liệu trong lớp 2
 - (c) Số dữ liệu trong lớp 1 bằng 2 lần số dữ liệu trong lớp 2
 - (d) Số dữ liệu trong lớp 1 bằng 4 lần số dữ liệu trong lớp 2
2. Tính giá trị Gini của các mẫu trong cột nhãn $D = \text{'Raise Salary'}$?
 - (a) $\text{Gini}(D) = 0.40$
 - (b) $\text{Gini}(D) = 0.44$
 - (c) $\text{Gini}(D) = 0.48$
 - (d) $\text{Gini}(D) = 0.46$
3. Tính Gini của bộ dữ liệu khi thuộc tính ‘Likes English’ được chọn là node gốc?

- (a) Gini(Likes English) = 0.40
- (b) Gini(Likes English) = 0.43
- (c) Gini(Likes English) = 0.45
- (d) Gini(Likes English) = 0.47

Bảng dữ liệu gồm các thuộc tính Age, Likes English, Likes AI và cột nhãn Raise Salary. Trong đó sẽ có thuộc tính ‘Age’ thuộc kiểu dữ liệu số liên tục. Vì vậy với thuộc tính này, chúng ta sẽ sắp xếp lại các hàng theo thứ tự tăng dần theo cột ‘Age’. Tương ứng với mỗi cặp giá trị liên tiếp chúng ta sẽ tính trung bình. Dựa vào các giá trị trung bình này để tính các độ đo. Ví dụ về phân chia các giá trị trong cột ‘Age’ mô tả như hình sau:

Age	Likes English	Likes AI	Raise Salary
24	23	0	0
26	25	1	0
28	27	1	1
29	29	0	1
29	29	0	0

Hình 13.8: Phân loại trên cột dữ liệu liên tục ‘Age’.

4. Tính Gini của bộ dữ liệu khi thuộc tính ‘Age’ được chọn là node gốc với điều kiện phân chia thành tập D_1 và D_2 là ‘Age ≤ 26 ’?
 - (a) Gini(Age ≤ 26) = 0.25
 - (b) Gini(Age ≤ 26) = 0.26
 - (c) Gini(Age ≤ 26) = 0.27
 - (d) Gini(Age ≤ 26) = 0.28
5. Tính giá trị Entropy của các sample trong cột nhãn D = ‘Raise Salary’?
 - (a) Entropy(D) = 0.99

- (b) Entropy(D) = 0.97
(c) Entropy(D) = 0.95
(d) Entropy(D) = 0.93
6. Tính Gain của bộ dữ liệu khi thuộc tính ‘Likes English’ được chọn là node gốc?
- (a) Gain(Likes English) = 0.049
(b) Gain(Likes English) = 0.039
(c) Gain(Likes English) = 0.029
(d) Gain(Likes English) = 0.019
7. Dòng lệnh nào sau đây để tải về bộ dữ liệu Iris từ thư viện ‘sklearn’ là?
- (a) iris_X, iris_y = datasets.load_iris(return_X_y=True)
(b) iris_X, iris_y = datasets.load_iris()
(c) iris_X, iris_y = datasets.load_iris_data(return_X_y=True)
(d) iris_X, iris_y = datasets.load_iris_data()

Age	Likes English	Likes AI	Salary
23	0	0	200
25	1	1	400
27	1	0	300
29	0	1	500
29	0	0	400

Hình 13.9: Bảng dữ liệu cho bài toán Regression.

Dựa vào công thức tính và bảng dữ liệu gồm các thuộc tính Age, Likes English, Likes AI và cột giá trị thực tế Salary trả lời các câu hỏi sau đây:

8. Tính SSE của bộ dữ liệu khi thuộc tính ‘Likes AI’ được chọn là node gốc?
 - (a) $SSE(\text{Likes AI}) = 25000$
 - (b) $SSE(\text{Likes AI}) = 20000$
 - (c) $SSE(\text{Likes AI}) = 45000$
 - (d) $SSE(\text{Likes AI}) = 50000$
9. Tính SSE của bộ dữ liệu khi thuộc tính ‘Age’ được chọn là node gốc với điều kiện phân chia thành tập D_1 và D_2 là ‘Age ≤ 24 ’?
 - (a) $SSE(\text{Age} \leq 24) = 20000$
 - (b) $SSE(\text{Age} \leq 24) = 30000$
 - (c) $SSE(\text{Age} \leq 24) = 40000$
 - (d) $SSE(\text{Age} \leq 24) = 50000$
10. Độ đo nào sau đây thường được sử dụng để đánh giá mô hình DecisionTreeRegressor?
 - (a) Accuracy
 - (b) Precision

- (c) Mean Squared Error (MSE)
- (d) F1-score

- 1. Hint:** Các file code gợi ý và dữ liệu nếu có được lưu trong thư mục có thể được tải [tại đây](#).
- 2. Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. Rubric:

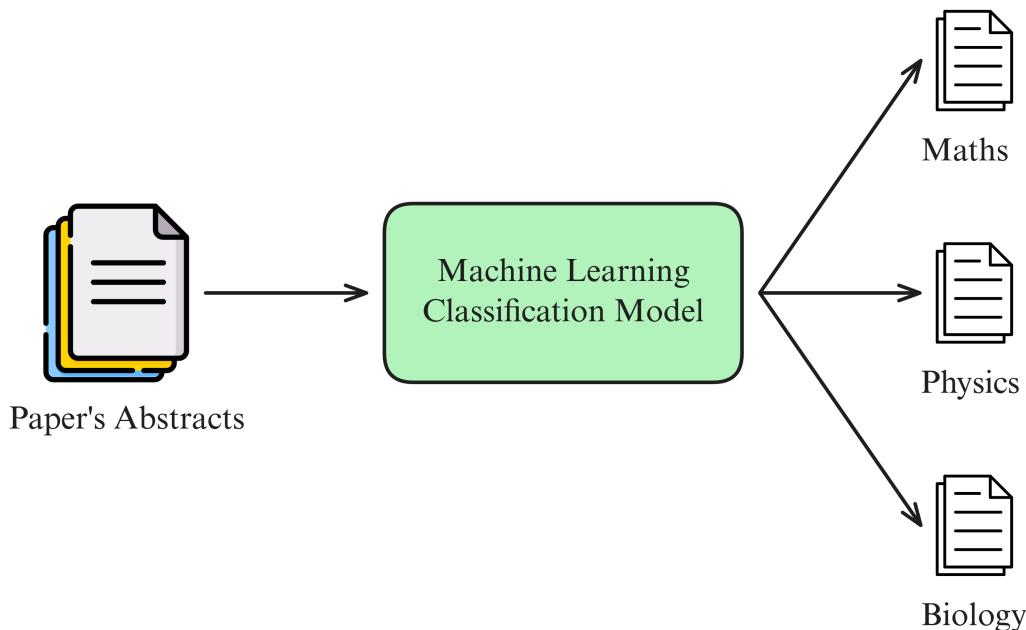
Mục	Kiến Thức	Đánh Giá
1.	<ul style="list-style-type: none"> • Nắm vững kiến thức của mô hình cây quyết định • Thuật toán ID3 để xây dựng cây quyết định 	<ul style="list-style-type: none"> • Xây dựng mô hình phân loại văn bản sử dụng Decision Tree • Sử dụng thư viện <code>sklearn</code> cho mô hình Decision Tree
2.	<ul style="list-style-type: none"> • Nắm vững các độ đo được sử dụng cho các bài toán regression và classification • Hiểu rõ một số kỹ thuật cải tiến mô hình học máy cây quyết định 	<ul style="list-style-type: none"> • Thực thi cho bộ dữ liệu IRIS và CPU Machine

Chương 14

Project 1: Phân loại chủ đề của một bài báo (dùng KNN, Decision Tree, và NBC)

14.1 Giới thiệu

Text Classification (Tạm dịch: Phân loại văn bản) là một trong những bài toán cơ bản và quan trọng trong lĩnh vực Xử lý ngôn ngữ tự nhiên (NLP). Mục tiêu của bài toán này là phân loại các đoạn văn bản vào các nhóm hoặc nhãn khác nhau dựa trên nội dung của chúng. Các ứng dụng phổ biến của Text Classification bao gồm phân loại email (spam vs. ham), phân loại tin tức, phân loại cảm xúc (sentiment analysis), và nhiều ứng dụng khác.



Hình 14.1: Minh họa ứng dụng phân loại topic của paper dựa trên abstract.

Trong project này, chúng ta sẽ xây dựng một chương trình Text Classification liên quan đến việc phân loại một abstract của publication (bài báo khoa học)

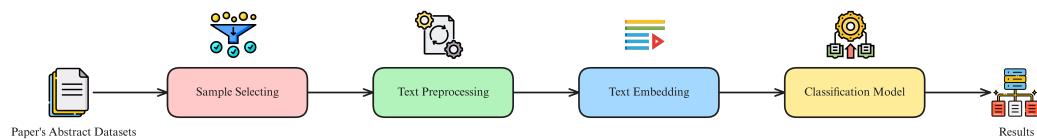
thành các topic khác nhau. Chương trình sẽ được xây dựng trên nền tảng Python và sử dụng các thư viện học máy phổ biến như scikit-learn, numpy, v.v.

Theo đó, Input/Output chung của chương trình sẽ bao gồm:

- **Input:** Một abstract của publication (bài báo khoa học).
- **Output:** Topic của abstract đó (ví dụ: vật lý, toán học, khoa học máy tính, v.v.).

14.2 Xây dựng chương trình phân loại topic của publication abstract

Trong phần này, ta sẽ tiến hành cài đặt chương trình phân loại topic của publication abstract. Chương trình sẽ được xây dựng dựa trên ba phương pháp mã hóa văn bản khác nhau, bao gồm Bag-of-Words (BoW), TF-IDF và Sentence Embeddings. Mỗi phương pháp sẽ được áp dụng với một mô hình phân loại khác nhau, bao gồm Naive Bayes, KNN và Decision Tree.



Hình 14.2: Pipeline

14.2.1 Cài đặt và Import thư viện

Chúng ta import các thư viện cần thiết để thực hiện các bước tiền xử lý, trích xuất đặc trưng, huấn luyện mô hình và đánh giá mô hình. Các thư viện này bao gồm:

```

1 from collections import Counter, defaultdict
2 from typing import List, Dict, Literal, Union
3
4 import re
5 import math
6 import numpy as np
7 import matplotlib.pyplot as plt
8 import seaborn as sns
9
10 from datasets import load_dataset
11 from sentence_transformers import SentenceTransformer
12
13 from sklearn.model_selection import train_test_split
14 from sklearn.metrics import accuracy_score, classification_report,
15                                         confusion_matrix
16 from sklearn.feature_extraction.text import CountVectorizer,
17                                         TfidfVectorizer

```

```
16 from sklearn.cluster import KMeans
17 from sklearn.neighbors import KNeighborsClassifier
18 from sklearn.tree import DecisionTreeClassifier
19 from sklearn.naive_bayes import GaussianNB
20
21 import warnings
22
23 warnings.filterwarnings("ignore")
24
25 CACHE_DIR = "./cache"
```

Trước tiên, chúng ta cùng điểm qua một số thư viện chính được sử dụng trong bài:

- **re:** Thư viện để làm việc với biểu thức chính quy (regular expressions), giúp xử lý và phân tích chuỗi văn bản.
- **datasets:** Thư viện để tải và làm việc với các bộ dữ liệu từ Hugging Face.
- **sentence-transformers:** Thư viện để tạo và sử dụng các mô hình sentence embeddings, giúp chuyển đổi văn bản thành các vector ngữ nghĩa.
- **matplotlib.pyplot:** Thư viện để vẽ đồ thị và biểu đồ, hỗ trợ trực quan hóa dữ liệu.
- **seaborn:** Thư viện vẽ đồ thị thống kê, cung cấp các hàm tiện ích để trực quan hóa dữ liệu một cách đẹp mắt.
- **numpy:** Cung cấp các đối tượng mảng đa chiều và các hàm toán học để làm việc với các mảng này.
- **scikit-learn:** Thư viện học máy phổ biến, giúp xây dựng và triển khai các mô hình học máy phức tạp một cách nhanh chóng.

14.2.2 Đọc và khám phá bộ dữ liệu

Trong bài này, chúng ta sẽ sử dụng bộ dữ liệu [UniverseTBD/arxiv-abstracts-large](#) được cung cấp trên HuggingFace. Bộ dữ liệu này bao gồm các abstract

(tóm tắt) của các bài báo khoa học được đăng trên arXiv, một kho lưu trữ trực tuyến cho các bài báo khoa học trong nhiều lĩnh vực khác nhau. Bộ dữ liệu này có thể được sử dụng để phân loại các abstract theo các chủ đề khác nhau hoặc để phân tích nội dung của các bài báo khoa học, bao gồm các thông tin như tiêu đề, tác giả, ngày cập nhật, v.v. Bộ dữ liệu này có kích thước lớn với hơn 2 triệu bản ghi.

Chúng ta load bộ dữ liệu này bằng thư viện `datasets` với hàm `load_dataset` và in ra một số thông tin cơ bản về bộ dữ liệu như sau:

```
1 ds = load_dataset("UniverseTBD/arxiv-abstracts-large")
2 ds
```

Nội dung được in ra màn hình như sau:

```
DatasetDict(
  train: Dataset(
    features: ['id', 'submitter', 'authors', 'title', 'comments', 'journal-ref',
               'doi', 'report-no', 'categories', 'license', 'abstract', 'versions', 'update_-_
date', 'authors_parsed'],
    num_rows: 2292057
  )
)
```

Các fields của bộ dữ liệu bao gồm:

- **id:** ID của paper trên arXiv.
- **submitter:** Người nộp bài báo.
- **authors:** Danh sách các tác giả của bài báo.
- **title:** Tiêu đề của bài báo.
- **comments:** Các bình luận liên quan đến bài báo.
- **journal-ref:** Tham chiếu đến tạp chí nơi bài báo được xuất bản.
- **doi:** Digital Object Identifier, mã định danh duy nhất cho bài báo.

- **report-no:** Số báo cáo liên quan đến bài báo.
- **categories:** Các chủ đề hoặc lĩnh vực mà bài báo thuộc về.
- **license:** Giấy phép sử dụng của bài báo.
- **abstract:** Tóm tắt nội dung của bài báo (đây là field chúng ta sẽ sử dụng để phân loại).
- **versions:** Phiên bản của bài báo.
- **update_date:** Ngày cập nhật của bài báo.
- **authors_parsed:** Danh sách các tác giả đã được phân tích và chuẩn hóa.

Chúng ta sẽ sử dụng field **abstract** để làm input cho mô hình phân loại, và field **categories** để làm nhãn (label) cho mô hình.

Để quan sát dữ liệu, chúng ta in ra màn hình 3 bản ghi đầu tiên của bộ dữ liệu như sau:

```

1 # print three first examples
2 for i in range(3):
3     print(f"Example {i+1}:")
4     print(ds['train'][i]['abstract'])
5     print(ds['train'][i]['categories'])
6     print("---" * 20)

```

Dựa vào dữ liệu ta thấy dữ liệu abstract của chúng ta có chứa nhiều kí tự đặc biệt như dấu chấm, dấu phẩy, dấu ngoặc kép, v.v. Ngoài ra, các abstract này cũng có độ dài khác nhau, có thể từ vài câu đến vài đoạn văn. Các nhãn (categories) của các abstract này cũng rất đa dạng, bao gồm nhiều lĩnh vực khác nhau như vật lý, toán học, khoa học máy tính, v.v. Chúng ta in toàn bộ các categories của abstract để xem xét các nhãn này có phù hợp với bài toán phân loại hay không như sau:

```

1 all_categories = ds['train']['categories']
2 print(set(all_categories))

```

Categories của abstract tuân theo format <primary category> <secondary category> ..., ví dụ như hep-ph hay math.CO cs.CG. Chúng ta sẽ sử dụng field `abstract` để làm input cho mô hình phân loại, và phần primary category (category đầu tiên) trong field `categories` để làm nhãn (label) cho mô hình. Để quan sát số lượng primary categories trong bộ dữ liệu, chúng ta sẽ đếm số lượng các nhãn này và in ra như sau:

```
1 all_categories = ds['train']['categories']
2 category_set = set()
3
4 # Collect unique labels
5 for category in all_categories:
6     parts = category.split(' ')
7     for part in parts:
8         topic = part.split('.')[0]
9         category_set.add(topic)
10
11 # Sort the labels and print them
12 sorted_categories= sorted(list(category_set), key=lambda x: x.lower
13                           ())
13 print(f'There are {len(sorted_categories)} unique primary categories
14   in the dataset:')
14 for category in sorted_categories:
15     print(category)
```

Kết quả in ra sẽ là danh sách các primary categories trong bộ dữ liệu như sau:

There are 38 unique primary categories in the dataset:

acc-phys
adap-org
alg-geom
ao-sci
astro-ph
atom-ph
bayes-an
chao-dyn
chem-ph

```

cmp-lg
comp-gas
cond-mat
cs
...

```

Chúng ta sẽ lấy 1000 samples, các samples này có primary là một trong 5 categories sau: `astro-ph`, `cond-mat`, `cs`, `math`, `physics`.

```

1 # load 1000 samples with single label belonging to specific
   categories
2 samples = []
3 CATEGORIES_TO_SELECT = ['astro-ph', 'cond-mat', 'cs', 'math', 'physics']
4 for s in ds['train']:
5     if len(s['categories'].split(' ')) != 1:
6         continue
7
8     cur_category = s['categories'].strip().split('.')[0]
9     if cur_category not in CATEGORIES_TO_SELECT:
10        continue
11
12    samples.append(s)
13
14    if len(samples) >= 1000:
15        break
16 print(f"Number of samples: {len(samples)}") # Number of samples:
                                                1000

```

14.2.3 Tiề̂n xử lý dữ liệu

Tiề̂n xử lý dữ liệu đặc trưng: Nội dung của abstract có chứa nhiều kí tự đặc biệt (newline, trailing space, v.v.), viết hoa, cũng như dấu câu. Do đó, chúng ta cần thực hiện các bước tiền xử lý để chuẩn hóa dữ liệu.

Đối với mỗi sample trong danh sách 1000 samples chúng ta đã chọn ra, chúng ta sẽ thực hiện các bước tiền xử lý như sau:

- Loại bỏ các kí tự \n và khoảng trắng ở đầu và cuối chuỗi.
- Loại bỏ các kí tự đặc biệt (dấu câu, kí tự không phải chữ cái hoặc số).

- Loại bỏ các chữ số.
- Chuyển đổi tất cả các chữ cái thành chữ thường (lowercase).
- Lấy nhãn (label) là primary category (phần đầu tiên) trong field categories.

Các bước tiền xử lý trên tương ứng với đoạn code sau:

```

1 preprocessed_samples = []
2 for s in samples:
3     abstract = s['abstract']
4
5     # Remove \n characters in the middle and leading/trailing spaces
6     abstract = abstract.strip().replace("\n", " ")
7
8     # Remove special characters
9     abstract = re.sub(r'^\w\s', ' ', abstract)
10
11    # Remove digits
12    abstract = re.sub(r'\d+', ' ', abstract)
13
14    # Remove extra spaces
15    abstract = re.sub(r'\s+', ' ', abstract).strip()
16
17    # Convert to lower case
18    abstract = abstract.lower()
19
20    # for the label, we only keep the first part
21    parts = s['categories'].split(' ')
22    category = parts[0].split('.')[0]
23
24    preprocessed_samples.append({
25        "text": abstract,
26        "label": category
27    })

```

Tiếp theo, chúng ta cần tạo hai dictionaries để lưu trữ các primary categories dưới dạng số và ngược lại bằng đoạn code sau:

```

1 label_to_id = {label: i for i, label in enumerate(sorted_labels)}
2 id_to_label = {i: label for i, label in enumerate(sorted_labels)}
3

```

```

4 # Print label to ID mapping
5 print("Label to ID mapping:")
6 for label, id_ in label_to_id.items():
7     print(f"{label} --> {id_}")

```

Kết quả in ra sẽ là danh sách các nhãn và ID tương ứng như sau:

Label to ID mapping:
 astro-ph -> 0
 cond-mat -> 1
 cs -> 2
 math -> 3
 physics -> 4

Cuối cùng, chúng ta sẽ chia dữ liệu trên thành 2 tập: tập huấn luyện (train) và tập kiểm tra (test) với tỉ lệ 80% cho tập huấn luyện và 20% cho tập kiểm tra. Chúng ta sẽ sử dụng hàm `train_test_split` từ thư viện `sklearn` để thực hiện việc này.

```

1 X_full = [sample['text'] for sample in preprocessed_samples]
2 y_full = [label_to_id[sample['label']] for sample in
            preprocessed_samples]
3
4 X_train, X_test, y_train, y_test = train_test_split(X_full, y_full,
                                                      test_size=0.2, random_state=42,
                                                      stratify=y_full)
5
6 print(f"Training samples: {len(X_train)}")
7 print(f"Test samples: {len(X_test)}")

```

Số lượng mẫu trong tập huấn luyện và tập kiểm tra sẽ được in ra như sau:

Training samples: 800
 Test samples: 200

14.2.4 Mã hóa văn bản

Văn bản của chúng ta sau khi tiền xử lý sẽ được mã hóa thành các vector số để có thể sử dụng trong mô hình học máy. Có nhiều phương pháp để mã hóa văn bản, nhưng trong bài này, chúng ta sẽ sử dụng từng phương pháp trong ba phương pháp sau và so sánh chúng với nhau: *Bag-of-Words* (BoW), *TF-IDF* (Term Frequency-Inverse Document Frequency), và *Sentence Embeddings*.

Bag-of-Words (BoW)

Bag-of-Words (BoW) là một kỹ thuật biểu diễn văn bản đơn giản và thường được sử dụng. Nó chuyển đổi văn bản thành một vector có độ dài cố định bằng cách đếm số lần xuất hiện của mỗi từ trong văn bản. Phương pháp này bỏ qua ngữ pháp và thứ tự từ nhưng vẫn giữ lại tần suất của các từ.

Ví dụ về cách sử dụng BoW để mã hóa văn bản với class `CountVectorizer` từ thư viện `sklearn` như sau:

```

1 docs = [
2     "I am going to school to study for the final exam.",
3     "The weather is nice today and I feel happy.",
4     "I love programming in Python and exploring new libraries.",
5     "Data science is an exciting field with many opportunities.",
6 ]
7
8 bow = CountVectorizer()
9 vectors = bow.fit_transform(docs)
10
11 for i, vec in enumerate(vectors):
12     print(f"Document {i+1}: {vec.toarray()}")

```

Đoạn code trên khai báo một danh sách các văn bản (`docs`) và sử dụng class `CountVectorizer` để mã hóa chúng thành các vector BoW với hàm `fit_transform`. Sau đó, nó in ra các vector tương ứng với từng văn bản như sau:

Vector BoW khi sử dụng CountVectorizer

```
Document 1: [[1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 2 0
0 0]]
Document 2: [[0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1
1 0]]
Document 3: [[0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0]]
Document 4: [[0 1 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0
0 1]]
```

Chúng ta thấy được mỗi văn bản được mã hóa thành một vector với kích thước bằng với số lượng từ trong từ điển. Mỗi phần tử trong vector tương ứng với tần suất xuất hiện của từ đó trong văn bản.

TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) là một phương pháp mã hóa văn bản nâng cao hơn BoW, nó không chỉ tính tần suất của từ trong văn bản mà còn xem xét tần suất của từ trong toàn bộ tập dữ liệu. Điều này giúp giảm trọng số của các từ phổ biến và tăng trọng số của các từ hiếm gặp, từ đó cải thiện khả năng phân loại. Ví dụ về cách sử dụng TF-IDF để mã hóa văn bản với class `TfidfVectorizer` từ thư viện `sklearn` như sau:

```
1 docs = [
2     "I am going to school to study for the final exam.",
3     "The weather is nice today and I feel happy.",
4     "I love programming in Python and exploring new libraries.",
5     "Data science is an exciting field with many opportunities.",
6 ]
7
8 vectorizer = TfidfVectorizer()
9 tfidf_vectors = vectorizer.fit_transform(docs)
10
11 for i, vec in enumerate(tfidf_vectors):
12     print(f"TF-IDF for Document {i+1}:")
13     print(vec.toarray())
```

Đoạn code trên khai báo một danh sách các văn bản (`docs`) và sử dụng

class `TfidfVectorizer` để mã hóa chúng thành các vector TF-IDF với hàm `fit_transform`. Sau đó, nó in ra các vector tương ứng với từng văn bản như sau:

Vector TF-IDF khi sử dụng `TfidfVectorizer`

TF-IDF for Document 1:

```
[[0.29333722 0. 0. 0. 0.29333722 0. 0. 0. 0.29333722 0.29333722  
0.29333722 0. 0. 0. 0. 0. 0. 0. 0. 0.29333722 0. 0.29333722  
0.23127044 0.58667444 0. 0. 0. ]]
```

TF-IDF for Document 2:

```
[[0. 0. 0.30091213 0. 0. 0. 0.38166888 0. 0. 0. 0. 0.38166888 0.  
0.30091213 0. 0. 0. 0. 0.38166888 0. 0. 0. 0. 0. 0.30091213 0.  
0.38166888 0.38166888 0. ]]
```

TF-IDF for Document 3:

```
[[0. 0. 0.2855815 0. 0. 0. 0.36222393 0. 0. 0. 0. 0. 0.36222393 0.  
0.36222393 0.36222393 0. 0.36222393 0. 0. 0.36222393 0.36222393 0.  
0. 0. 0. 0. 0. 0. ]]
```

TF-IDF for Document 4:

```
[[0. 0.34056989 0. 0.34056989 0. 0.34056989 0. 0. 0.34056989 0. 0. 0.  
0. 0.26850921 0. 0. 0.34056989 0. 0. 0.34056989 0. 0. 0. 0.34056989 0.  
0. 0. 0. 0.34056989]]
```

Khác với BoW, các vector TF-IDF có trọng số khác nhau cho từng từ, do đó các vector này biểu diễn từng trọng số với kiểu số thực (float) thay vì số nguyên (int). Điều này giúp mô hình phân loại có thể nhận biết được tầm quan trọng của từng từ trong văn bản.

Sentence Embeddings

Sentence Embeddings là một phương pháp mã hóa văn bản nâng cao hơn, nó chuyển đổi toàn bộ câu hoặc đoạn văn thành một vector có kích thước cố định. Các vector này thường được huấn luyện trên các tập dữ liệu lớn và có thể nắm bắt được ngữ nghĩa của câu, từ đó cải thiện khả năng phân loại.

Chúng ta sẽ implement class `EmbeddingVectorizer` để mã hóa văn bản thành các vector embeddings. Class này sẽ sử dụng mô hình pre-trained từ thư viện `sentence-transformers` để chuyển đổi văn bản thành vector.

Xây dựng EmbeddingVectorizer để mã hóa văn bản

```
1 class EmbeddingVectorizer:
2     def __init__(
3         self,
4         model_name: str = 'intfloat/multilingual-e5-base',
5         normalize: bool = True
6     ):
7         self.model = SentenceTransformer(model_name)
8         self.normalize = normalize
9
10    def _format_inputs(
11        self,
12        texts: List[str],
13        mode: Literal['query', 'passage']
14    ) -> List[str]:
15        if mode not in {"query", "passage"}:
16            raise ValueError("Mode must be either 'query' or 'passage'")
17        return [f"{mode}: {text.strip()}" for text in texts]
18
19    def transform(
20        self,
21        texts: List[str],
22        mode: Literal['query', 'passage'] = 'query'
23    ) -> List[List[float]]:
24        if mode == 'raw':
25            inputs = texts
26        else:
27            inputs = self._format_inputs(texts, mode)
28
29        embeddings = self.model.encode(inputs, normalize_embeddings=
```

```

            self.normalize)
30     return embeddings.tolist()
31
32     def transform_numpy(
33         self,
34         texts: List[str],
35         mode: Literal['query', 'passage'] = 'query'
36     ) -> np.ndarray:
37         return np.array(self.transform(texts, mode=mode))

```

Class `EmbeddingVectorizer` được xây dựng với các methods sau:

- `__init__`: Khởi tạo mô hình SentenceTransformer với tên mô hình và tùy chọn chuẩn hóa.
- `_format_inputs`: Định dạng đầu vào cho mô hình, thêm tiền tố "query" hoặc "passage" vào văn bản.
- `transform`: Chuyển đổi danh sách văn bản thành các vector embeddings.
- `transform_numpy`: Chuyển đổi danh sách văn bản thành các vector embeddings dưới dạng numpy array.

Tương tự hai phương pháp trên, chúng ta sẽ sử dụng class `EmbeddingVectorizer` để mã hóa văn bản trong bộ dữ liệu của mình. Đoạn code sau sẽ thực hiện việc này:

```

1 docs = [
2     "I am going to school to study for the final exam.",
3     "The weather is nice today and I feel happy.",
4     "I love programming in Python and exploring new libraries.",
5     "Data science is an exciting field with many opportunities.",
6 ]
7
8 vectorizer = EmbeddingVectorizer()
9 embeddings = vectorizer.transform(docs)
10
11 for i, emb in enumerate(embeddings):
12     print(f"Embedding for Document {i+1}:")
13     print(emb[:3]) # Print first 10 dimensions for brevity

```

Đoạn code trên sẽ mã hóa các văn bản trong danh sách `docs` thành các vector embeddings và in ra 10 chiều đầu tiên của mỗi vector như sau:

Vector Embeddings khi sử dụng EmbeddingVectorizer

Embedding for Document 1:

-0.014805682934820652, 0.031276583671569824, -0.01615864224731922

Embedding for Document 2:

0.011917386204004288, 0.03327357769012451, -0.025739021599292755

Embedding for Document 3:

0.012662884779274464, 0.03936118632555008, -0.024181174114346504

Embedding for Document 4:

0.0063935937359929085, 0.04922199621796608, -

0.028402965515851974

Các vectors này tương tự như các vectors TF-IDF, nhưng chúng có kích thước cố định và có thể nắm bắt được ngữ nghĩa của câu. Bên cạnh đó, có rất ít từ trong các vectors này có giá trị bằng 0, điều này cho thấy các vectors embeddings có thể biểu diễn tốt hơn ngữ nghĩa của văn bản so với các phương pháp mã hóa khác.

Khai báo và khởi tạo các vectorizers

Chúng ta sẽ khai báo và khởi tạo các vectorizers và huấn luyện chúng trên tập dữ liệu của chúng ta cho từng phương pháp mã hóa văn bản đã đề cập ở trên. Đoạn code sau sẽ thực hiện việc này:

```
1 bow_vectorizer = CountVectorizer()
2 X_train_bow = # Your code here
3 X_test_bow = bow_vectorizer.transform(X_test)
4
5 tfidf_vectorizer = TfidfVectorizer()
6 X_train_tfidf = # Your code here
7 X_test_tfidf = tfidf_vectorizer.transform(X_test)
8
9 embedding_vectorizer = EmbeddingVectorizer()
10 X_train_embeddings = # Your code here
11 X_test_embeddings = embedding_vectorizer.transform(X_test)
12
```

```

13 # convert all to numpy arrays for consistency
14 X_train_bow, X_test_bow = np.array(X_train_bow.toarray()), np.array(
15                                     X_test_bow.toarray())
16 X_train_tfidf, X_test_tfidf = np.array(X_train_tfidf.toarray()), np.
17                                     array(X_test_tfidf.toarray())
18 X_train_embeddings, X_test_embeddings = np.array(X_train_embeddings)
19                                     , np.array(X_test_embeddings)
20
21 # Print shapes of the transformed datasets
22 print(f"Shape of X_train_bow: {X_train_bow.shape}\n")
23 print(f"Shape of X_test_bow: {X_test_bow.shape}\n")
24 print(f"Shape of X_train_tfidf: {X_train_tfidf.shape}\n")
25 print(f"Shape of X_test_tfidf: {X_test_tfidf.shape}\n")
26 print(f"Shape of X_train_embeddings: {X_train_embeddings.shape}\n")
27 print(f"Shape of X_test_embeddings: {X_test_embeddings.shape}\n")

```

Kết quả in ra màn hình lần lượt sẽ là kích thước của các tập dữ liệu đã được mã hóa:

Kích thước của các tập dữ liệu đã được mã hóa

Shape of X_train_bow: (800, 10373)
 Shape of X_test_bow: (200, 10373)

Shape of X_train_tfidf: (800, 10373)
 Shape of X_test_tfidf: (200, 10373)

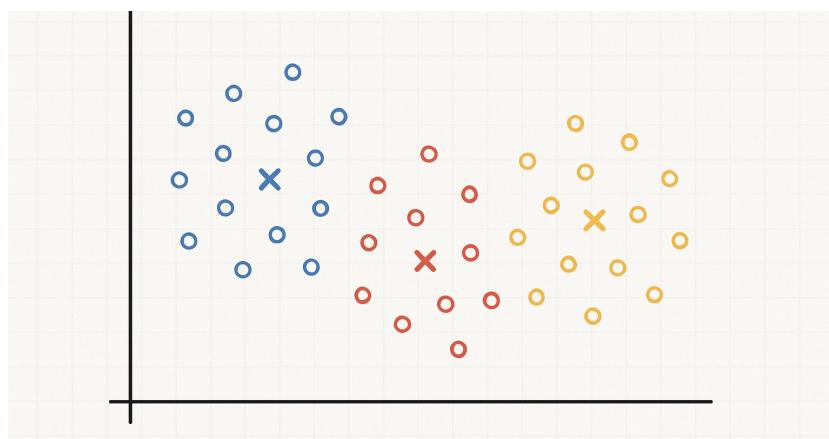
Shape of X_train_embeddings: (800, 768)
 Shape of X_test_embeddings: (200, 768)

Chúng ta thấy rằng các vector BoW và TF-IDF có kích thước bằng nhau, tương ứng với số lượng từ trong từ điển (10373 từ), trong khi các vector embeddings có kích thước cố định là 768, cho thấy chúng được mã hóa thành các vector có kích thước nhỏ hơn nhiều so với BoW và TF-IDF.

14.2.5 Huấn luyện và đánh giá mô hình phân loại

KMeans Clustering

KMean là một thuật toán phân cụm không giám sát, nó sẽ phân chia các vector thành các cụm dựa trên khoảng cách giữa chúng. Chúng ta sẽ sử dụng KMeans để phân cụm các vector BoW, TF-IDF và embeddings đã được mã hóa ở trên.



Hình 14.3: KMean Clustering Algorithm

Đối với mỗi phương pháp mã hóa, chúng ta sẽ khởi tạo một mô hình KMeans với số lượng cụm (clusters) là 5 và huấn luyện nó trên tập dữ liệu huấn luyện. Để thực hiện việc này, chúng ta sẽ viết hàm `train_and_test_kmeans` để huấn luyện mô hình KMeans và in ra các cụm đã được phân chia như sau:

Hàm `train_and_test_kmeans` để huấn luyện mô hình KMeans

```

1 def train_and_test_kmeans(X_train, y_train, X_test, y_test,
                           n_clusters: int):
2     kmeans = # Your code here
3     cluster_ids = kmeans.fit_predict(X_train)
4
5     # Assign label to clusters
6     cluster_to_label = {}
7     for cluster_id in set(cluster_ids):
8         # Get all labels in this cluster

```

```

9     labels_in_cluster = [y_train[i] for i in range(len(y_train))
10        if cluster_ids[i] == cluster_id]
11    most_common_label = Counter(labels_in_cluster).most_common(1)
12        )[0][0]
13    cluster_to_label[cluster_id] = most_common_label
14
15    # Predict labels for test set
16    test_cluster_ids = kmeans.predict(X_test)
17    y_pred = [cluster_to_label[cluster_id] for cluster_id in
18              test_cluster_ids]
19    accuracy = accuracy_score(y_test, y_pred)
20    report = classification_report(y_test, y_pred, target_names=[
21        id_to_label[i] for i in range(len(
22            id_to_label))], output_dict=True)
23
24    return y_pred, accuracy, report

```

Hàm `train_and_test_kmeans` sẽ thực hiện các bước sau:

- Khởi tạo mô hình KMeans với số lượng cụm là `n_clusters`.
- Huấn luyện mô hình trên tập dữ liệu huấn luyện và dự đoán các cụm cho tập dữ liệu huấn luyện.
- Gán nhãn cho các cụm dựa trên nhãn phổ biến nhất trong mỗi cụm.
- Dự đoán nhãn cho tập dữ liệu kiểm tra và tính toán độ chính xác (accuracy) cũng như báo cáo phân loại (classification report).

Sau đó, chúng ta chạy hàm `train_and_test_kmeans` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình KMeans cho các phương pháp mã hóa

```

1  # Train and test K-Means on different vectorized datasets
2 km_bow_labels, km_bow_accuracy, km_bow_report =
3   train_and_test_kmeans(X_train_bow,
4     y_train, X_test_bow, y_test,
5     n_clusters=len(label_to_id))
6 km_tfidf_labels, km_tfidf_accuracy, km_tfidf_report =
7   train_and_test_kmeans(
8     X_train_tfidf, y_train,
9

```

```

        X_test_tfidf, y_test, n_clusters=
        len(label_to_id))
4 km_embeddings_labels, km_embeddings_accuracy, km_embeddings_report =
        train_and_test_kmeans(
        X_train_embeddings, y_train,
        X_test_embeddings, y_test,
        n_clusters=len(label_to_id))

5
6 # Print K-Means results
7 print("Accuracies for K-Means:")
8 print(f"Bag of Words: {km_bow_accuracy:.4f}")
9 print(f"TF-IDF: {km_tfidf_accuracy:.4f}")
10 print(f"Embeddings: {km_embeddings_accuracy:.4f}")

```

Kết quả in ra sẽ là độ chính xác của mô hình KMeans cho từng phương pháp mã hóa:

Độ chính xác của mô hình KMeans cho từng phương pháp mã hóa

Accuracies for K-Means:
 Bag of Words: 0.5600
 TF-IDF: 0.6150
 Embeddings: 0.8400

Trong đó, mô hình KMeans với phương pháp mã hóa embeddings đạt độ chính xác cao nhất là 84%, trong khi phương pháp BoW và TF-IDF chỉ đạt độ chính xác lần lượt là 56% và 61.5%. Điều này chứng minh rằng phương pháp mã hóa embeddings có thể nắm bắt được ngữ nghĩa của văn bản tốt hơn so với các phương pháp mã hóa khác mà tiêu tốn ít tài nguyên hơn (vector embeddings có kích thước nhỏ hơn nhiều so với BoW và TF-IDF). Bên cạnh đó, việc tích hợp thông tin tần suất của từ để đánh giá độ quan trọng của từ trong văn bản cũng giúp cải thiện độ chính xác của mô hình (accuracy của mô hình KMeans với phương pháp mã hóa TF-IDF cao hơn so với BoW).

Cuối cùng, chúng ta vẽ confusion matrix cho từng phương pháp mã hóa để trực quan hóa kết quả phân cụm với hàm `plot_confusion_matrix` như sau:

Vẽ confusion matrix cho từng phương pháp mã hóa với hàm plot_confusion_matrix

```

1 def plot_confusion_matrix(y_true, y_pred, label_list, figure_name="Confusion Matrix", save_path=None):
2     """
3         Plots a confusion matrix with raw counts and normalized values
4             using Seaborn.
5
6     Parameters:
7         y_true (array-like): True class labels.
8         y_pred (array-like): Predicted class labels.
9         label_list (list or dict): Class names (list or dict from ID
10             to name).
11         figure_name (str): Title of the plot.
12         save_path (str, optional): Path to save the figure. If None,
13             the figure will not be saved.
14     """
15     # Compute confusion matrix and normalize
16     cm = confusion_matrix(y_true, y_pred)
17     cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
18
19     # Map class indices to names
20     labels = np.unique(y_true)
21     if isinstance(label_list, dict):
22         class_names = [label_list[i] for i in labels]
23     else:
24         class_names = [label_list[i] for i in labels]
25
26     # Create annotations with raw + normalized values
27     annotations = np.empty_like(cm).astype(str)
28     for i in range(cm.shape[0]):
29         for j in range(cm.shape[1]):
30             raw = cm[i, j]
31             norm = cm_normalized[i, j]
32             annotations[i, j] = f"{raw}\n({norm:.2%})"
33
34     # Plot
35     plt.figure(figsize=(6, 4))
36     sns.heatmap(cm, annot=annotations, fmt="", cmap="Blues",
37                 xticklabels=class_names, yticklabels=class_names,
38                 cbar=False, linewidths=1, linecolor='black')

```

```

36     plt.xlabel('Predicted Label')
37     plt.ylabel('True Label')
38     plt.title(figure_name)
39     plt.tight_layout()
40
41     if save_path:
42         plt.savefig(save_path)
43
44
45     plt.show()

```

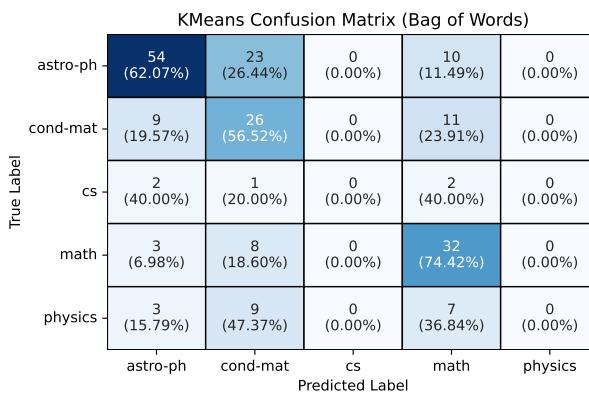
Và sử dụng hàm này để vẽ confusion matrix cho từng phương pháp mã hóa:

```

1 # Draw confusion matrices
2 plot_confusion_matrix(y_test, km_bow_labels, sorted_labels, "KMeans
   Confusion Matrix (Bag of Words)")
3 plot_confusion_matrix(y_test, km_tfidf_labels, sorted_labels, "
   KMeans Confusion Matrix (TF-IDF)")
4 plot_confusion_matrix(y_test, km_embeddings_labels, sorted_labels, "
   KMeans Confusion Matrix (
   Embeddings)")

```

Hình dưới đây là các confusion matrix cho từng phương pháp mã hóa:



(a) Confusion Matrix KMeans với BoW

		KMeans Confusion Matrix (TF-IDF)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	79 (90.80%)	0 (0.00%)	0 (0.00%)	8 (9.20%)	0 (0.00%)
	cond-mat	27 (58.70%)	4 (8.70%)	0 (0.00%)	15 (32.61%)	0 (0.00%)
	cs	2 (40.00%)	0 (0.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	3 (6.98%)	0 (0.00%)	0 (0.00%)	40 (93.02%)	0 (0.00%)
	physics	6 (31.58%)	2 (10.53%)	0 (0.00%)	11 (57.89%)	0 (0.00%)
	Predicted Label	astro-ph	cond-mat	cs	math	physics

(b) Confusion Matrix KMeans với TF-IDF

		KMeans Confusion Matrix (Embeddings)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	84 (96.55%)	0 (0.00%)	0 (0.00%)	3 (3.45%)	0 (0.00%)
	cond-mat	0 (0.00%)	41 (89.13%)	0 (0.00%)	5 (10.87%)	0 (0.00%)
	cs	0 (0.00%)	0 (0.00%)	0 (0.00%)	5 (100.00%)	0 (0.00%)
	math	0 (0.00%)	0 (0.00%)	0 (0.00%)	43 (100.00%)	0 (0.00%)
	physics	2 (10.53%)	10 (52.63%)	0 (0.00%)	7 (36.84%)	0 (0.00%)
	Predicted Label	astro-ph	cond-mat	cs	math	physics

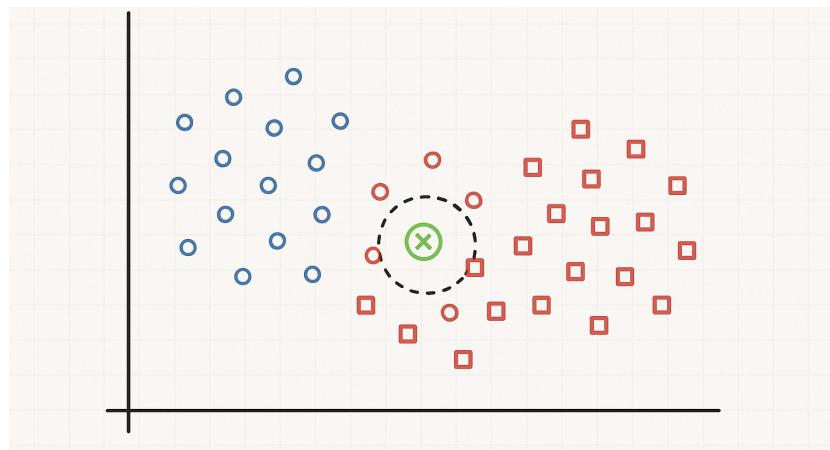
(c) Confusion Matrix KMeans với Embeddings

Hình 14.4: Confusion Matrix cho các phương pháp mã hóa với KMeans

Chúng ta có thể thấy class astro-ph và math luôn được dự đoán đúng nhiều nhất. Trong khi class cs và physics có số lượng dự đoán sai nhiều nhất. Điều này có thể do các class này có nội dung tương tự nhau, hoặc do số lượng mẫu trong các class này không đủ lớn để mô hình học được các đặc trưng phân biệt.

K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) là một thuật toán phân loại đơn giản và hiệu quả, nó phân loại một điểm dữ liệu dựa trên lớp đa số của k láng giềng gần nhất trong không gian đặc trưng. Chúng ta sẽ sử dụng KNN để phân loại các vector BoW, TF-IDF và embeddings đã được mã hóa ở trên.



Hình 14.5: K-Nearest-Neighbor Algorithm

Đầu tiên, chúng ta sẽ thiết kế một hàm `train_and_test_knn` để huấn luyện mô hình KNN và in ra các kết quả phân loại như sau:

Hàm `train_and_test_knn` để huấn luyện mô hình KNN

```

1 def train_and_test_knn(X_train, y_train, X_test, y_test, n_neighbors
2                         : int = 5):
3     knn = # Your code here
4     knn.fit(X_train, y_train)
5
6     # Predict on the test set
7     y_pred = # Your code here
8
9     # Calculate accuracy and classification report
10    accuracy = accuracy_score(y_test, y_pred)
11    report = classification_report(y_test, y_pred, target_names=
12                                    sorted_labels, output_dict=True)

```

```
12 |     return y_pred, accuracy, report
```

Hàm `train_and_test_knn` sẽ thực hiện các bước sau:

- Khởi tạo mô hình KNN với số lượng láng giềng là `n_neighbors`.
- Huấn luyện mô hình trên tập dữ liệu huấn luyện.
- Dự đoán nhãn cho tập dữ liệu kiểm tra và tính toán độ chính xác (accuracy) cũng như báo cáo phân loại (classification report).
- Trả về các nhãn dự đoán, độ chính xác và báo cáo phân loại.

Sau đó, chúng ta sẽ chạy hàm `train_and_test_knn` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình KNN cho các phương pháp mã hóa

```
1 knn_bow_labels, knn_bow_accuracy, knn_bow_report =
      train_and_test_knn(X_train_bow,
                          y_train, X_test_bow, y_test)
2 knn_tfidf_labels, knn_tfidf_accuracy, knn_tfidf_report =
      train_and_test_knn(X_train_tfidf,
                          y_train, X_test_tfidf, y_test)
3 knn_embeddings_labels, knn_embeddings_accuracy,
      knn_embeddings_report =
      train_and_test_knn(
          X_train_embeddings, y_train,
          X_test_embeddings, y_test)
4
5 # Print KNN results
6 print("Accuracies for KNN:")
7 print(f"Bag of Words: {knn_bow_accuracy:.4f}")
8 print(f"TF-IDF: {knn_tfidf_accuracy:.4f}")
9 print(f"Embeddings: {knn_embeddings_accuracy:.4f}")
```

Kết quả in ra sẽ là độ chính xác của mô hình KNN cho từng phương pháp mã hóa:

Độ chính xác của mô hình KNN cho từng phương pháp mã hóa

Accuracies for KNN:

Bag of Words: 0.5350

TF-IDF: 0.8150

Embeddings: 0.8900

Trong đó, mô hình KNN với phương pháp mã hóa embeddings đạt độ chính xác cao nhất là 89%, trong khi phương pháp BoW chỉ đạt độ chính xác là 53.5%. Điểm chung khi so sánh với KMeans là mô hình KNN với phương pháp mã hóa TF-IDF đạt độ chính xác cao hơn so với BoW, cho thấy việc sử dụng tần suất từ để đánh giá độ quan trọng của từ trong văn bản có thể cải thiện độ chính xác của mô hình. Mô hình KNN với phương pháp mã hóa embeddings cũng đạt độ chính xác cao nhất, vượt trội hơn cả KMeans với cùng phương pháp mã hóa (84% của KMeans so với 89% của KNN).

Cuối cùng, chúng ta vẽ confusion matrix cho từng phương pháp mã hóa để trực quan hóa kết quả phân loại với hàm `plot_confusion_matrix` như sau:

Vẽ confusion matrix cho từng phương pháp mã hóa với hàm `plot_confusion_matrix`

```

1 # Draw confusion matrices
2 plot_confusion_matrix(y_test, knn_bow_labels, sorted_labels, "KNN
Confusion Matrix (Bag of Words)",
"pdf/Figures/
knn_bow_confusion_matrix.pdf")
3 plot_confusion_matrix(y_test, knn_tfidf_labels, sorted_labels, "KNN
Confusion Matrix (TF-IDF)", "pdf/
Figures/knn_tfidf_confusion_matrix
.pdf")
4 plot_confusion_matrix(y_test, knn_embeddings_labels, sorted_labels,
"KNN Confusion Matrix (Embeddings)
", "pdf/Figures/
knn_embeddings_confusion_matrix.
pdf")

```

Confusion matrix cho từng phương pháp mã hóa kết hợp với KNN được hiển thị trong Hình 14.6. Chúng ta có thể thấy rằng mô hình KNN với phương pháp mã hóa embeddings đạt được độ chính xác cao nhất, với hầu hết các dự đoán đều đúng. Trong khi đó, mô hình KNN với phương pháp mã hóa

BoW có nhiều dự đoán sai hơn, đặc biệt là đối với các class như astro-ph và cs.

		KNN Confusion Matrix (Bag of Words)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	51 (58.62%)	13 (14.94%)	0 (0.00%)	20 (22.99%)	3 (3.45%)
	cond-mat	4 (8.70%)	20 (43.48%)	0 (0.00%)	18 (39.13%)	4 (8.70%)
	cs	1 (20.00%)	1 (20.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	3 (6.98%)	3 (6.98%)	0 (0.00%)	36 (83.72%)	1 (2.33%)
	physics	4 (21.05%)	3 (15.79%)	0 (0.00%)	12 (63.16%)	0 (0.00%)

(a) Confusion Matrix KNN với BoW

		KNN Confusion Matrix (TF-IDF)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	85 (97.70%)	1 (1.15%)	0 (0.00%)	1 (1.15%)	0 (0.00%)
	cond-mat	3 (6.52%)	42 (91.30%)	0 (0.00%)	0 (0.00%)	1 (2.17%)
	cs	3 (60.00%)	2 (40.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	math	5 (11.63%)	4 (9.30%)	0 (0.00%)	34 (79.07%)	0 (0.00%)
	physics	13 (68.42%)	3 (15.79%)	0 (0.00%)	1 (5.26%)	2 (10.53%)

(b) Confusion Matrix KNN với TF-IDF

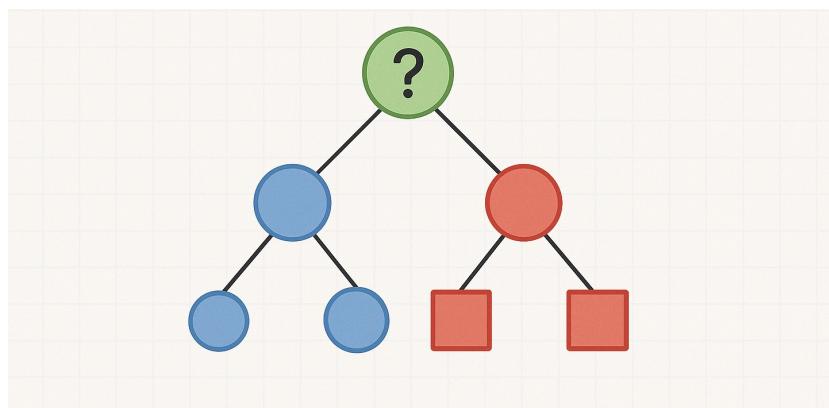
		KNN Confusion Matrix (Embeddings)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	87 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	cond-mat	2 (4.35%)	43 (93.48%)	0 (0.00%)	0 (0.00%)	1 (2.17%)
	cs	1 (20.00%)	1 (20.00%)	1 (20.00%)	2 (40.00%)	0 (0.00%)
	math	2 (4.65%)	1 (2.33%)	0 (0.00%)	40 (93.02%)	0 (0.00%)
	physics	8 (42.11%)	3 (15.79%)	0 (0.00%)	1 (5.26%)	7 (36.84%)
	Predicted Label	astro-ph	cond-mat	cs	math	physics

(c) Confusion Matrix KNN với Embeddings

Hình 14.6: Confusion Matrix cho từng phương pháp mã hóa với KNN

Decision Tree

Decision Tree là một thuật toán phân loại dựa trên cấu trúc cây, nó phân chia dữ liệu thành các nhánh dựa trên các đặc trưng của dữ liệu để đưa ra quyết định cuối cùng. Chúng ta sẽ sử dụng Decision Tree để phân loại các vector BoW, TF-IDF và embeddings đã được mã hóa ở trên.



Hình 14.7: Decision Tree Algorithm

Đầu tiên, chúng ta sẽ thiết kế một hàm `train_and_test_decision_tree` để huấn luyện mô hình Decision Tree và in ra các kết quả phân loại như sau:

Hàm train_and_test_decision_tree để huấn luyện mô hình Decision Tree

```

1 def train_and_test_decision_tree(X_train, y_train, X_test, y_test):
2     dt = # Your code here
3     dt.fit(X_train, y_train)
4
5     # Predict on the test set
6     y_pred = # Your code here
7
8     # Calculate accuracy and classification report
9     accuracy = accuracy_score(y_test, y_pred)
10    report = classification_report(y_test, y_pred, target_names=
11                                    sorted_labels, output_dict=True)
12
13    return y_pred, accuracy, report

```

Hàm `train_and_test_decision_tree` sẽ thực hiện các bước sau:

- Khởi tạo mô hình Decision Tree với một seed cố định để đảm bảo tính tái lập.
- Huấn luyện mô hình trên tập dữ liệu huấn luyện.
- Dự đoán nhãn cho tập dữ liệu kiểm tra và tính toán độ chính xác (accuracy) cũng như báo cáo phân loại (classification report).
- Trả về các nhãn dự đoán, độ chính xác và báo cáo phân loại.

Sau đó, chúng ta sẽ chạy hàm `train_and_test_decision_tree` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình Decision Tree cho các phương pháp mã hóa

```

1 dt_bow_labels, dt_bow_accuracy, dt_bow_report =
2                                train_and_test_decision_tree(
3                                    X_train_bow, y_train, X_test_bow,
4                                    y_test)
5 dt_tfidf_labels, dt_tfidf_accuracy, dt_tfidf_report =
6                                train_and_test_decision_tree(
7                                    X_train_tfidf, y_train,
8                                    X_test_tfidf, y_test)

```

```

3 dt_embeddings_labels, dt_embeddings_accuracy, dt_embeddings_report =
        train_and_test_decision_tree(
            X_train_embeddings, y_train,
            X_test_embeddings, y_test)

4
5 # Print Decision Tree results
6 print("Accuracies for Decision Tree:")
7 print(f"Bag of Words: {dt_bow_accuracy:.4f}")
8 print(f"TF-IDF: {dt_tfidf_accuracy:.4f}")
9 print(f"Embeddings: {dt_embeddings_accuracy:.4f}")

```

Kết quả in ra sẽ là độ chính xác của mô hình Decision Tree cho từng phương pháp mã hóa:

Độ chính xác của mô hình Decision Tree cho từng phương pháp mã hóa

Accuracies for Decision Tree:
 Bag of Words: 0.6350
 TF-IDF: 0.5950
 Embeddings: 0.7150

Trái ngược với KMeans và KNN, mô hình Decision Tree với phương pháp mã hóa BoW đạt độ chính xác cao hơn so với TF-IDF (63.5% so với 59.5%). Điều này có thể do mô hình Decision Tree có khả năng phân chia dữ liệu tốt hơn khi sử dụng các đặc trưng tần suất từ trong BoW. Tuy nhiên, mô hình Decision Tree với phương pháp mã hóa embeddings vẫn đạt độ chính xác cao nhất là 71.5%, cho thấy việc sử dụng embeddings có thể cải thiện độ chính xác của mô hình.

Cuối cùng, chúng ta vẽ confusion matrix cho từng phương pháp mã hóa để trực quan hóa kết quả phân loại với hàm `plot_confusion_matrix` như sau:

Vẽ confusion matrix cho từng phương pháp mã hóa với hàm `plot_confusion_matrix`

```

1 # Draw confusion matrices
2 plot_confusion_matrix(y_test, dt_bow_labels, sorted_labels, "
        Decision Tree Confusion Matrix (
        Bag of Words)", "pdf/Figures/

```

```

3 plot_confusion_matrix(y_test, dt_tfidf_labels, sorted_labels, "dt_bow_confusion_matrix.pdf")
        Decision Tree Confusion Matrix (TF-IDF)", "pdf/Figures/
        dt_tfidf_confusion_matrix.pdf")
4 plot_confusion_matrix(y_test, dt_embeddings_labels, sorted_labels, "Decision Tree Confusion Matrix (Embeddings)", "pdf/Figures/
        dt_embeddings_confusion_matrix.pdf")

```

Đây là các confusion matrix cho từng phương pháp mã hóa với mô hình Decision Tree:

Decision Tree Confusion Matrix (Bag of Words)						
True Label	astro-ph	cond-mat	cs	math	physics	
	astro-ph	69 (79.31%)	11 (12.64%)	1 (1.15%)	2 (2.30%)	4 (4.60%)
	cond-mat	7 (15.22%)	28 (60.87%)	1 (2.17%)	4 (8.70%)	6 (13.04%)
	cs	3 (60.00%)	2 (40.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	math	5 (11.63%)	7 (16.28%)	0 (0.00%)	28 (65.12%)	3 (6.98%)
	physics	6 (31.58%)	5 (26.32%)	1 (5.26%)	5 (26.32%)	2 (10.53%)
Predicted Label						

(a) Confusion Matrix Decision Tree với BoW

		Decision Tree Confusion Matrix (TF-IDF)				
		Predicted Label				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	68 (78.16%)	13 (14.94%)	0 (0.00%)	3 (3.45%)	3 (3.45%)
	cond-mat	12 (26.09%)	19 (41.30%)	0 (0.00%)	10 (21.74%)	5 (10.87%)
	cs	1 (20.00%)	0 (0.00%)	0 (0.00%)	2 (40.00%)	2 (40.00%)
	math	5 (11.63%)	4 (9.30%)	1 (2.33%)	29 (67.44%)	4 (9.30%)
	physics	6 (31.58%)	3 (15.79%)	3 (15.79%)	4 (21.05%)	3 (15.79%)

(b) Confusion Matrix Decision Tree với TF-IDF

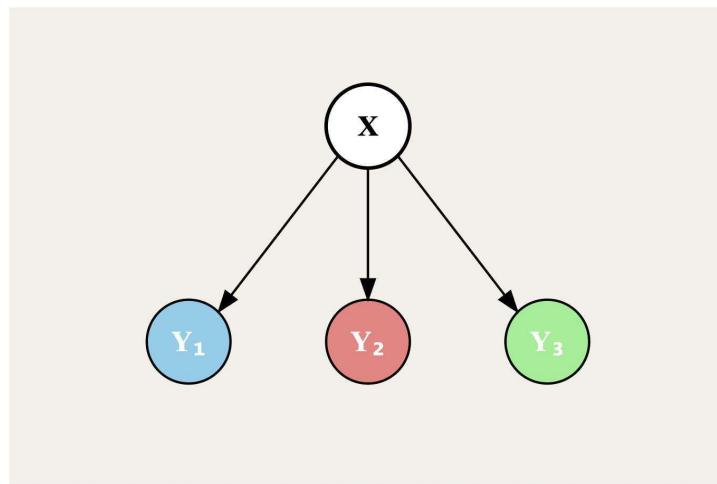
		Decision Tree Confusion Matrix (Embeddings)				
		Predicted Label				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	73 (83.91%)	6 (6.90%)	0 (0.00%)	2 (2.30%)	6 (6.90%)
	cond-mat	5 (10.87%)	32 (69.57%)	3 (6.52%)	1 (2.17%)	5 (10.87%)
	cs	2 (40.00%)	1 (20.00%)	0 (0.00%)	1 (20.00%)	1 (20.00%)
	math	5 (11.63%)	3 (6.98%)	0 (0.00%)	32 (74.42%)	3 (6.98%)
	physics	6 (31.58%)	5 (26.32%)	0 (0.00%)	2 (10.53%)	6 (31.58%)

(c) Confusion Matrix Decision Tree với Embeddings

Hình 14.8: Confusion Matrix cho từng phương pháp mã hóa với mô hình Decision Tree

Naive Bayes

Naive Bayes là một thuật toán phân loại dựa trên định lý Bayes, nó giả định rằng các đặc trưng của dữ liệu là độc lập với nhau. Chúng ta sẽ sử dụng Naive Bayes để phân loại các vector BoW, TF-IDF và embeddings đã được mã hóa ở trên.



Hình 14.9: Naive Bayes Algorithm

Đầu tiên, chúng ta sẽ thiết kế một hàm `train_and_test_naive_bayes` với classs `GaussianNB` để huấn luyện mô hình Naive Bayes và in ra các kết quả phân loại như sau:

Hàm `train_and_test_naive_bayes` để huấn luyện mô hình Naive Bayes

```

1 def train_and_test_naive_bayes(X_train, y_train, X_test, y_test):
2     nb = GaussianNB()
3
4     # Naive Bayes requires input to be in dense format
5     X_train_dense = X_train.toarray() if hasattr(X_train, 'toarray')
6                                         else X_train
6     X_test_dense = X_test.toarray() if hasattr(X_test, 'toarray')
7                                         else X_test
8
8     nb.fit(X_train_dense, y_train)
  
```

```

9      # Predict on the test set
10     y_pred = nb.predict(X_test_dense)
11
12     # Calculate accuracy and classification report
13     accuracy = accuracy_score(y_test, y_pred)
14     report = classification_report(y_test, y_pred, target_names=
15                                     sorted_labels, output_dict=True)
16
17     return y_pred, accuracy, report

```

Hàm `train_and_test_naive_bayes` sẽ thực hiện các bước sau:

- Khởi tạo mô hình Naive Bayes với class `GaussianNB`.
- Chuyển đổi dữ liệu đầu vào sang định dạng dày đặc (dense) nếu cần thiết, vì Naive Bayes yêu cầu dữ liệu đầu vào phải ở định dạng này.
- Huấn luyện mô hình trên tập dữ liệu huấn luyện.
- Dự đoán nhãn cho tập dữ liệu kiểm tra và tính toán độ chính xác (accuracy) cũng như báo cáo phân loại (classification report).
- Trả về các nhãn dự đoán, độ chính xác và báo cáo phân loại.

Sau đó, chúng ta sẽ chạy hàm `train_and_test_naive_bayes` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình Naive Bayes cho các phương pháp mã hóa

```

1 # Train and test Naive Bayes on different vectorized datasets
2 nb_bow_labels, nb_bow_accuracy, nb_bow_report =
3   train_and_test_naive_bayes(
4     X_train_bow, y_train, X_test_bow,
5     y_test)
6 nb_tfidf_labels, nb_tfidf_accuracy, nb_tfidf_report =
7   train_and_test_naive_bayes(
8     X_train_tfidf, y_train,
9     X_test_tfidf, y_test)
10 nb_embeddings_labels, nb_embeddings_accuracy, nb_embeddings_report =
11   train_and_test_naive_bayes(
12     X_train_embeddings, y_train,
13     y_test)

```

```

5                                         X_test_embeddings, y_test)
6 # Print Naive Bayes results
7 print("Accuracies for Naive Bayes:")
8 print(f"Bag of Words: {nb_bow_accuracy:.4f}")
9 print(f"TF-IDF: {nb_tfidf_accuracy:.4f}")
10 print(f"Embeddings: {nb_embeddings_accuracy:.4f}")

```

Kết quả in ra sẽ là độ chính xác của mô hình Naive Bayes cho từng phương pháp mã hóa:

Độ chính xác của mô hình Naive Bayes cho từng phương pháp mã hóa

Accuracies for Naive Bayes:
 Bag of Words: 0.8500
 TF-IDF: 0.8300
 Embeddings: 0.8900

Trong đó, mô hình Naive Bayes với phương pháp mã hóa embeddings đạt độ chính xác cao nhất là 89%, trong khi phương pháp BoW và TF-IDF đạt độ chính xác lần lượt là 85% và 83%. Điều này cho thấy mô hình Naive Bayes có thể hoạt động tốt với các phương pháp mã hóa khác nhau, nhưng vẫn có xu hướng hoạt động tốt hơn với phương pháp mã hóa embeddings (giống với các mô hình đã thí nghiệm phía trên).

Cuối cùng, chúng ta vẽ confusion matrix cho từng phương pháp mã hóa để trực quan hóa kết quả phân loại với hàm `plot_confusion_matrix` như sau:

Vẽ confusion matrix cho từng phương pháp mã hóa với hàm `plot_confusion_matrix`

```

1 # Draw confusion matrices
2 plot_confusion_matrix(y_test, nb_bow_labels, sorted_labels, "Naive
   Bayes Confusion Matrix (Bag of
   Words)", "pdf/Figures/
   nb_bow_confusion_matrix.pdf")
3 plot_confusion_matrix(y_test, nb_tfidf_labels, sorted_labels, "Naive
   Bayes Confusion Matrix (TF-IDF)", |

```

```

    "pdf/Figures/
nb_tfidf_confusion_matrix.pdf")
4 plot_confusion_matrix(y_test, nb_embeddings_labels, sorted_labels, "
Naive Bayes Confusion Matrix (
Embeddings)", "pdf/Figures/
nb_embeddings_confusion_matrix.pdf"
")

```

Đây là các confusion matrix cho từng phương pháp mã hóa với mô hình Naive Bayes:

Naive Bayes Confusion Matrix (Bag of Words)					
True Label	astro-ph	0 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	cond-mat	4 (8.70%)	41 (89.13%)	0 (0.00%)	1 (2.17%)
	cs	3 (60.00%)	1 (20.00%)	0 (0.00%)	1 (20.00%)
	math	5 (11.63%)	1 (2.33%)	0 (0.00%)	37 (86.05%)
	physics	11 (57.89%)	3 (15.79%)	0 (0.00%)	0 (0.00%)
	Predicted Label	astro-ph	cond-mat	cs	math

u'

(a) Confusion Matrix Naive Bayes với BoW

Naive Bayes Confusion Matrix (TF-IDF)					
True Label	astro-ph	0 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	cond-mat	4 (8.70%)	39 (84.78%)	0 (0.00%)	1 (2.17%)
	cs	2 (40.00%)	2 (40.00%)	0 (0.00%)	1 (20.00%)
	math	7 (16.28%)	1 (2.33%)	0 (0.00%)	35 (81.40%)
	physics	12 (63.16%)	2 (10.53%)	0 (0.00%)	0 (0.00%)
	Predicted Label	astro-ph	cond-mat	cs	math

(b) Confusion Matrix Naive Bayes với TF-IDF

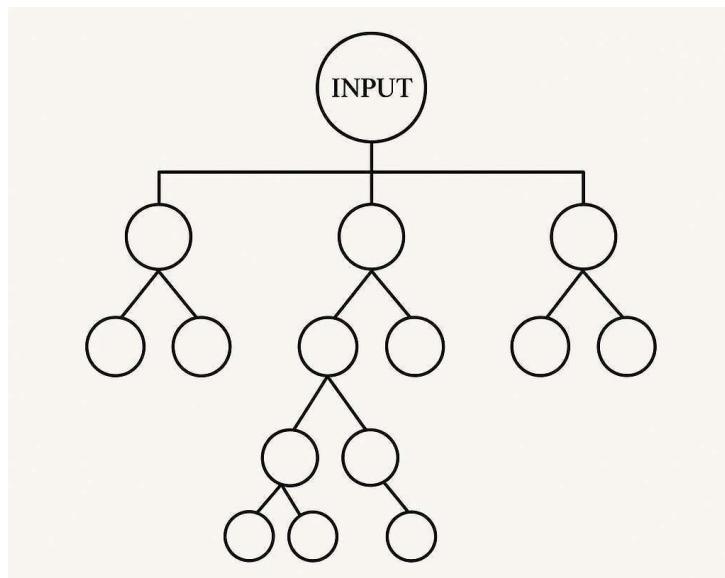
		Naive Bayes Confusion Matrix (Embeddings)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	81 (93.10%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	6 (6.90%)
	cond-mat	0 (0.00%)	43 (93.48%)	0 (0.00%)	0 (0.00%)	3 (6.52%)
	cs	0 (0.00%)	0 (0.00%)	1 (20.00%)	3 (60.00%)	1 (20.00%)
	math	0 (0.00%)	0 (0.00%)	0 (0.00%)	42 (97.67%)	1 (2.33%)
	physics	1 (5.26%)	5 (26.32%)	0 (0.00%)	2 (10.53%)	11 (57.89%)
	Predicted Label	astro-ph	cond-mat	cs	math	physics

(c) Confusion Matrix Naive Bayes với Embeddings

Hình 14.10: Confusion Matrix cho từng phương pháp mã hóa với mô hình Naive Bayes

Random Forest

Random Forest là một thuật toán học máy mạnh mẽ, nó cải thiện độ chính xác dự đoán bằng cách kết hợp đầu ra của nhiều decision trees độc lập từ các mẫu ngẫu nhiên của dữ liệu huấn luyện thông qua quá trình bootstrapping. Khi đưa ra dự đoán, Random Forest sẽ tổng hợp kết quả của các decision trees này - bằng majority vote cho các bài toán phân loại hoặc bằng cách lấy trung bình cho các bài toán hồi quy.



Hình 14.11: Random Forest Algorithm

Tương tự như các mô hình trước, chúng ta sẽ thiết kế một hàm `train_and_test_random_forest` để huấn luyện mô hình Random Forest và in ra các kết quả phân loại như sau:

Hàm `train_and_test_random_forest` để huấn luyện mô hình Random Forest

```

1 def train_and_test_random_forest(X_train, y_train, X_test, y_test,
2                                 n_estimators: int = 100):
3     rf = RandomForestClassifier(n_estimators=n_estimators,
4                                random_state=42)
5     rf.fit(X_train, y_train)
6
7     # Predict on the test set
8     y_pred = rf.predict(X_test)
9
10    # Calculate accuracy and classification report
11    accuracy = accuracy_score(y_test, y_pred)
12    report = classification_report(y_test, y_pred, target_names=
13                                    sorted_labels, output_dict=True)
14
15    return y_pred, accuracy, report
  
```

Sau đó, chúng ta sẽ chạy hàm `train_and_test_random_forest` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình Random Forest cho các phương pháp mã hóa

```

1 rf_bow_labels, rf_bow_accuracy, rf_bow_report =
   train_and_test_random_forest(
      X_train_bow, y_train, X_test_bow,
      y_test)
2 rf_tfidf_labels, rf_tfidf_accuracy, rf_tfidf_report =
   train_and_test_random_forest(
      X_train_tfidf, y_train,
      X_test_tfidf, y_test)
3 rf_embeddings_labels, rf_embeddings_accuracy, rf_embeddings_report =
   train_and_test_random_forest(
      X_train_embeddings, y_train,
      X_test_embeddings, y_test)
4
5 # Print Random Forest results
6 print("Accuracies for Random Forest:")
7 print(f"Bag of Words: {rf_bow_accuracy:.4f}")
8 print(f"TF-IDF: {rf_tfidf_accuracy:.4f}")
9 print(f"Embeddings: {rf_embeddings_accuracy:.4f}")

```

Và đây là kết quả độ chính xác của mô hình Random Forest cho từng phương pháp mã hóa:

Độ chính xác của mô hình Random Forest cho từng phương pháp mã hóa

Accuracies for Random Forest:

Bag of Words: 0.7950

TF-IDF: 0.7750

Embeddings: 0.8550

Trong đó, mô hình Random Forest với phương pháp mã hóa embeddings đạt độ chính xác cao nhất là 85.5%, trong khi phương pháp BoW và TF-IDF đạt độ chính xác lần lượt là 79.5% và 77.5%. Điều này cho thấy mô hình Random Forest có thể hoạt động tốt với các phương pháp mã hóa khác nhau, nhưng vẫn có xu hướng hoạt động tốt hơn với phương pháp mã hóa embeddings (giống với các mô hình đã thí nghiệm phía trên).

Random Forest Confusion Matrix (Bag of Words)					
True Label	astro-ph	84 (96.55%)	1 (1.15%)	0 (0.00%)	2 (2.30%)
	cond-mat	11 (23.91%)	34 (73.91%)	0 (0.00%)	1 (2.17%)
	cs	1 (20.00%)	0 (0.00%)	0 (0.00%)	4 (80.00%)
	math	2 (4.65%)	0 (0.00%)	0 (0.00%)	41 (95.35%)
	physics	7 (36.84%)	7 (36.84%)	0 (0.00%)	5 (26.32%)
	Predicted Label	astro-ph	cond-mat	cs	math

(a) Confusion Matrix Random Forest với BoW

Cuối cùng, chúng ta có confusion matrices cho từng phương pháp mã hóa trong Hình 19.4.

Random Forest Confusion Matrix (TF-IDF)					
True Label	astro-ph	84 (96.55%)	0 (0.00%)	0 (0.00%)	3 (3.45%)
	cond-mat	13 (28.26%)	30 (65.22%)	0 (0.00%)	3 (6.52%)
	cs	0 (0.00%)	1 (20.00%)	0 (0.00%)	4 (80.00%)
	math	2 (4.65%)	0 (0.00%)	0 (0.00%)	41 (95.35%)
	physics	7 (36.84%)	7 (36.84%)	0 (0.00%)	5 (26.32%)
	Predicted Label	astro-ph	cond-mat	cs	math

(b) Confusion Matrix Random Forest với TF-IDF

		Random Forest Confusion Matrix (Embeddings)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	87 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	cond-mat	3 (6.52%)	43 (93.48%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	cs	2 (40.00%)	0 (0.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	1 (2.33%)	1 (2.33%)	0 (0.00%)	41 (95.35%)	0 (0.00%)
	physics	5 (26.32%)	11 (57.89%)	0 (0.00%)	3 (15.79%)	0 (0.00%)

(c) Confusion Matrix Random Forest với Embeddings

Hình 14.12: Confusion Matrix cho từng phương pháp mã hóa với mô hình Random Forest

Support Vector Machine

Support Vector Machine (SVM) là một thuật toán học máy có giám sát được sử dụng cho các bài toán phân loại và hồi quy bằng cách tìm một siêu phẳng tối ưu phân tách dữ liệu thành các lớp. SVM tối đa hóa khoảng cách giữa siêu phẳng và các điểm dữ liệu gần nhất, gọi là các vector hỗ trợ. Việc tối đa hóa khoảng cách này cải thiện khả năng tổng quát hóa của mô hình.

Tương tự như các mô hình trước, chúng ta sẽ thiết kế một hàm `train_and_test_svm` để huấn luyện mô hình SVM với linear kernel và in ra các kết quả phân loại như sau:

Hàm train_and_test_svm để huấn luyện mô hình SVM

```

1 def train_and_test_svm(X_train, y_train, X_test, y_test, kernel: str
2                             = 'linear'):
3     svm = SVC(kernel=kernel, random_state=42)
4     svm.fit(X_train, y_train)
5
6     # Predict on the test set
7     y_pred = svm.predict(X_test)

```

```

8     # Calculate accuracy and classification report
9     accuracy = accuracy_score(y_test, y_pred)
10    report = classification_report(y_test, y_pred, target_names=
11                                    sorted_labels, output_dict=True)
12
13    return y_pred, accuracy, report

```

Sau đó, chúng ta sẽ chạy hàm `train_and_test_svm` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình SVM cho các phương pháp mã hóa

```

1 svm_bow_labels, svm_bow_accuracy, svm_bow_report =
2     train_and_test_svm(X_train_bow,
3                         y_train, X_test_bow, y_test)
4 svm_tfidf_labels, svm_tfidf_accuracy, svm_tfidf_report =
5     train_and_test_svm(X_train_tfidf,
6                         y_train, X_test_tfidf, y_test)
7 svm_embeddings_labels, svm_embeddings_accuracy,
8     svm_embeddings_report =
9     train_and_test_svm(
10        X_train_embeddings, y_train,
11        X_test_embeddings, y_test)
12
13 # Print SVM results
14 print("Accuracies for SVM:")
15 print(f"Bag of Words: {svm_bow_accuracy:.4f}")
16 print(f"TF-IDF: {svm_tfidf_accuracy:.4f}")
17 print(f"Embeddings: {svm_embeddings_accuracy:.4f}")

```

Và đây là kết quả độ chính xác của mô hình SVM cho từng phương pháp mã hóa:

Độ chính xác của mô hình SVM cho từng phương pháp mã hóa

Accuracies for SVM:
 Accuracies for SVM: Bag of Words: 0.8150
 TF-IDF: 0.8800
 Embeddings: 0.8650

		SVM Confusion Matrix (Bag of Words)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	79 (90.80%)	4 (4.60%)	0 (0.00%)	3 (3.45%)	1 (1.15%)
	cond-mat	1 (2.17%)	39 (84.78%)	0 (0.00%)	2 (4.35%)	4 (8.70%)
	cs	1 (20.00%)	1 (20.00%)	0 (0.00%)	2 (40.00%)	1 (20.00%)
	math	1 (2.33%)	2 (4.65%)	1 (2.33%)	38 (88.37%)	1 (2.33%)
	physics	3 (15.79%)	6 (31.58%)	0 (0.00%)	3 (15.79%)	7 (36.84%)

(a) Confusion Matrix SVM với BoW

Kết quả bất ngờ là mô hình SVM với phương pháp mã hóa TF-IDF đạt độ chính xác cao nhất là 88%, trong khi phương pháp embeddings và BoW đạt độ chính xác lần lượt là 86.5% và 81.5%. Điều này cho thấy mô hình SVM có thể hoạt động tốt với các phương pháp mã hóa khác nhau (accuracy đều trên 80%), nhưng trong trường hợp này, TF-IDF lại là phương pháp mã hóa hiệu quả nhất khi kết hợp với SVM.

Cuối cùng, chúng ta có confusion matrices cho từng phương pháp mã hóa trong Hình 14.13.

		SVM Confusion Matrix (TF-IDF)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	86 (98.85%)	0 (0.00%)	0 (0.00%)	1 (1.15%)	0 (0.00%)
	cond-mat	1 (2.17%)	44 (95.65%)	0 (0.00%)	0 (0.00%)	1 (2.17%)
	cs	2 (40.00%)	0 (0.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	0 (0.00%)	0 (0.00%)	0 (0.00%)	43 (100.00%)	0 (0.00%)
	physics	3 (15.79%)	9 (47.37%)	0 (0.00%)	4 (21.05%)	3 (15.79%)

(b) Confusion Matrix SVM với TF-IDF

		SVM Confusion Matrix (Embeddings)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	86 (98.85%)	0 (0.00%)	0 (0.00%)	1 (1.15%)	0 (0.00%)
	cond-mat	0 (0.00%)	45 (97.83%)	0 (0.00%)	1 (2.17%)	0 (0.00%)
	cs	1 (20.00%)	1 (20.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	0 (0.00%)	1 (2.33%)	0 (0.00%)	42 (97.67%)	0 (0.00%)
	physics	4 (21.05%)	13 (68.42%)	0 (0.00%)	2 (10.53%)	0 (0.00%)

(c) Confusion Matrix SVM với Embeddings

Hình 14.13: Confusion Matrix cho từng phương pháp mã hóa với mô hình SVM

AdaBoost

AdaBoost, hay Adaptive Boosting, là một thuật toán học máy kết hợp nhiều mô hình học đơn giản và "yếu" để tạo ra một mô hình "mạnh" có độ chính xác cao. Nó hoạt động bằng cách huấn luyện lặp đi lặp lại các mô hình yếu này, với mỗi mô hình tiếp theo tập trung nhiều hơn vào các ví dụ bị phân loại sai từ các mô hình trước đó bằng cách tăng trọng số của chúng. Dự đoán cuối cùng là trung bình có trọng số của các dự đoán từ tất cả các mô hình yếu.

Chúng ta xây dựng hàm huấn luyện và kiểm thử mô hình AdaBoost trên bộ dữ liệu của chúng ta như sau:

Hàm train_and_test_adaboost để huấn luyện mô hình AdaBoost

```

1 def train_and_test_adaboost(X_train, y_train, X_test, y_test,
                                n_estimators: int = 50):
2     ada = AdaBoostClassifier(n_estimators=n_estimators, random_state
3                             =42)
4     ada.fit(X_train, y_train)
5
6     # Predict on the test set
7     y_pred = ada.predict(X_test)

```

```

7      # Calculate accuracy and classification report
8      accuracy = accuracy_score(y_test, y_pred)
9      report = classification_report(y_test, y_pred, target_names=
10                                     sorted_labels, output_dict=True)
11
12     return y_pred, accuracy, report

```

Sau đó, chúng ta sẽ chạy hàm `train_and_test_adaboost` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình AdaBoost cho các phương pháp mã hóa

```

1 ada_bow_labels, ada_bow_accuracy, ada_bow_report =
2   train_and_test_adaboost(
3     X_train_bow, y_train, X_test_bow,
4     y_test)
5 ada_tfidf_labels, ada_tfidf_accuracy, ada_tfidf_report =
6   train_and_test_adaboost(
7     X_train_tfidf, y_train,
8     X_test_tfidf, y_test)
9 ada_embeddings_labels, ada_embeddings_accuracy,
10  ada_embeddings_report =
11    train_and_test_adaboost(
12      X_train_embeddings, y_train,
13      X_test_embeddings, y_test)

14
15 # Print AdaBoost results
16 print("Accuracies for AdaBoost:")
17 print(f"Bag of Words: {ada_bow_accuracy:.4f}")
18 print(f"TF-IDF: {ada_tfidf_accuracy:.4f}")
19 print(f"Embeddings: {ada_embeddings_accuracy:.4f}")

```

Và đây là kết quả độ chính xác của mô hình AdaBoost cho từng phương pháp mã hóa:

		AdaBoost Confusion Matrix (Bag of Words)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	61 (70.11%)	21 (24.14%)	0 (0.00%)	4 (4.60%)	1 (1.15%)
	cond-mat	13 (28.26%)	27 (58.70%)	0 (0.00%)	5 (10.87%)	1 (2.17%)
	cs	2 (40.00%)	0 (0.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	3 (6.98%)	9 (20.93%)	0 (0.00%)	31 (72.09%)	0 (0.00%)
	physics	6 (31.58%)	6 (31.58%)	0 (0.00%)	6 (31.58%)	1 (5.26%)

(a) Confusion Matrix AdaBoost với BoW

Độ chính xác của mô hình AdaBoost cho từng phương pháp mã hóa

Accuracies for AdaBoost:

Bag of Words: 0.6000

TF-IDF: 0.5700

Embeddings: 0.6200

Chúng ta có thể thấy kết quả đánh giá mô hình AdaBoost không được như mong đợi, với độ chính xác chỉ đạt khoảng 60% cho phương pháp BoW và embeddings, và thấp hơn một chút với TF-IDF là 57%. Kết quả này thấp hơn các mô hình machine learning trước đó như Decision Tree, Naive Bayes và SVM. Để cải thiện, chúng ta có thể tối ưu các hyperparameters của mô hình AdaBoost hoặc thử các thuật toán boosting khác như Gradient Boosting hoặc XGBoost.

Cuối cùng, chúng ta có confusion matrices cho từng phương pháp mã hóa trong Hình 19.5.

		AdaBoost Confusion Matrix (TF-IDF)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	77 (88.51%)	3 (3.45%)	0 (0.00%)	7 (8.05%)	0 (0.00%)
	cond-mat	29 (63.04%)	10 (21.74%)	0 (0.00%)	7 (15.22%)	0 (0.00%)
	cs	1 (20.00%)	1 (20.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	13 (30.23%)	3 (6.98%)	0 (0.00%)	27 (62.79%)	0 (0.00%)
	physics	9 (47.37%)	4 (21.05%)	0 (0.00%)	6 (31.58%)	0 (0.00%)

(b) Confusion Matrix AdaBoost với TF-IDF

		AdaBoost Confusion Matrix (Embeddings)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	70 (80.46%)	1 (1.15%)	0 (0.00%)	6 (6.90%)	10 (11.49%)
	cond-mat	0 (0.00%)	30 (65.22%)	0 (0.00%)	9 (19.57%)	7 (15.22%)
	cs	0 (0.00%)	1 (20.00%)	0 (0.00%)	2 (40.00%)	2 (40.00%)
	math	5 (11.63%)	7 (16.28%)	0 (0.00%)	21 (48.84%)	10 (23.26%)
	physics	4 (21.05%)	8 (42.11%)	0 (0.00%)	4 (21.05%)	3 (15.79%)

(c) Confusion Matrix AdaBoost với Embeddings

Hình 14.14: Confusion Matrix cho từng phương pháp mã hóa với mô hình AdaBoost

Gradient Boosting

Gradient boosting là một kỹ thuật học máy tập hợp, xây dựng các cây quyết định (hoặc các mô hình yếu khác) theo trình tự, với mỗi cây mới sửa lỗi của các cây trước đó để tạo thành một mô hình dự đoán mạnh mẽ tổng thể. Nó hoạt động bằng cách lặp đi lặp lại việc tối thiểu hóa một hàm mất mát, sử dụng gradient để hướng dẫn việc huấn luyện mỗi mô hình con mới. Việc sử

dụng gradient boosting có thể tồn kém về mặt tính toán và dễ bị overfitting, đòi hỏi việc điều chỉnh hyperparameter cẩn thận.

Chúng ta xây dựng hàm huấn luyện và kiểm thử mô hình Gradient Boosting trên bộ dữ liệu của chúng ta như sau:

Hàm train_and_test_gradient_boosting để huấn luyện mô hình Gradient Boosting

```

1 def train_and_test_gradient_boosting(X_train, y_train, X_test,
2                                     y_test, n_estimators: int = 100):
3     gb = GradientBoostingClassifier(n_estimators=n_estimators,
4                                     random_state=42)
5     gb.fit(X_train, y_train)
6
7     # Predict on the test set
8     y_pred = gb.predict(X_test)
9
10    # Calculate accuracy and classification report
11    accuracy = accuracy_score(y_test, y_pred)
12    report = classification_report(y_test, y_pred, target_names=
13                                    sorted_labels, output_dict=True)
14
15    return y_pred, accuracy, report

```

Sau đó, chúng ta sẽ chạy hàm `train_and_test_gradient_boosting` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình Gradient Boosting cho các phương pháp mã hóa

```

1 gb_bow_labels, gb_bow_accuracy, gb_bow_report =
2     train_and_test_gradient_boosting(
3         X_train_bow, y_train, X_test_bow,
4         y_test)
5 gb_tfidf_labels, gb_tfidf_accuracy, gb_tfidf_report =
6     train_and_test_gradient_boosting(
7         X_train_tfidf, y_train,
8         X_test_tfidf, y_test)
9 gb_embeddings_labels, gb_embeddings_accuracy, gb_embeddings_report =
10    train_and_test_gradient_boosting(
11        X_train_embeddings, y_train,
12        y_test)

```

```
4                                         X_test_embeddings, y_test)
5 # Print Gradient Boosting results
6 print("Accuracies for Gradient Boosting:")
7 print(f"Bag of Words: {gb_bow_accuracy:.4f}")
8 print(f"TF-IDF: {gb_tfidf_accuracy:.4f}")
9 print(f"Embeddings: {gb_embeddings_accuracy:.4f}")
```

Và đây là kết quả độ chính xác của mô hình Gradient Boosting cho từng phương pháp mã hóa:

Độ chính xác của mô hình Gradient Boosting cho từng phương pháp mã hóa

Accuracies for Gradient Boosting:
Bag of Words: 0.8050
TF-IDF: 0.8000
Embeddings: 0.8650

Kết quả đánh giá mô hình Gradient Boosting cho thấy độ chính xác đạt khoảng 80.5% với phương pháp BoW, 80% với TF-IDF và cao nhất là 86.5% với embeddings. Kết quả này khá tương đồng với các mô hình machine learning trước đó như Decision Tree và SVM, đặc biệt là khi sử dụng embeddings. Để cải thiện hơn nữa, chúng ta có thể tối ưu các hyperparameters của mô hình Gradient Boosting hoặc thử các thuật toán boosting khác như XGBoost hoặc LightGBM.

Cuối cùng, chúng ta có confusion matrices cho từng phương pháp mã hóa trong Hình 19.6.

		Gradient Boosting Confusion Matrix (Bag of Words)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	79 (90.80%)	3 (3.45%)	0 (0.00%)	3 (3.45%)	2 (2.30%)
	cond-mat	4 (8.70%)	38 (82.61%)	0 (0.00%)	3 (6.52%)	1 (2.17%)
	cs	1 (20.00%)	1 (20.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	2 (4.65%)	2 (4.65%)	0 (0.00%)	39 (90.70%)	0 (0.00%)
	physics	2 (10.53%)	6 (31.58%)	0 (0.00%)	6 (31.58%)	5 (26.32%)

(a) Confusion Matrix Gradient Boosting với BoW

		Gradient Boosting Confusion Matrix (TF-IDF)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	83 (95.40%)	2 (2.30%)	0 (0.00%)	2 (2.30%)	0 (0.00%)
	cond-mat	6 (13.04%)	34 (73.91%)	0 (0.00%)	3 (6.52%)	3 (6.52%)
	cs	1 (20.00%)	1 (20.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	3 (6.98%)	2 (4.65%)	0 (0.00%)	38 (88.37%)	0 (0.00%)
	physics	3 (15.79%)	7 (36.84%)	0 (0.00%)	4 (21.05%)	5 (26.32%)

(b) Confusion Matrix Gradient Boosting với TF-IDF

		Gradient Boosting Confusion Matrix (Embeddings)				
		astro-ph	0 (0.00%)	0 (0.00%)	2 (2.30%)	0 (0.00%)
True Label	astro-ph	85 (97.70%)	0 (0.00%)	0 (0.00%)	2 (2.30%)	0 (0.00%)
	cond-mat	0 (0.00%)	44 (95.65%)	0 (0.00%)	0 (0.00%)	2 (4.35%)
	cs	1 (20.00%)	1 (20.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	1 (2.33%)	2 (4.65%)	0 (0.00%)	40 (93.02%)	0 (0.00%)
	physics	6 (31.58%)	7 (36.84%)	0 (0.00%)	2 (10.53%)	4 (21.05%)

(c) Confusion Matrix Gradient Boosting với Embeddings

Hình 14.15: Confusion Matrix cho từng phương pháp mã hóa với mô hình Gradient Boosting

XGBoost

XGBoost là một thư viện học máy mã nguồn mở, được sử dụng rộng rãi vì cách triển khai tối ưu hóa và mở rộng của thuật toán gradient boosting cho các tác vụ như phân loại, hồi quy và xếp hạng. XGBoost nổi tiếng nhờ tốc độ, độ chính xác, cũng như khả năng xử lý các tập dữ liệu lớn thông qua xử lý song song và phân tán.

Chúng ta xây dựng hàm huấn luyện và kiểm thử mô hình XGBoost như sau:

Hàm train_and_test_xgboost để huấn luyện mô hình XGBoost

```

1 def train_and_test_xgboost(X_train, y_train, X_test, y_test,
                           n_estimators: int = 100):
2     xgb = XGBClassifier(
3         n_estimators=n_estimators, use_label_encoder=False,
4         eval_metric='mlogloss',
5         random_state=42
6     )
7     xgb.fit(X_train, y_train)
8
# Predict on the test set
y_pred = xgb.predict(X_test)

```

```

9      # Calculate accuracy and classification report
10     accuracy = accuracy_score(y_test, y_pred)
11     report = classification_report(y_test, y_pred, target_names=
12                               sorted_labels, output_dict=True)
13
14     return y_pred, accuracy, report

```

Tiếp theo, chúng ta sẽ chạy hàm `train_and_test_xgboost` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình XGBoost cho các phương pháp mã hóa

```

1 xgb_bow_labels, xgb_bow_accuracy, xgb_bow_report =
2   train_and_test_xgboost(X_train_bow,
3                           y_train, X_test_bow, y_test)
4 xgb_tfidf_labels, xgb_tfidf_accuracy, xgb_tfidf_report =
5   train_and_test_xgboost(
6     X_train_tfidf, y_train,
7     X_test_tfidf, y_test)
8 xgb_embeddings_labels, xgb_embeddings_accuracy,
9   xgb_embeddings_report =
10  train_and_test_xgboost(
11    X_train_embeddings, y_train,
12    X_test_embeddings, y_test)

# Print XGBoost results
print("Accuracies for XGBoost:")
print(f"Bag of Words: {xgb_bow_accuracy:.4f}")
print(f"TF-IDF: {xgb_tfidf_accuracy:.4f}")
print(f"Embeddings: {xgb_embeddings_accuracy:.4f}")

```

Và đây là kết quả độ chính xác của mô hình XGBoost cho từng phương pháp mã hóa:

Độ chính xác của mô hình XGBoost cho từng phương pháp mã hóa

Accuracies for XGBoost:
 Bag of Words: 0.7850

		XGBoost Confusion Matrix (Bag of Words)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	81 (93.10%)	3 (3.45%)	0 (0.00%)	2 (2.30%)	1 (1.15%)
	cond-mat	8 (17.39%)	35 (76.09%)	0 (0.00%)	3 (6.52%)	0 (0.00%)
	cs	0 (0.00%)	2 (40.00%)	0 (0.00%)	2 (40.00%)	1 (20.00%)
	math	1 (2.33%)	0 (0.00%)	0 (0.00%)	41 (95.35%)	1 (2.33%)
	physics	2 (10.53%)	10 (52.63%)	1 (5.26%)	6 (31.58%)	0 (0.00%)

(a) Confusion Matrix XGBoost với BoW

TF-IDF: 0.7650

Embeddings: 0.8750

Kết quả đánh giá mô hình XGBoost cho thấy độ chính xác đạt khoảng 78.5% với phương pháp BoW, 76.5% với TF-IDF và cao nhất là 87.5% với embeddings. Kết quả này cho thấy XGBoost hoạt động rất tốt với phương pháp mã hóa embeddings, đạt độ chính xác trong top những mô hình hoạt động tốt nhất trong các mô hình đã thử nghiệm. Tuy nhiên, kết quả với BoW và TF-IDF thấp hơn một chút so với các mô hình khác như SVM và Gradient Boosting.

Cuối cùng, chúng ta có confusion matrices cho từng phương pháp mã hóa trong Hình 19.7.

		XGBoost Confusion Matrix (TF-IDF)				
		Predicted Label				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	83 (95.40%)	3 (3.45%)	0 (0.00%)	1 (1.15%)	0 (0.00%)
	cond-mat	9 (19.57%)	32 (69.57%)	0 (0.00%)	3 (6.52%)	2 (4.35%)
	cs	0 (0.00%)	2 (40.00%)	0 (0.00%)	2 (40.00%)	1 (20.00%)
	math	2 (4.65%)	0 (0.00%)	0 (0.00%)	36 (83.72%)	5 (11.63%)
	physics	3 (15.79%)	9 (47.37%)	0 (0.00%)	5 (26.32%)	2 (10.53%)

(b) Confusion Matrix XGBoost với TF-IDF

		XGBoost Confusion Matrix (Embeddings)				
		Predicted Label				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	85 (97.70%)	0 (0.00%)	0 (0.00%)	1 (1.15%)	1 (1.15%)
	cond-mat	0 (0.00%)	44 (95.65%)	0 (0.00%)	1 (2.17%)	1 (2.17%)
	cs	1 (20.00%)	1 (20.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	1 (2.33%)	1 (2.33%)	0 (0.00%)	41 (95.35%)	0 (0.00%)
	physics	4 (21.05%)	8 (42.11%)	0 (0.00%)	2 (10.53%)	5 (26.32%)

(c) Confusion Matrix XGBoost với Embeddings

Hình 14.16: Confusion Matrix cho từng phương pháp mã hóa với mô hình XGBoost

LightGBM

LightGBM (Light Gradient Boosting Machine) được xây dựng dựa trên thuật toán Gradient Boosting Decision Tree (GBDT). Đây là một framework học tăng cường theo gradient, tận dụng các thuật toán học dựa trên cây, đặc biệt là decision tree đã được tối ưu hóa.

Đặc điểm và tối ưu hóa nổi bật của LightGBM được liệt kê như sau:

- Phát triển cây theo lá (Leaf-wise, Best-first): Khác với nhiều phương pháp triển khai GBDT khác thường mở rộng cây theo từng tầng (level-wise), LightGBM mở rộng cây theo lá. Điều này có nghĩa là nó sẽ chọn lá có độ giảm tổn thất (delta loss) lớn nhất để chia, giúp mô hình hội tụ nhanh hơn và tiềm năng đạt độ chính xác cao hơn.
- Thuật toán dựa trên histogram: LightGBM sử dụng thuật toán dựa trên histogram để tìm điểm chia tối ưu, từ đó rút ngắn đáng kể thời gian huấn luyện, đặc biệt với các tập dữ liệu lớn.
- Tiết kiệm bộ nhớ: Thiết kế hướng đến hiệu quả sử dụng bộ nhớ, phù hợp để xử lý dữ liệu ở quy mô lớn.
- Hỗ trợ học song song và phân tán: LightGBM hỗ trợ nhiều chiến lược huấn luyện song song và phân tán, giúp mở rộng khả năng xử lý cho các bài toán Big Data.

Đầu tiên, chúng ta xây dựng hàm huấn luyện và kiểm thử mô hình LightGBM như sau:

Hàm train_and_test_lightgbm để huấn luyện mô hình LightGBM

```

1 def train_and_test_lightgbm(X_train, y_train, X_test, y_test,
2                             n_estimators: int = 100):
3     lgbm = lgb.LGBMClassifier(boosting_type='goss', n_estimators=
4                                n_estimators, random_state=42)
5     lgbm.fit(X_train, y_train)
6
7     # Predict on the test set
8     y_pred = lgbm.predict(X_test)
9
10    # Calculate accuracy and classification report
11    accuracy = accuracy_score(y_test, y_pred)
12    report = classification_report(y_test, y_pred, target_names=
13                                    sorted_labels, output_dict=True)
14
15    return y_pred, accuracy, report

```

Tiếp theo, chúng ta sẽ chạy hàm `train_and_test_lightgbm` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình LightGBM cho các phương pháp mã hóa

```

1 lgbm_bow_labels, lgbm_bow_accuracy, lgbm_bow_report =
   train_and_test_lightgbm(
      X_train_bow, y_train, X_test_bow,
      y_test)
2 lgbm_tfidf_labels, lgbm_tfidf_accuracy, lgbm_tfidf_report =
   train_and_test_lightgbm(
      X_train_tfidf, y_train,
      X_test_tfidf, y_test)
3 lgbm_embeddings_labels, lgbm_embeddings_accuracy,
   lgbm_embeddings_report =
   train_and_test_lightgbm(
      X_train_embeddings, y_train,
      X_test_embeddings, y_test)
4
5 # Print LightGBM results
6 print("Accuracies for LightGBM:")
7 print(f"Bag of Words: {lgbm_bow_accuracy:.4f}")
8 print(f"TF-IDF: {lgbm_tfidf_accuracy:.4f}")
9 print(f"Embeddings: {lgbm_embeddings_accuracy:.4f}")

```

Và đây là kết quả độ chính xác của mô hình LightGBM cho từng phương pháp mã hóa:

Độ chính xác của mô hình LightGBM cho từng phương pháp mã hóa

Accuracies for LightGBM:
 Bag of Words: 0.7500
 TF-IDF: 0.7750
 Embeddings: 0.8800

Kết quả in ra màn hình cho thấy độ chính xác của mô hình LightGBM đạt khoảng 75% với phương pháp BoW, 77.5% với TF-IDF và cao nhất là 88% với embeddings. Tương tự với hầu hết các mô hình khác, mô hình LightGBM hoạt động tốt nhất với phương pháp mã hóa embeddings, đạt độ chính xác trong top những mô hình hoạt động tốt nhất trong các mô hình đã thử nghiệm. Kết quả với BoW và TF-IDF thấp hơn một chút so với các mô hình khác như SVM và Gradient Boosting.

		LightGBM Confusion Matrix (Bag of Words)				
		Predicted Label				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	82 (94.25%)	3 (3.45%)	0 (0.00%)	1 (1.15%)	1 (1.15%)
	cond-mat	9 (19.57%)	32 (69.57%)	0 (0.00%)	3 (6.52%)	2 (4.35%)
	cs	0 (0.00%)	2 (40.00%)	0 (0.00%)	1 (20.00%)	2 (40.00%)
	math	2 (4.65%)	1 (2.33%)	0 (0.00%)	36 (83.72%)	4 (9.30%)
	physics	4 (21.05%)	9 (47.37%)	0 (0.00%)	6 (31.58%)	0 (0.00%)

(a) Confusion Matrix LightGBM với BoW

Cuối cùng, chúng ta có confusion matrices cho từng phương pháp mã hóa trong Hình 19.8.

		LightGBM Confusion Matrix (TF-IDF)				
		Predicted Label				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	82 (94.25%)	4 (4.60%)	0 (0.00%)	1 (1.15%)	0 (0.00%)
	cond-mat	8 (17.39%)	34 (73.91%)	0 (0.00%)	2 (4.35%)	2 (4.35%)
	cs	0 (0.00%)	1 (20.00%)	0 (0.00%)	2 (40.00%)	2 (40.00%)
	math	2 (4.65%)	0 (0.00%)	0 (0.00%)	36 (83.72%)	5 (11.63%)
	physics	3 (15.79%)	9 (47.37%)	0 (0.00%)	4 (21.05%)	3 (15.79%)

(b) Confusion Matrix LightGBM với TF-IDF

		LightGBM Confusion Matrix (Embeddings)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	86 (98.85%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	1 (1.15%)
	cond-mat	1 (2.17%)	44 (95.65%)	0 (0.00%)	0 (0.00%)	1 (2.17%)
	cs	1 (20.00%)	0 (0.00%)	0 (0.00%)	3 (60.00%)	1 (20.00%)
	math	0 (0.00%)	1 (2.33%)	0 (0.00%)	41 (95.35%)	1 (2.33%)
	physics	3 (15.79%)	8 (42.11%)	0 (0.00%)	3 (15.79%)	5 (26.32%)

(c) Confusion Matrix LightGBM với Embeddings

Hình 14.17: Confusion Matrix cho từng phương pháp mã hóa với mô hình LightGBM

14.3 Câu hỏi trắc nghiệm

1. Trong bài toán phân loại abstract bài báo khoa học, mục tiêu chính của mô hình là gì?
 - a) Dịch abstract sang tiếng Việt
 - b) Tóm tắt nội dung abstract
 - c) Dự đoán chủ đề (topic) của abstract
 - d) Phát hiện đạo văn trong abstract
2. Đâu là ba phương pháp mã hóa văn bản được sử dụng trong project?
 - a) BoW, SVM, CNN
 - b) TF-IDF, BoW, Sentence Embeddings
 - c) One-hot Encoding, PCA, LSTM
 - d) TF-IDF, Word2Vec, LDA
3. Trong project, mô hình KMeans đạt độ chính xác cao nhất khi dùng với phương pháp mã hóa nào?
 - a) TF-IDF
 - b) Bag-of-Words
 - c) Sentence Embeddings
 - d) One-hot Encoding
4. Phương pháp mã hóa nào giúp giảm ảnh hưởng của các từ phỏ biến và tăng trọng số của các từ hiếm?
 - a) BoW
 - b) Embeddings
 - c) TF-IDF
 - d) PCA
5. Số lượng mẫu được sử dụng để huấn luyện và đánh giá mô hình là bao nhiêu?
 - a) 500 mẫu
 - b) 1000 mẫu

- c) 2000 mẫu
d) 5000 mẫu
6. Trong các mô hình được sử dụng, mô hình nào là mô hình không giám sát?
a) Decision Tree
b) KMeans
c) KNN
d) Logistic Regression
7. Câu nào đúng về kích thước vector được sinh ra bởi Sentence Embeddings?
a) Luôn bằng số lượng từ trong từ điển
b) Thay đổi theo độ dài văn bản
c) Cố định, thường là 768 chiều
d) Luôn là số nguyên dương
8. Label (nhãn) cho mỗi abstract được lấy từ đâu?
a) Tiêu đề bài báo
b) Trường “submitter”
c) Trường “categories”, lấy phần primary
d) Trường “authors_parsed”
9. Khi sử dụng KNN, phương pháp mã hóa nào đạt độ chính xác cao nhất?
a) Bag-of-Words
b) TF-IDF
c) Sentence Embeddings
d) Không có sự khác biệt
10. Trong pipeline, bước tiền xử lý văn bản **KHÔNG** bao gồm thao tác nào sau đây?
a) Chuyển về chữ thường
b) Xoá ký tự đặc biệt

- c) Thêm tiêu đề bài báo vào abstract
- d) Xoá số và dấu câu

1. Hint: Các file code gợi ý và dữ liệu được lưu trong thư mục có thể được tải [tại đây](#).

2. Solution: Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. Kiến thức cần cho project:

Để hoàn thành và hiểu rõ phần project một cách hiệu quả nhất, các học viên cần nắm rõ các kiến thức được trang bị

4. Đề xuất phương pháp cải tiến project:

Project tiếp cận bài toán phân tích chủ đề văn bản (topic modeling) theo hướng sử dụng các phương pháp biểu diễn vector như Bag of Words (BoW), TF-IDF, S-BERT kết hợp với các thuật toán KNN, K-Means clustering và Decision Tree. Để tiếp tục cải tiến, tối ưu hệ thống; nhóm tác biên soạn gợi ý một số phương pháp như sau:

- **Tiền xử lý nâng cao:** Sử dụng các kỹ thuật tiền xử lý chuyên sâu hơn như loại bỏ từ ngữ ít xuất hiện (rare words), lemmatization thay vì stemming, phát hiện và xử lý stopwords theo ngữ cảnh, và phát hiện cụm từ (phrase detection) để tăng chất lượng vector biểu diễn.
- **Thử nghiệm nhiều phương pháp biểu diễn:**
 - So sánh hiệu quả giữa BoW, TF-IDF và mô hình embedding hiện đại như S-BERT.
 - Kết hợp nhiều phương pháp biểu diễn (feature fusion) để tận dụng ưu điểm của từng phương pháp.
- **Tối ưu thuật toán phân cụm và phân loại:**
 - Với K-Means: Sử dụng phương pháp tìm k tối ưu (Elbow Method, Silhouette Score).
 - Với KNN: Tối ưu số láng giềng (k) và sử dụng weighted KNN để cải thiện độ chính xác.
 - Với Decision Tree: Áp dụng kỹ thuật pruning (cắt tia) để giảm overfitting và cải thiện khả năng tổng quát hóa.

- **Topic Modeling nâng cao:**
 - Áp dụng clustering để nhóm tài liệu trước khi xây dựng Decision Tree nhằm tăng độ rõ ràng của các chủ đề.
 - Sử dụng hierarchical clustering để khám phá các chủ đề con (sub-topics).
- **Đánh giá đa chỉ số:** Áp dụng thêm các chỉ số như Silhouette Score, Davies–Bouldin Index cho clustering, và F1-score, ROC-AUC cho phân loại để đánh giá mô hình toàn diện.
- **Tăng cường dữ liệu (Data Augmentation):** Áp dụng các phương pháp như thay thế từ đồng nghĩa, paraphrasing hoặc back-translation để làm phong phú tập dữ liệu huấn luyện.

Trên đây là một số gợi ý, giúp học viên có thể cải tiến thêm hệ thống để đạt hiệu quả tốt hơn về cả tốc độ xử lý và độ chính xác.

5. Rubric:

Mục	Kiến Thức	Đánh Giá
II.	<ul style="list-style-type: none"> - Lý thuyết về bài toán Topic Modeling trong Natural Language Processing. - Lý thuyết về các phương pháp biểu diễn văn bản: Bag of Words (BoW), TF-IDF, S-BERT và các thuật toán K-Means, KNN, Decision Tree. 	<ul style="list-style-type: none"> - Hiểu được khái niệm topic modeling và các phương pháp biểu diễn văn bản. - Có khả năng sử dụng Python để triển khai các thuật toán K-Means, KNN và Decision Tree cho bài toán topic modeling.
III.	<ul style="list-style-type: none"> - Ứng dụng K-Means clustering để phân cụm chủ đề và Decision Tree để gán nhãn chủ đề. - So sánh hiệu quả các phương pháp biểu diễn văn bản (BoW, TF-IDF, S-BERT) trong bài toán topic modeling. 	<ul style="list-style-type: none"> - Thực hiện phân cụm chủ đề và đánh giá mô hình bằng các chỉ số như Silhouette Score. - Đánh giá, so sánh và chọn phương pháp biểu diễn văn bản phù hợp cho dữ liệu thực tế.

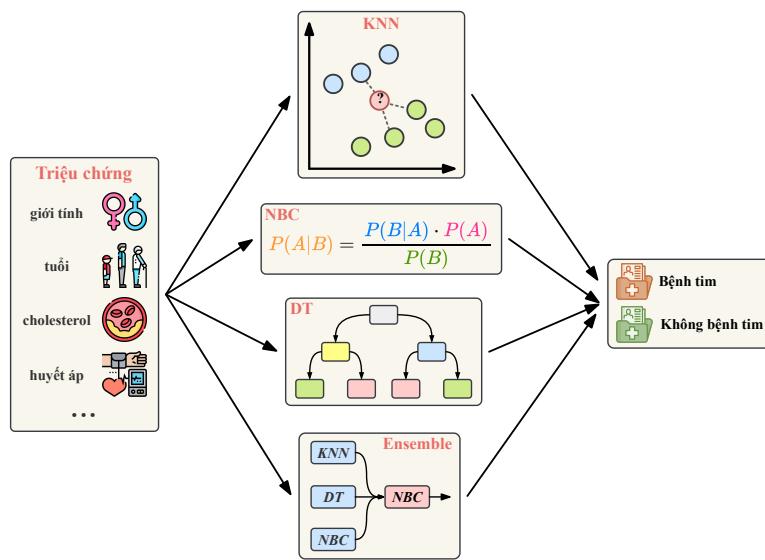
Chương 15

Project 2: Phân loại khả năng mắc bệnh tim dựa vào các triệu chứng (dùng KNN, Decision Tree, và NBC)

15.1 Dẫn nhập

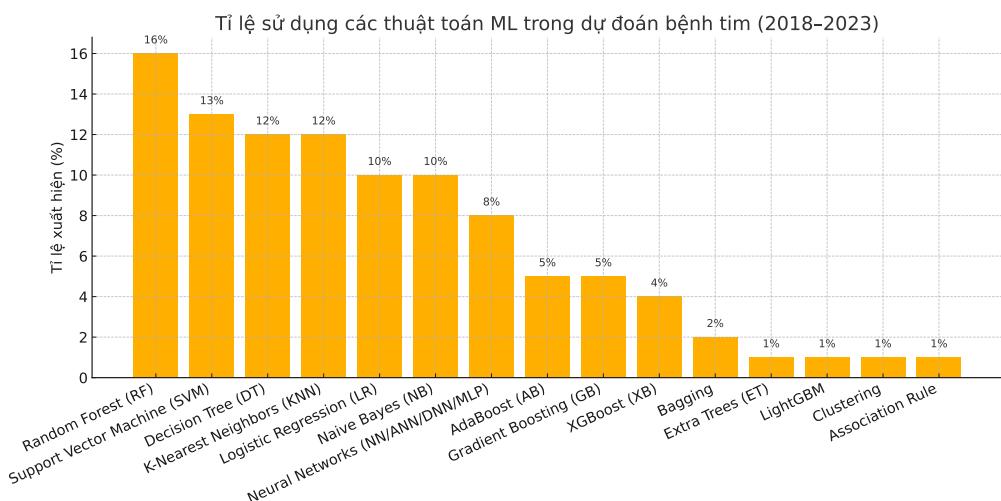
Bệnh tim mạch hiện là nguyên nhân gây tử vong hàng đầu trên thế giới, với khoảng 20,5 triệu ca vào năm 2023, chiếm xấp xỉ 32% tổng số ca tử vong toàn cầu [?, ?, ?]. Tại Việt Nam, ước tính của WHO năm 2016 cho thấy 31% trường hợp tử vong có liên quan đến bệnh tim mạch. Dự báo mới nhất cũng chỉ ra rằng 80% gánh nặng tử vong do bệnh tim sẽ rơi vào các quốc gia có thu nhập từ thấp đến trung bình - nơi việc tầm soát và điều trị còn gặp nhiều hạn chế về chi phí và nguồn lực [?, ?].

Trước thực trạng đó, các giải pháp hỗ trợ chẩn đoán sớm và chính xác đóng vai trò vô cùng quan trọng. Trong những năm gần đây, các mô hình học máy (machine learning) đã nổi lên như một hướng tiếp cận hiệu quả, nhờ khả năng xử lý khối lượng dữ liệu lớn và phát hiện các mối quan hệ ẩn giữa các đặc trưng y tế. Đặc biệt, nhiều nghiên cứu đã chứng minh rằng các thuật toán học máy truyền thống như **Naive Bayes**, **K-Nearest Neighbors**, **Decision Tree**... vẫn được ưa chuộng nhờ tính đơn giản, khả năng diễn giải, và độ chính xác chấp nhận được trong thực tế.



Hình 15.1: Tổng quát dự án dự đoán bệnh tim bằng các phương pháp học máy.

Theo một công trình khảo sát [?] (2025) về bài toán liên quan cung cấp góc nhìn về mức độ phổ biến của các kỹ thuật học máy truyền thống thường được sử dụng (trực quan như hình Hình 15.2 bên dưới). Một cách bất ngờ rằng các mô hình cơ bản như **K-Nearest Neighbors KNN**, **Decision Tree DT** với 12% và **Naive Bayes NB** là 10%, cho thấy các mô hình này vẫn được ưu chuộng vì tính đơn giản mà vẫn đem lại độ chính xác chấp nhận được.



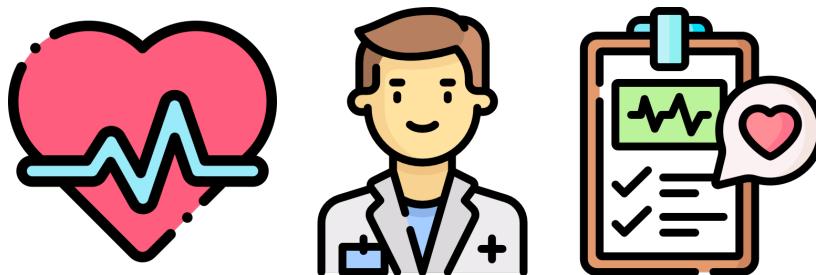
Hình 15.2: Trực quan tần suất xuất hiện của các mô hình học máy truyền thống trong các công trình nghiên cứu về dự đoán bệnh tim từ năm 2018 đến 2023.

Xuất phát từ bối cảnh này, tài liệu sẽ hướng dẫn cách xây dựng và so sánh nhiều mô hình học máy trên bộ dữ liệu bệnh tim. Nội dung tiếp theo bao gồm: Tổng quan dữ liệu, kỹ thuật xử lý đặc trưng, cùng phần giới thiệu và thực hành với các mô hình học máy trên hai phiên bản dữ liệu: bộ dữ liệu gốc và bộ dữ liệu đã được xử lý đặc trưng.

15.2 Cài đặt chương trình

15.2.1 Tổng quan về bộ dữ liệu

“Biết người biết ta, trăm trận trăm thắng”, trong học máy, việc hiểu rõ dữ liệu chính là yếu tố quyết định để xây dựng mô hình hiệu quả. Trong dự án này, chúng ta sử dụng bộ dữ liệu Cleveland Heart Disease, một trong bốn tập con thuộc bộ dữ liệu bệnh tim tổng hợp do UCI Machine Learning Repository cung cấp [?]. So với các tập con khác (Hungary, Switzerland, Long Beach VA), Cleveland được lựa chọn rộng rãi trong nghiên cứu nhờ có ít giá trị thiếu, nhãn phân loại rõ ràng và xuất phát từ nghiên cứu y học kinh điển của Detrano và cộng sự [?].



Cleveland Heart Disease Diagnosis

Hình 15.3: Bộ dữ liệu dự đoán bệnh tim: Cleveland Heart Disease Diagnosis.

Bộ dữ liệu gồm 303 bệnh nhân với 14 đặc trưng y tế phản ánh tình trạng sức khỏe tim mạch, bao gồm các thông tin nhân khẩu học (tuổi, giới tính), các chỉ số lâm sàng (huyết áp, cholesterol, nhịp tim tối đa), các kết quả kiểm tra chuyên sâu (điện tâm đồ, mức độ trầm ST, số mạch máu chính) và thông tin về bệnh lý thalassemia. Mỗi đặc trưng đã được mã hóa dạng số để thuận tiện cho việc huấn luyện mô hình. Bảng dưới đây minh họa một vài mẫu dữ liệu được lấy từ bộ dữ liệu gốc:

Bảng 15.1: Ví dụ một số mẫu dữ liệu từ bộ Cleveland Heart Disease.

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0	0
67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0	1
67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0	1
37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0	0
41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0	0

Việc nắm vững cấu trúc và ý nghĩa của các đặc trưng này không chỉ giúp lựa chọn mô hình phù hợp mà còn đóng vai trò quan trọng trong các bước xử lý đặc trưng và phân tích kết quả. Ở phần tiếp theo, chúng ta sẽ tìm hiểu các kỹ thuật xử lý đặc trưng cần thiết trước khi huấn luyện mô hình.

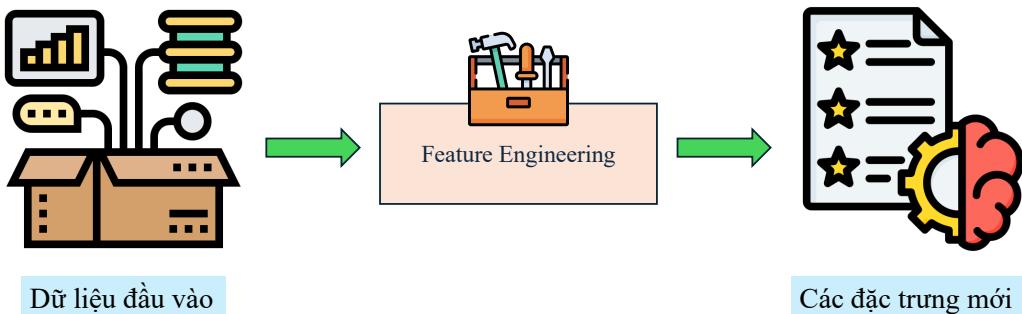
Bảng mô tả đặc trưng bộ dữ liệu Cleveland Heart Disease

Đặc trưng	Mô tả và giá trị mã hóa
age	Tuổi của bệnh nhân (năm).
sex	Giới tính. (1 = nam, 0 = nữ)
cp	Loại đau ngực. (1 = đau thắt ngực điển hình, 2 = không điển hình, 3 = đau không do tim, 4 = không triệu chứng)
trestbps	Huyết áp tâm thu lúc nghỉ (mmHg).
chol	Nồng độ cholesterol huyết thanh (mg/dL).
fbs	Đường huyết lúc đói $> 120 \text{ mg/dL}$. (1 = đúng, 0 = sai)
restecg	Điện tâm đồ lúc nghỉ. (0 = bình thường, 1 = bất thường ST-T, 2 = phì đại thất trái)
thalach	Nhip tim tối đa đạt được (lần/phút).
exang	Đau ngực khi gắng sức. (1 = có, 0 = không)
oldpeak	Mức độ trầm ST do gắng sức so với nghỉ.
slope	Độ dốc đoạn ST. (1 = dốc lên, 2 = bằng phẳng, 3 = dốc xuống)
ca	Số mạch máu chính được nhuộm màu (0–3).
thal	Thalassemia. (3 = bình thường, 6 = tổn thương cố định, 7 = tổn thương có thể đảo ngược)
num	Nhân mục tiêu. (0 = không bệnh, 1–4 = có bệnh)

Các giá trị rời rạc được mã hóa trong ngoặc để phục vụ huấn luyện mô hình.

15.2.2 Kỹ thuật xử lý đặc trưng

Feature engineering là một kỹ thuật đặc biệt có thể biến đổi và bổ sung đặc trưng từ dữ liệu gốc để mô hình có khả năng học tốt hơn. Việc này giúp làm nổi bật các tín hiệu hữu ích, giảm nhiễu, đồng thời tạo ra các đặc trưng mới có giá trị phân tách cao hơn cho quá trình huấn luyện. Trong bài toán dự đoán bệnh tim, chúng ta có thể tiến hành xử lý đặc trưng nhằm cải thiện chất lượng dữ liệu đầu vào, từ đó nâng cao hiệu quả của các mô hình học máy. Việc so sánh kết quả trên tập dữ liệu gốc và tập đã xử lý đặc trưng sẽ cho thấy mức độ cải thiện rõ rệt của kỹ thuật này. **Lưu ý:** Các bạn có thể tải trước bộ dữ liệu sau khi feature engineering tại phần 60.4



Hình 15.4: Minh họa về kỹ thuật feature engineering khi áp dụng vào một bộ dữ liệu nhằm tạo thêm các đặc trưng mới tốt hơn.

Trước tiên, nạp các thư viện xử lý dữ liệu, tiền xử lý, chọn đặc trưng và trực quan, ngoài ra đặt seed để kết quả có thể lặp lại ổn định.

```

1 import os
2 import json
3 import random
4 import numpy as np
5 import pandas as pd
6 import matplotlib.pyplot as plt
7
8 from pathlib import Path
9 from sklearn.pipeline import Pipeline
10 from sklearn.impute import SimpleImputer
11 from sklearn.compose import ColumnTransformer
12 from sklearn.linear_model import LinearRegression

```

```

13 from sklearn.model_selection import train_test_split
14 from sklearn.feature_selection import VarianceThreshold,
15                                     mutual_info_classif
16 from sklearn.preprocessing import (
17     OneHotEncoder, StandardScaler,
18     FunctionTransformer, MinMaxScaler
19 )
20 os.environ["PYTHONHASHSEED"] = "42"
21 np.random.seed(42)
22 random.seed(42)
23 print("Seed: 42")

```

Tiếp theo là tải tệp dữ liệu Cleveland từ Google Drive bằng gdown để chuẩn bị cho các bước làm sạch và tách thành các tập con, các bạn có thể kiểm tra bộ dữ liệu tại phần 60.4

```
1 !gdown 16HPyuXWXPPtt5g3xvS_kR_wXAfjpR1Ju
```

Kế tiếp, đặt tên cột theo các kiểu chuẩn, ép kiểu số cho các trường quan trọng, nhị phân hóa cho cột target (0: không bệnh và > 1: có bệnh).

```

1 DATA_PATH = "cleveland.csv"
2 COLUMNS = ["age", "sex", "cp", "trestbps", "chol", "fbs", "restecg",
3             "thalach", "exang", "oldpeak", "slope", "ca", "thal", "target"]
4
5 raw = pd.read_csv(DATA_PATH, header=None)
6 raw.columns = COLUMNS
7
8 for c in ["age", "trestbps", "chol", "thalach", "oldpeak", "ca", "thal"]:
9     raw[c] = pd.to_numeric(raw[c], errors="coerce")
10
11 raw["target"] = (raw["target"] > 0).astype(int)
12 print("Shape:", raw.shape)
13 display(raw.head())
14 display(raw.isna().sum())

```

Sau đó, tách danh sách cột dạng “số” và cột dạng “phân loại”, làm cơ sở cho

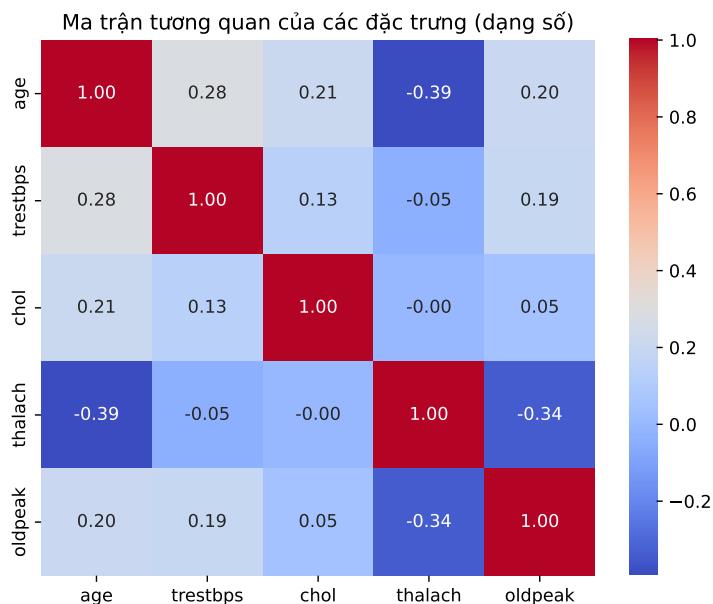
2 luồng tiền xử lý khác nhau.

```

1 numeric_cols = ["age", "trestbps", "chol", "thalach", "oldpeak"]
2 categorical_cols = ["sex", "cp", "fbs", "restecg", "exang", "slope", "ca",
                      "thal"]

```

Tiếp đến là khám phá dữ liệu như trực quan ma trận tương quan, phân phối của các đặc trưng dạng số.



Hình 15.5: Ma trận tương quan giữa các đặc trưng.

```

1 plt.figure(figsize=(6, 5))
2 corr_matrix = raw[numeric_cols].corr()
3 sns.heatmap(corr_matrix, annot=True, cmap="coolwarm", fmt=".2f")
4 plt.title("Ma trận tương quan của các đặc trưng (dạng số)")
5 plt.tight_layout()
6 plt.show()
7
8 sns.pairplot(raw, vars=numeric_cols, hue="target", diag_kind="kde")
9 plt.figure(figsize=(10, 10))
10 plt.suptitle("Phân phối theo từng cặp đặc trưng (dạng số) theo nhãn")

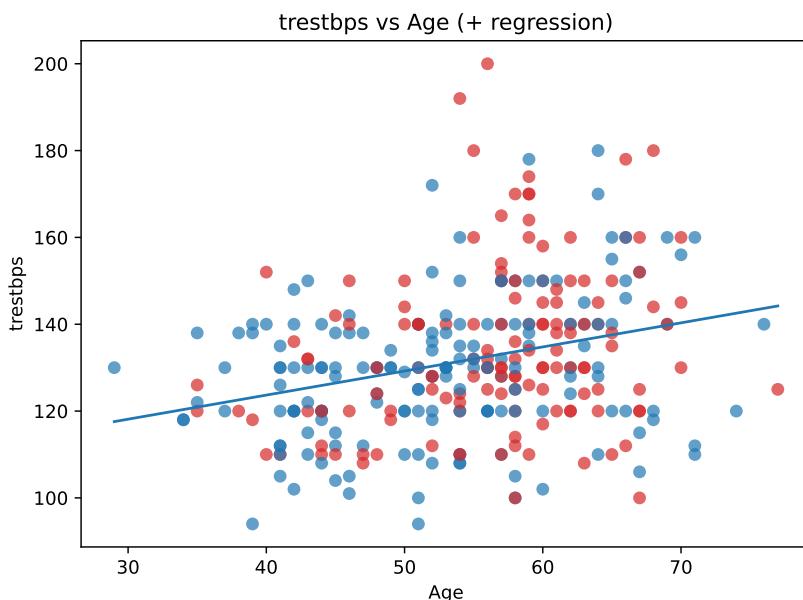
```

```

11 plt.savefig("pairwise_distribution.pdf", bbox_inches="tight")
12 plt.show()

```

Bên cạnh đó, tiến hành trực quan một số mối liên hệ như scatter age với các chỉ số (kèm đường hồi quy), xu hướng theo nhóm tuổi, và tương quan so với age, giúp gợi ý đặc trưng tiềm năng trước khi thực hiện tạo đặc trưng mới.



Hình 15.6: Mối quan hệ giữa tuổi và huyết áp tâm thu lúc nghỉ (trestbps), kèm đường hồi quy tuyến tính.

```

1 X, y = raw.drop(columns=["target"]), raw["target"]
2 colors = np.where(y==0, "tab:blue", "tab:red")
3 metrics = [c for c in ["chol", "trestbps", "thalach"] if c in X.
             columns]
4
5 def scatter_with_regression(xs, ys, xlab, ylab, title):
6     m = xs.notna() & ys.notna()
7     x, yy = xs[m].values, ys[m].values
8     plt.figure()

```

```
9     plt.scatter(xs, ys, c=colors, alpha=0.7)
10    if len(x) > 1:
11        k, b = np.polyfit(x, yy, 1)
12        xline = np.linspace(np.nanmin(x), np.nanmax(x), 100)
13        plt.plot(xline, k*xline + b)
14    plt.xlabel(xlab)
15    plt.ylabel(ylab)
16    plt.title(title)
17    plt.tight_layout()
18    plt.show()
19
20 if "age" in X.columns and metrics:
21     for col in metrics:
22         scatter_with_regression(
23             X["age"], X[col],
24             "Age", col,
25             f"{col} vs Age (+ regression)")
26
27 if "age" in X.columns and metrics:
28     age_bins = pd.cut(X["age"], bins=5)
29     for m in metrics:
30         plt.figure()
31         avg_vals = X.groupby(age_bins, observed=True)[m].mean()
32         plt.plot(range(len(avg_vals)), avg_vals, marker="o", label=f
33                         "Average {m}")
34         plt.xticks(
35             range(len(avg_vals)), map(str, avg_vals.index),
36             rotation=45, ha="right"
37         )
38         plt.xlabel("Age group")
39         plt.ylabel(m)
40         plt.title(f"{m} average per age group")
41         plt.legend()
42         plt.tight_layout()
43         plt.show()
44
45 cols = [c for c in ["age", "chol", "trestbps", "thalach"] if c in X.
46                      columns]
47 if "age" in cols:
48     corr = X[cols].corr(numeric_only=True)[["age"]].drop("age", errors
49                           ="ignore")
50     plt.figure(); corr.plot(kind="bar")
51     plt.ylabel("Correlation with age")
52     plt.title("Correlation of features with Age")
```

```
50     plt.tight_layout()  
51     plt.show()
```

Chia bộ dữ liệu gốc thành 3 tập con train, val, test (tỉ lệ 80:10:10), sau đó biến đổi theo luồng xử lý như sau:

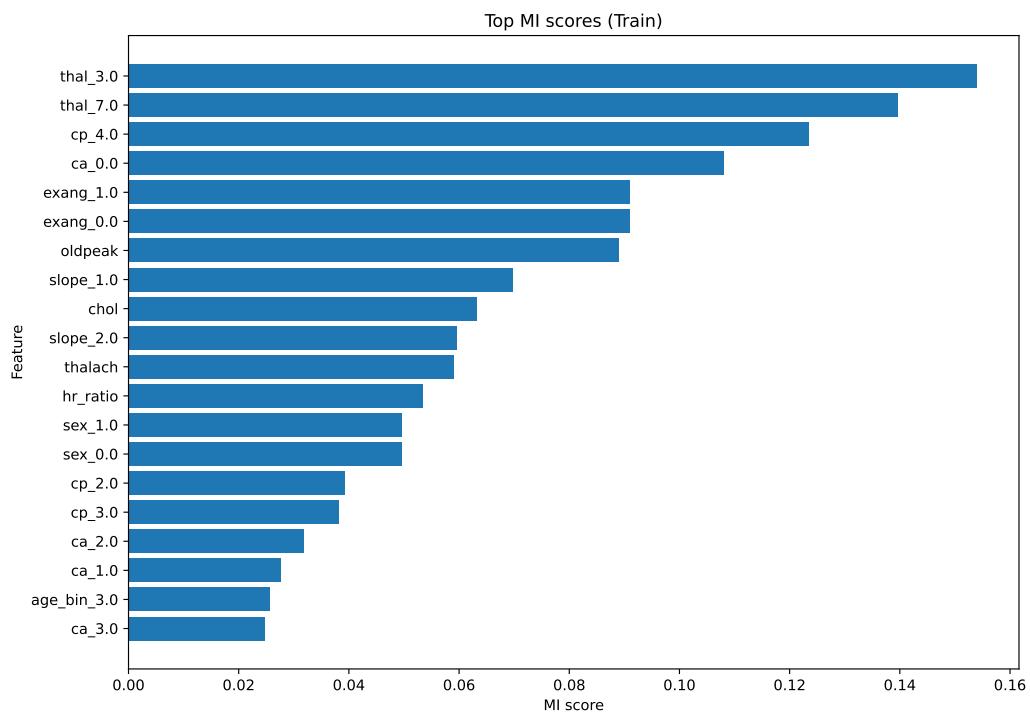
- Nếu cột là dạng “số”: thực hiện thay thế các giá trị thiếu bằng giá trị trung vị (median) trong cột đó, sau đó chuẩn hóa dữ liệu số về phân phối có trung bình là 0 và độ lệch chuẩn bằng 1.
 - Nếu cột là dạng “phân loại”: thay thế giá trị thiếu bằng giá trị xuất hiện nhiều nhất (mode) trong cột đó, sau đó chuẩn hóa dữ liệu theo phương pháp Min-Max.

Sau đó lưu thành các tập gốc gồm raw_train/val/test.csv.

```
1 TARGET = "target"
2 feat_cols = raw.columns.drop(TARGET)
3 X_all, y_all = raw[feat_cols], raw[TARGET]
4 K = len(feat_cols)
5
6 num_proc = Pipeline([('imp', SimpleImputer(strategy="median")),
7                      ('sc', StandardScaler())])
8 cat_proc = Pipeline([('imp', SimpleImputer(strategy="most_frequent"),
9                      ('sc', MinMaxScaler()))])
10
11 preprocess = ColumnTransformer([
12     ("num", num_proc, numeric_cols),
13     ("cat", cat_proc, categorical_cols),
14 ], verbose_feature_names_out=False).set_output(transform="pandas")
15
16 raw_pipeline = Pipeline([('preprocess', preprocess)])
17
18 X_train, X_temp, y_train, y_temp = train_test_split(
19     X_all, y_all, test_size=0.2, stratify=y_all, random_state=42
20 )
21 X_val, X_test, y_val, y_test = train_test_split(
22     X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=42
23 )
```

```
25 X_raw_train = raw_pipeline.fit_transform(X_train, y_train)
26 X_raw_val = raw_pipeline.transform(X_val)
27 X_raw_test = raw_pipeline.transform(X_test)
28
29 out_dir = Path("splits"); out_dir.mkdir(parents=True, exist_ok=True)
30 for name, (X_, y_) in {
31     "raw_train": (X_raw_train, y_train),
32     "raw_val": (X_raw_val, y_val),
33     "raw_test": (X_raw_test, y_test),
34 }.items():
35     pd.concat(
36         [X_, y_.rename(TARGET)], axis=1
37     ).to_csv(out_dir / f"{name}.csv", index=False)
38
39 print(f"Saved RAW splits. K (RAW features) = {K}")
40 display(X_raw_train)
41 display(X_raw_train.isna().sum())
```

Cuối cùng, tạo bộ dữ liệu được xử lý đặc trưng bằng cách sinh thêm đặc trưng mới theo độ tuổi. Sau đó dữ liệu được tiền xử lý theo hai luồng tương tự như trên tập gốc, nhưng các giá trị trong cột dạng “phân loại” được mã hóa theo One-hot. Sau đó, ước lượng mutual information (IM) để xếp hạng và chọn Top-K đặc trưng (với K bằng số lượng đặc trưng trên bộ gốc) và lưu thành các tập `fe_train/val/test.csv`.



Hình 15.7: Các đặc trưng quan trọng theo thang điểm Mutual Information (MI) trên tập huấn luyện.

```

1 def add_new_features(df):
2     df = df.copy()
3     if {"chol", "age"} <= set(df.columns):
4         df["chol_per_age"] = df["chol"]/df["age"]
5     if {"trestbps", "age"} <= set(df.columns):
6         df["bps_per_age"] = df["trestbps"]/df["age"]
7     if {"thalach", "age"} <= set(df.columns):
8         df["hr_ratio"] = df["thalach"]/df["age"]
9     if "age" in df.columns:
10        df["age_bin"] = pd.cut(df["age"], bins=5, labels=False).
11                                astype("category")
12
13 gen_num = ["chol_per_age", "bps_per_age", "hr_ratio"]
14 gen_cat = ["age_bin"]
15 all_nums = [c for c in numeric_cols] + gen_num
16 all_cats = [c for c in categorical_cols] + gen_cat

```

```
17 num_proc = Pipeline([('imp', SimpleImputer(strategy="median")),
18                     ('sc', StandardScaler())])
19 cat_proc = Pipeline([('imp', SimpleImputer(strategy="most_frequent")
20                     ),
21                     ('ohe', OneHotEncoder(handle_unknown="ignore",
22                     sparse_output=False))])
23
24 pre = ColumnTransformer([
25     ("num", num_proc, all_nums),
26     ("cat", cat_proc, all_cats),
27 ], verbose_feature_names_out=False).set_output(transform="pandas")
28
29 fe_pre = Pipeline([
30     ("add", FunctionTransformer(add_new_features, validate=False)),
31     ("pre", pre),
32 ]).set_output(transform="pandas")
33
34 Xt_tr = fe_pre.fit_transform(X_train, y_train)
35 Xt_va = fe_pre.transform(X_val)
36 Xt_te = fe_pre.transform(X_test)
37
38 nz_cols = Xt_tr.columns[Xt_tr.nunique(dropna=False) > 1]
39 Xt_tr, Xt_va, Xt_te = Xt_tr[nz_cols], Xt_va[nz_cols], Xt_te[nz_cols]
40
41 ohe = fe_pre.named_steps["pre"].named_transformers_["cat"].
42         named_steps["ohe"]
43 cat_names = list(ohe.get_feature_names_out(all_cats))
44 is_discrete = np.array([c in cat_names for c in Xt_tr.columns],
45                         dtype=bool)
46
47 mi = mutual_info_classif(Xt_tr.values, y_train.values,
48                           discrete_features=is_discrete, random_state
49                           =42)
50 mi_series = pd.Series(mi, index=Xt_tr.columns).sort_values(ascending
51                           =False)
52
53 K = raw.columns.drop("target").shape[0]
54 topk_cols = list(mi_series.head(K).index)
55
56 fe_tr = Xt_tr[topk_cols].assign(target=y_train.values)
57 fe_va = Xt_va[topk_cols].assign(target=y_val.values)
58 fe_te = Xt_te[topk_cols].assign(target=y_test.values)
```

```

55 out = Path("splits"); out.mkdir(parents=True, exist_ok=True)
56 fe_tr.to_csv(out/"fe_train.csv", index=False)
57 fe_va.to_csv(out/"fe_val.csv", index=False)
58 fe_te.to_csv(out/"fe_test.csv", index=False)
59
60 print(f"Saved FE splits. K (FE features) = {K}")
61 display(pd.Series(topk_cols, name="fe_topk_features").reset_index(
62                                         drop=True))
63
64 N = min(20, len(mi_series))
65 topN = mi_series.head(N).iloc[::-1]
66 plt.figure(figsize=(10, max(6, 0.35*N)))
67 plt.barh(topN.index, topN.values)
68 plt.title("Top MI scores (Train)")
69 plt.xlabel("MI score")
70 plt.ylabel("Feature")
71 plt.tight_layout()
72 plt.show()

```

Sau khi xử lý bộ dữ liệu với các bước trên, chạy code bên dưới để nén folder thành file ZIP **database.zip**, trong đó gồm:

- Bộ dữ liệu gốc: 3 file **fe_train/val/test.csv**.
- Bộ dữ liệu bộ dữ liệu được xử lý đặc trưng: 3 file **fe_train/val/test-.csv**.

```

1 !zip -r dataset.zip splits
2
3 # adding: splits/
4 # adding: splits/fe_train.csv
5 # adding: splits/raw_test.csv
6 # adding: splits/raw_val.csv
7 # adding: splits/fe_test.csv
8 # adding: splits/fe_val.csv
9 # adding: splits/raw_train.csv

```

15.2.3 Naive Bayes Classifier

Giới thiệu

Bắt đầu với Naive Bayes Classifier, một nhóm mô hình phân loại dựa trên định lý Bayes, tính toán xác suất hậu nghiệm của mỗi lớp và gán mẫu dữ liệu vào lớp có xác suất cao nhất.

$$P(A|B) = \frac{\text{Khả năng xảy ra} \quad \text{Xác suất tiên nghiệm}}{\text{Xác suất hậu nghiệm} \quad P(B)} \quad \text{Bằng chứng}$$

Hình 15.8: Công thức Naive Bayes và các thành phần liên quan.

Để đơn giản hóa quá trình tính toán các xác suất phức tạp, mô hình áp dụng giả định “ngây thơ” (naive) rằng các đặc trưng đều vào độc lập có điều kiện khi đã biết nhãn lớp. Nói cách khác, một khi biết bệnh nhân thuộc lớp “mắc bệnh tim” hay “không mắc bệnh tim”, giá trị của một đặc trưng (ví dụ: cholesterol) sẽ không ảnh hưởng đến giá trị của đặc trưng khác (ví dụ: huyết áp) trong việc ước lượng xác suất thuộc lớp. Mặc dù giả định đơn giản, Naive Bayes vẫn cho thấy hiệu quả tốt trong nhiều trường hợp, đặc biệt khi dữ liệu có số lượng mẫu hạn chế và đặc trưng rời rạc.

Triển khai

Lưu ý: Các khối code được đánh dấu “**Tái sử dụng**” nghĩa là được sử dụng nhiều lần mà không cần sửa đổi giữa các mô hình khác nhau trong dự án này.

Bắt đầu bằng việc nạp các thư viện cần thiết và đặt seed để giữ kết quả nhất quán giữa các lần chạy.

Tái sử dụng	
1	# Import thư viện cần thiết
2	import os

```
3 import random
4 import warnings
5 import numpy as np
6 import pandas as pd
7 import seaborn as sns
8 import matplotlib.pyplot as plt
9
10 from collections import Counter
11 from sklearn.cluster import KMeans
12 from sklearn.pipeline import Pipeline
13 from sklearn.naive_bayes import GaussianNB
14 from sklearn.ensemble import StackingClassifier
15 from sklearn.tree import DecisionTreeClassifier
16 from sklearn.neighbors import KNeighborsClassifier
17 from sklearn.preprocessing import StandardScaler
18 from sklearn.metrics import accuracy_score, classification_report
19 from sklearn.model_selection import train_test_split,
                                         StratifiedKFold, cross_val_score
20
21 warnings.filterwarnings("ignore")
22
23 # Đặt seed để đảm bảo kết quả nhất quán giữa các lần chạy
24 SEED = 42
25 os.environ["PYTHONHASHSEED"] = str(SEED)
26 np.random.seed(SEED)
27 random.seed(SEED)
28 print(f"Seed: {SEED}")
```

Tiếp theo, tải bộ dữ liệu đã được xử lý như ở mục 15.2.2, khi này bộ dữ liệu gồm 2 loại: bộ gốc và bộ được xử lý đặc trưng (được đánh dấu bằng ký hiệu “fe”), mỗi bộ gồm các tập train, val, test đã được chia sẵn.

Tái sử dụng

```
1 !gdown 1T6AWCoyeC2MqGvmqZPPcA4ni5Md1k0sI
2 !unzip dataset.zip
```

Kế tiếp, xây dựng một hàm đọc file CSV, tách các đặc trưng X và nhãn y để phục vụ cho việc huấn luyện mô hình, ngoài ra, hiển thị thêm một số thông tin cơ bản của tập dữ liệu.

Tái sử dụng

```

1 def read_csv(file_path):
2     df = pd.read_csv(file_path)
3     display(df.head())
4     X = df.drop("target", axis=1)
5     y = df["target"]
6
7     display(y.value_counts())
8     print("Shape df: ", df.shape)
9     print("Shape X: ", X.shape)
10    print("Shape y: ", y.shape)
11    return X, y

```

Sau đó, lần lượt đọc các tập dữ liệu gốc: raw_train/val/test

Tái sử dụng

```

1 # Tải các bộ dữ liệu gốc đã được xử lý
2 X_train, y_train = read_csv("splits/raw_train.csv")
3 X_val, y_val = read_csv("splits/raw_val.csv")
4 X_test, y_test = read_csv("splits/raw_test.csv")

```

và các tập fe: fe_train/val/test để chuẩn bị cho những thực nghiệm bên dưới.

Tái sử dụng

```

1 # Tải các bộ dữ liệu được áp dụng kỹ thuật tạo đặc trưng
2 X_train_fe, y_train_fe = read_csv("splits/fe_train.csv")
3 X_val_fe, y_val_fe = read_csv("splits/fe_val.csv")
4 X_test_fe, y_test_fe = read_csv("splits/fe_test.csv")

```

Theo sau là 2 hàm evaluate_val và evaluate_test, mục đích chung của 2 này trong toàn bộ các mô hình trong dự án này là:

- Hàm evaluate_val: huấn luyện trên tập train và đánh giá trên tập val,
- Hàm evaluate_test: đánh giá mô hình đã huấn luyện trên tập test.

```

1 def evaluate_val(X_train, y_train, X_val, y_val):
2     # Huấn luyện Naive Bayes
3     nbc_model = GaussianNB()
4     nbc_model.fit(X_train, y_train)
5
6     # Dự đoán và đánh giá trên tập val
7     nb_pred = nbc_model.predict(X_val)
8     nb_accuracy = accuracy_score(y_val, nb_pred)
9
10    print(f"Độ chính xác Naive Bayes: {nb_accuracy:.4f}")
11    print("\nClassification Report:")
12    print(classification_report(y_val, nb_pred))
13    return nbc_model, nb_accuracy
14
15 def evaluate_test(nbc_model, X_test, y_test):
16     # Dự đoán và đánh giá trên tập test
17     nb_test_pred = nbc_model.predict(X_test)
18     nb_test_accuracy = accuracy_score(y_test, nb_test_pred)
19
20     print(f"Độ chính xác Naive Bayes trên tập test: {
21                     nb_test_accuracy:.4f}")
22     print("\nClassification Report:")
23     print(classification_report(y_test, nb_test_pred))
24     return nb_test_accuracy

```

Khi này, tiến hành huấn luyện và đánh giá Naive Bayes trên các tập gốc, lưu lại độ chính xác trên val và test.

```

1 # Huấn luyện và đánh giá với dữ liệu gốc
2 model, accuracy = evaluate_val(X_train, y_train, X_val, y_val)
3 test_accuracy = evaluate_test(model, X_test, y_test)

```

Tương tự như bước trên cho tập fe.

```

1 # Huấn luyện và đánh giá với kỹ thuật tạo đặc trưng
2 model_fe, accuracy_fe = evaluate_val(X_train_fe, y_train_fe,
3                                         X_val_fe, y_val_fe)
3 test_accuracy_fe = evaluate_test(model_fe, X_test_fe, y_test_fe)

```

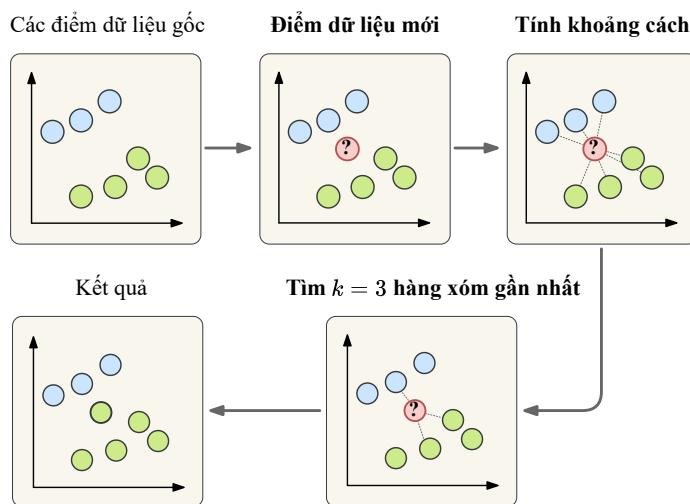
Cuối cùng, vẽ biểu đồ cột so sánh độ chính xác (acc) trên tập val và test cho hai kịch bản (tập gốc và tập fe).

Tái sử dụng

```
1 plt.rcParams["font.family"] = "DejaVu Serif"
2 labels = ["Original Dataset", "Feature Engineering Dataset"]
3 val_accs = [accuracy, accuracy_fe]
4 test_accs = [test_accuracy, test_accuracy_fe]
5
6 x = np.arange(len(labels))
7 width = 0.3
8
9 fig, ax = plt.subplots(figsize=(5, 5))
10 rectss1 = ax.bar(x - width/2, val_accs, width,
11                   label="Validation Accuracy",
12                   color="tab:blue", edgecolor="black", linewidth=1.2)
13 rectss2 = ax.bar(x + width/2, test_accs, width,
14                   label="Test Accuracy",
15                   color="tab:red", edgecolor="black", linewidth=1.2)
16
17 ax.set_ylim(0.5, 1.05)
18 ax.set_ylabel("Accuracy")
19 ax.set_title("Acc trên tập Val và Test")
20 ax.set_xticks(x)
21 ax.set_xticklabels(labels)
22 ax.legend(ncol=2, loc="upper center")
23
24 def autolabel(rects):
25     for rect in rects:
26         h = rect.get_height()
27         ax.annotate(f"{h:.2f}", xy=(rect.get_x() + rect.get_width() / 2,
28                               h),
29                     ha="center", va="bottom")
30
31 autolabel(rectss1)
32 autolabel(rectss2)
33
34 fig.tight_layout()
35 plt.show()
```

15.2.4 K-Nearest Neighbors

Giới thiệu



Hình 15.9: Tổng quan các bước trong mô hình KNN.

Tiếp theo là K-Nearest Neighbors (KNN), một mô hình học có giám sát và phi tham số (non-parametric). Thuật toán này không học một hàm mục tiêu cụ thể, mà lưu trữ toàn bộ tập huấn luyện để sử dụng khi dự đoán. Nguyên tắc hoạt động dựa trên giả định rằng các điểm dữ liệu có đặc trưng tương tự nhau sẽ có xu hướng nằm gần nhau trong không gian đặc trưng. Khi phân loại một mẫu mới, KNN xác định K điểm gần nhất trong tập huấn luyện (các “hàng xóm”) dựa trên một thước đo khoảng cách, sau đó gán nhãn dựa trên đa số phiếu của các hàng xóm này. Lựa chọn giá trị K là yếu tố quan trọng: K quá nhỏ khiến mô hình nhạy cảm với nhiễu, trong khi K quá lớn có thể làm mờ ranh giới giữa các lớp.

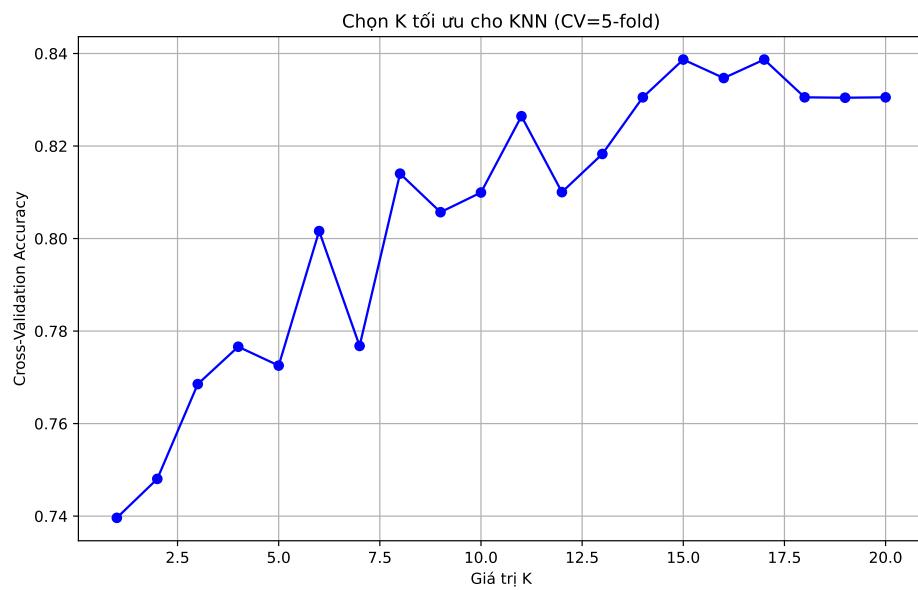
Triển khai

Với mô hình KNN, tái sử dụng các khối code cũ và điều chỉnh một số hàm chính như sau:

- Bổ sung hàm tìm k tối ưu theo phương pháp Stratified K-Fold Cross

Validation,

- Điều chỉnh hàm `evaluate_val` và `evaluate_test` cho KNN.



Hình 15.10: Biểu đồ xác định K tối ưu cho KNN (CV=5-fold).

```

1 def find_optimal_k(X_train, y_train, X_val=None, y_val=None,
2                     k_range=range(1, 21), cv_splits=5):
3     # Tạo bộ chia dữ liệu cross-validation
4     cv = StratifiedKFold(n_splits=cv_splits, shuffle=True,
5                           random_state=SEED)
6     k_scores = []
7
8     # Dánh giá bằng cross-validation trên tập train
9     for k in k_range:
10         knn = KNeighborsClassifier(n_neighbors=k)
11         cv_score = cross_val_score(
12             knn, X_train, y_train,
13             cv=cv, scoring="accuracy", n_jobs=-1)
14         k_scores.append(cv_score.mean())
15
16     # Vẽ biểu đồ chọn k theo CV
17     plt.figure(figsize=(10, 6))

```

```

17     plt.plot(list(k_range), k_scores, "bo-")
18     plt.title(f"Chọn K tối ưu cho KNN (CV={cv_splits}-fold)")
19     plt.xlabel("Giá trị K")
20     plt.ylabel("Cross-Validation Accuracy")
21     plt.grid(True)
22     plt.show()
23
24     # K tối ưu theo CV
25     optimal_k = list(k_range)[int(np.argmax(k_scores))]
26     print(f"K tối ưu (CV): {optimal_k}")
27     return optimal_k
28
29 def evaluate_val(X_train, y_train, X_val, y_val, optimal_k):
30     # Huấn luyện KNN với k tối ưu
31     print(f"Huấn luyện KNN với k tối ưu: {optimal_k}")
32     knn_model = KNeighborsClassifier(n_neighbors=optimal_k)
33     knn_model.fit(X_train, y_train)
34
35     # Dự đoán và đánh giá trên tập val
36     knn_pred = knn_model.predict(X_val)
37     knn_accuracy = accuracy_score(y_val, knn_pred)
38
39     print(f"\nĐộ chính xác KNN trên tập validation: {knn_accuracy:.2f}")
40     print("Classification Report:")
41     print(classification_report(y_val, knn_pred))
42     return knn_model, knn_accuracy
43
44 def evaluate_test(knn_model, X_test, y_test):
45     # Dự đoán và đánh giá trên tập test
46     knn_test_pred = knn_model.predict(X_test)
47     knn_test_accuracy = accuracy_score(y_test, knn_test_pred)
48
49     print(f"\nĐộ chính xác KNN trên tập test: {knn_test_accuracy:.2f}")
50     print("Classification Report:")
51     print(classification_report(y_test, knn_test_pred))
52     return knn_test_accuracy

```

Trong phần huấn luyện mô hình, thực hiện tương tự như mô hình trước nhưng bổ sung thêm việc tìm k tối ưu và sau đó huấn luyện mô hình KNN với giá trị k này.

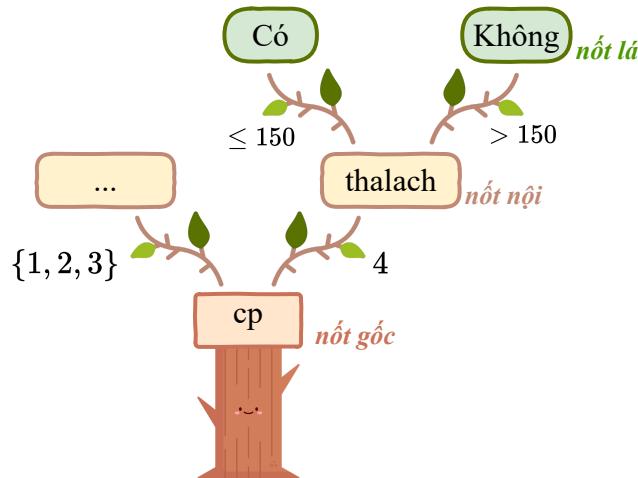
```

1 k_optimal = find_optimal_k(X_train, y_train, X_val, y_val)
2 model, accuracy = evaluate_val(X_train, y_train, X_val, y_val,
                                 k_optimal)
3 test_accuracy = evaluate_test(model, X_test, y_test)
4
5 ###
6
7 k_optimal = find_optimal_k(X_train_fe, y_train_fe, X_val_fe,
                             y_val_fe)
8 model_fe, accuracy_fe = evaluate_val(
9     X_train_fe, y_train_fe,
10    X_val_fe, y_val_fe,
11    k_optimal
12 )
13 test_accuracy_fe = evaluate_test(model_fe, X_test_fe, y_test_fe)

```

15.2.5 Decision Tree

Giới thiệu



Hình 15.11: Minh họa mô hình cây quyết định bằng các thuộc tính của bài toán dự đoán bệnh tim.

Sau KNN, chúng ta chuyển sang Decision Tree, một mô hình học có giám sát sử dụng cấu trúc dạng cây để đưa ra quyết định phân loại. Mô hình chia nhỏ không gian đặc trưng thành các vùng bằng cách áp dụng các quy tắc “nếu-thì” tại mỗi nút. Mỗi **nút nội** biểu diễn một phép kiểm tra trên một đặc trưng, mỗi nhánh thể hiện kết quả của phép kiểm tra đó, và mỗi **nút lá** đại diện cho nhãn dự đoán. Việc xây dựng cây dựa trên việc chọn đặc trưng và ngưỡng phân chia tối ưu tại mỗi bước, nhằm tối đa hóa độ thuần khiết của các nút con, thường đo bằng các chỉ số như **Gini Impurity** hoặc **Information Gain**. Nhờ tính trực quan và khả năng diễn giải cao, Decision Tree là một trong những mô hình nền tảng và dễ hiểu nhất trong học máy.

Triển khai

Với mô hình DT, tái sử dụng các khối code cũ và điều chỉnh một số hàm chính như sau:

- Bổ sung hàm tìm độ sâu (Depth) tối ưu theo phương pháp Stratified K-Fold Cross Validation.
- Điều chỉnh hàm `evaluate_val` và `evaluate_test` cho DT.
- Bổ sung hàm hiển thị các đặc trưng có ảnh hưởng với mô hình.

```

1 def find_optimal_depth(X_train, y_train,
2                         depth_range=range(1, 11), cv_splits=5, min_depth=
2):
3     # Tạo bộ chia dữ liệu cross-validation
4     cv = StratifiedKFold(n_splits=cv_splits, shuffle=True,
5                           random_state=SEED)
6     depth_scores = []
7
8     # Dánh giá bằng cross-validation trên tập train
9     for depth in range(min_depth, depth_range.stop):
10         dt = DecisionTreeClassifier(max_depth=depth, random_state=
11                                       SEED)
12         cv_score = cross_val_score(
13             dt, X_train, y_train,
14             cv=cv, scoring="accuracy", n_jobs=-1)
15         depth_scores.append(cv_score.mean())
16
17     # Vẽ biểu đồ chọn k theo CV

```

```
16     plt.figure(figsize=(10, 6))
17     plt.plot(list(range(min_depth, depth_range.stop)), depth_scores,
18               "bo-")
19     plt.title(f"Chọn Depth tối ưu cho DT (CV={cv_splits}-fold)")
20     plt.xlabel("Giá trị Depth")
21     plt.ylabel("Cross-Validation Accuracy")
22     plt.grid(True)
23     plt.show()
24
25     # K tối ưu theo CV
26     optimal_depth = list(range(min_depth, depth_range.stop))[int(np.
27                           argmax(depth_scores))]
28     print(f"Depth tối ưu (CV): {optimal_depth}")
29     return optimal_depth
30
31 def evaluate_val(X_train, y_train, X_val, y_val, optimal_depth):
32     # Huấn luyện với độ sâu tối ưu
33     dt_model = DecisionTreeClassifier(max_depth=optimal_depth,
34                                       random_state=SEED)
35     dt_model.fit(X_train, y_train)
36
37     # Dự đoán và đánh giá trên tập val
38     dt_pred = dt_model.predict(X_val)
39     dt_accuracy = accuracy_score(y_val, dt_pred)
40     print(f"\nĐộ chính xác Decision Tree trên tập validation: {
41           dt_accuracy:.2f}")
42
43     print("Classification Report:")
44     print(classification_report(y_val, dt_pred))
45     return dt_model, dt_accuracy
46
47 def evaluate_test(dt_model, X_test, y_test):
48     # Dự đoán và đánh giá trên tập test
49     dt_test_pred = dt_model.predict(X_test)
50     dt_test_accuracy = accuracy_score(y_test, dt_test_pred)
51     print(f"\nĐộ chính xác Decision Tree trên tập test: {
52           dt_test_accuracy:.2f}")
53
54     print("Classification Report:")
55     print(classification_report(y_test, dt_test_pred))
56     return dt_test_accuracy
57
58 def display_feature_importance(dt_model, X_train):
59     # Hiển thị các đặc trưng có ảnh hưởng lớn
```

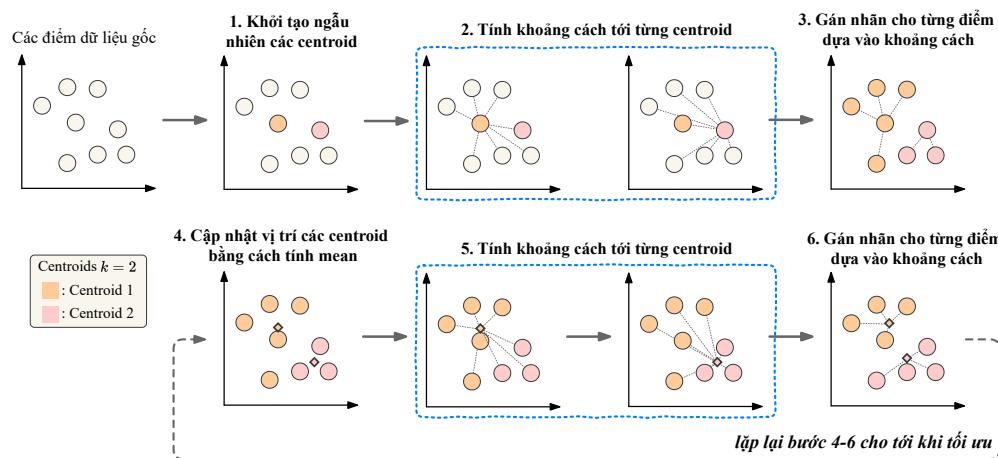
```
55     feature_importance = pd.DataFrame([
56         "feature": X_train.columns,
57         "importance": dt_model.feature_importances_
58     }).sort_values("importance", ascending=False)
59     print("\nXếp hạng Features quan trọng:")
60     display(feature_importance.head(10))
```

Phần huấn luyện này tương tự KNN thực hiện tương tự như mô hình trước nhưng bổ sung thêm việc tìm k tối ưu và sau đó huấn luyện mô hình KNN với giá trị k này.

```
1 depth_optimal = find_optimal_depth(X_train, y_train, X_val, y_val)
2 model, accuracy = evaluate_val(X_train, y_train, X_val, y_val,
3                                 depth_optimal)
4 test_accuracy = evaluate_test(model, X_test, y_test)
5 display_feature_importance(model, X_train)
6 ####
7
8 depth_optimal = find_optimal_depth(X_train_fe, y_train_fe, X_val_fe,
9                                     y_val_fe)
10 model_fe, accuracy_fe = evaluate_val(X_train_fe, y_train_fe,
11                                       X_val_fe, y_val_fe, depth_optimal)
12 test_accuracy_fe = evaluate_test(model_fe, X_test_fe, y_test_fe)
13 display_feature_importance(model_fe, X_train_fe)
```

15.2.6 K-means

Giới thiệu



Hình 15.12: Minh họa từng bước hoạt động của thuật toán k-means với $k = 2$.

Khác với các mô hình học có giám sát bên trên, K-means là một thuật toán học không giám sát được sử dụng phổ biến cho bài toán phân cụm dữ liệu. Ý tưởng chính của K-means là nhóm các điểm dữ liệu thành K cụm sao cho các điểm trong cùng một cụm có đặc trưng tương tự nhau và khác biệt so với các cụm khác.

Thuật toán bắt đầu bằng việc khởi tạo ngẫu nhiên K điểm làm centroid (tâm cụm), sau đó lặp lại hai bước chính:

1. Gán mỗi điểm dữ liệu vào cụm có centroid gần nhất và
2. cập nhật lại vị trí của các centroid dựa trên trung bình các điểm trong cụm.

Quá trình này tiếp tục cho đến khi vị trí các centroid hội tụ hoặc đạt điều kiện dừng.

Triển khai

```
1 def evaluate_val(X_train, y_train, X_val, y_val):
2     # Huấn luyện
3     kmean_model = KMeans(n_clusters=2, random_state=SEED, init="random")
4     kmean_model.fit(X_train)
5     # Dự đoán và đánh giá trên tập val
6     val_clusters = kmean_model.predict(X_val)
7     train_clusters = kmean_model.labels_
8     cluster_class_mapping = {
9         cluster_id: Counter(y_train[train_clusters == cluster_id]).most_common(1)[0][0]
10        for cluster_id in np.unique(train_clusters)
11    }
12     kmeans_pred = np.array([cluster_class_mapping[cluster_id] for cluster_id in val_clusters])
13     kmeans_accuracy = accuracy_score(y_val, kmeans_pred)
14
15     print(f"Độ chính xác KMeans (Binary Prediction): {kmeans_accuracy:.4f}")
16
17     print("\nClassification Report (KMeans Binary Prediction):")
18     print(classification_report(y_val, kmeans_pred))
19     return kmean_model, kmeans_accuracy
20
21 def evaluate_test(kmean_model, X_test, y_test):
22     # Dự đoán trên tập test
23     test_clusters = kmean_model.predict(X_test)
24     train_clusters = kmean_model.labels_
25     cluster_class_mapping = {
26         cluster_id: Counter(y_train[train_clusters == cluster_id]).most_common(1)[0][0]
27        for cluster_id in np.unique(train_clusters)
28    }
29     kmeans_test_pred = np.array([cluster_class_mapping[cluster_id] for cluster_id in test_clusters])
30     kmeans_test_accuracy = accuracy_score(y_test, kmeans_test_pred)
31     print(f"Độ chính xác KMeans trên tập test: {kmeans_test_accuracy:.4f}")
32
33     print("\nClassification Report:")
34     print(classification_report(y_test, kmeans_test_pred))
35     return kmeans_test_accuracy
```

Phần huấn luyện K-Means thực hiện tương tự như các phần trước.

```

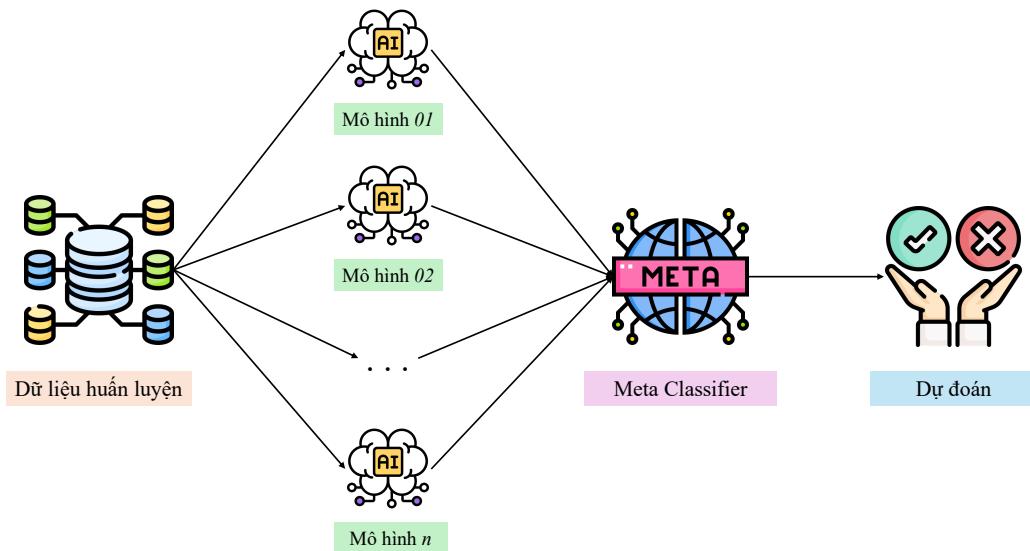
1 model, accuracy = evaluate_val(X_train, y_train, X_val, y_val)
2 test_accuracy = evaluate_test(model, X_test, y_test)
3
4 model_fe, accuracy_fe = evaluate_val(X_train_fe, y_train_fe,
                                         X_val_fe, y_val_fe)
5 test_accuracy_fe = evaluate_test(model_fe, X_test_fe, y_test_fe)

```

15.2.7 Cải tiến: Ensemble

Giới thiệu

Ensemble learning là một kỹ thuật trong học máy nhằm kết hợp nhiều mô hình dự đoán để tạo thành một mô hình mạnh hơn. Ý tưởng chính là tận dụng ưu điểm của từng mô hình đơn lẻ, từ đó giảm sai số và tăng độ ổn định trong quá trình dự đoán. Chính bởi lẽ đó, chúng ta có thể tận dụng các mô hình đã huấn luyện từ trước trong project để ứng dụng kỹ thuật này nhằm làm tăng độ chính xác tổng thể của toàn hệ thống.



Hình 15.13: Minh họa ý tưởng cơ bản của kỹ thuật ensemble learning tận dụng n mô hình AI đã được huấn luyện.

Ensemble learning bao gồm rất nhiều những kỹ thuật con xoay quanh, song trong phạm vi của project này, chúng ta sử dụng phương pháp Stacking, kết hợp ba mô hình cơ bản (KNN, Decision Tree và Naive Bayes) và dùng một bộ phân loại cuối cùng để đưa ra kết quả. Cách tiếp cận này được kỳ vọng sẽ cải thiện hiệu năng so với từng mô hình riêng lẻ.

Triển khai

Đầu tiên, chúng ta khởi tạo mô hình Stacking Ensemble, trong đó ba mô hình con (KNN, Decision Tree và Naive Bayes) đóng vai trò là các bộ phân loại cơ sở. Mô hình KNN được chọn làm bộ phân loại cuối cùng để tổng hợp kết quả.

```

1 def init_model():
2     # Khởi tạo mô hình Stacking Ensemble
3     ensemble_model = StackingClassifier(
4         estimators=[
5             ("knn", KNeighborsClassifier()),
6             ("dt", DecisionTreeClassifier(random_state=SEED)),
7             ("nb", GaussianNB()),
8         ],
9         final_estimator=KNeighborsClassifier(),
10        stack_method="predict_proba",
11        passthrough=False
12    )
13    display(ensemble_model)
14    return ensemble_model

```

Tiếp theo, chúng ta định nghĩa các hàm đánh giá, bao gồm huấn luyện và kiểm tra trên tập validation, cũng như đánh giá hiệu năng cuối cùng trên tập test. Các hàm này trả về độ chính xác cùng báo cáo chi tiết về precision, recall và F1-score.

```

1 def evaluate_ensemble(ensemble_model, X_train, y_train, X_val, y_val):
2     ensemble_model.fit(X_train, y_train)
3     ensemble_pred = ensemble_model.predict(X_val)
4     ensemble_accuracy = accuracy_score(y_val, ensemble_pred)
5     print(f"Độ chính xác Ensemble Model: {ensemble_accuracy:.4f}")

```

```

6   print("\nClassification Report:")
7   print(classification_report(y_val, ensemble_pred))
8   return ensemble_accuracy
9
10 def evaluate_ensemble_test(ensemble_model, X_test, y_test):
11     ensemble_test_pred = ensemble_model.predict(X_test)
12     ensemble_test_accuracy = accuracy_score(y_test,
13                                              ensemble_test_pred)
14     print(f"Độ chính xác Ensemble trên tập test: {ensemble_test_accuracy:.4f}")
15     print("\nClassification Report:")
16     print(classification_report(y_test, ensemble_test_pred))
17     return ensemble_test_accuracy

```

Sau khi đã xây dựng các hàm, chúng ta tiến hành huấn luyện mô hình Ensemble trên bộ dữ liệu gốc, đồng thời đánh giá hiệu năng trên cả tập validation và tập test.

```

1 ensemble_model = init_model()
2 accuracy = evaluate_ensemble(
3     ensemble_model,
4     X_train,
5     y_train,
6     X_val,
7     y_val
8 )
9 test_accuracy = evaluate_ensemble_test(
10    ensemble_model,
11    X_test,
12    y_test
13 )

```

Tương tự, chúng ta cũng lặp lại quy trình với bộ dữ liệu đã được xử lý đặc trưng (feature engineering). Điều này giúp đánh giá mức độ cải thiện hiệu năng của kỹ thuật ensemble khi dữ liệu đầu vào được tinh chỉnh tốt hơn.

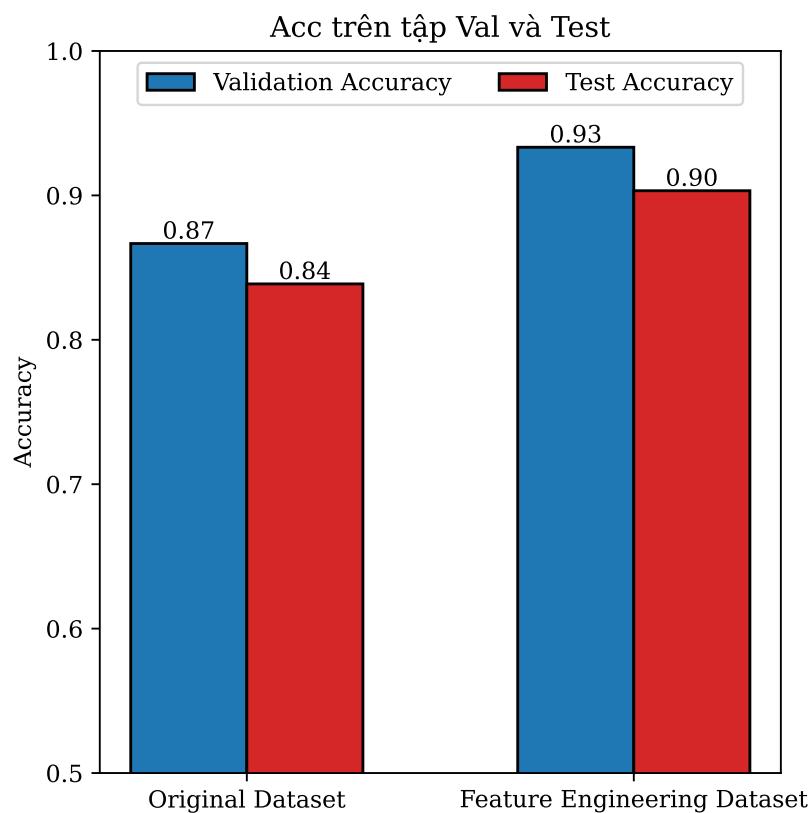
```

1 ensemble_model_fe = init_model()
2 accuracy_fe = evaluate_ensemble(
3     ensemble_model_fe,

```

```
4     X_train_fe,
5     y_train_fe,
6     X_val_fe,
7     y_val_fe
8 )
9 test_accuracy_fe = evaluate_ensemble_test(
10    ensemble_model_fe,
11    X_test_fe,
12    y_test_fe
13 )
```

Biểu đồ so sánh ở Hình 15.14 minh họa sự khác biệt về độ chính xác giữa hai cách tiếp cận: sử dụng dữ liệu gốc và dữ liệu đã được xử lý đặc trưng. Có thể nhận thấy rằng việc áp dụng kỹ thuật feature engineering kết hợp với Ensemble giúp cải thiện hiệu năng một cách rõ rệt.



Hình 15.14: Biểu đồ so sánh mô hình Ensemble (Stacking) trên bộ dữ liệu gốc và bộ được xử lý đặc trưng.

15.3 Câu hỏi trắc nghiệm

1. Trong phần Naive Bayes Classifier, mô hình giả định rằng:
 - (a) Tất cả đặc trưng y tế đều phụ thuộc tuyến tính vào tuổi.
 - (b) Mọi đặc trưng phải được chuẩn hóa trước khi tính toán.
 - (c) Cholesterol và huyết áp độc lập có điều kiện khi biết bệnh nhân mắc hay không mắc bệnh tim.
 - (d) Mỗi đặc trưng chỉ có thể nhận 2 giá trị rời rạc.
2. Trong phần cài đặt code KNN, giá trị K được chọn bằng cách:
 - (a) Dùng GridSearchCV trên toàn bộ dữ liệu.
 - (b) Chọn K bằng cách thử nghiệm ngẫu nhiên nhiều lần.
 - (c) Luôn cố định K = 5 theo tài liệu tham khảo.
 - (d) Tối ưu bằng Stratified K-Fold Cross Validation với k từ 1 đến 20.
3. Khi huấn luyện Decision Tree trong bài, tác giả dùng chỉ số nào để chọn thuộc tính chia tách?
 - (a) Entropy hoặc Gini Impurity để tối đa hóa Information Gain.
 - (b) Mean Squared Error vì đây là bài toán hồi quy.
 - (c) Cosine Similarity để so sánh đặc trưng.
 - (d) Precision-Recall Curve trên tập validation.
4. Với việc thuật toán K-means được cài đặt số cụm bằng 2, điều này phản ánh ý nghĩa gì?
 - (a) Hai trạng thái sức khỏe tim mạch: có bệnh và không bệnh.
 - (b) Hai nhóm bệnh nhân nam và nữ.
 - (c) Hai loại chỉ số lâm sàng: cholesterol và huyết áp.
 - (d) Hai bộ dữ liệu raw và feature engineering.
5. Trong phần Ensemble, bài viết sử dụng **StackingClassifier** với ba mô hình con là:
 - (a) Logistic Regression, SVM, Random Forest.

- (b) KNN, Naive Bayes, K-means.
(c) Decision Tree, K-means, SVM.
(d) KNN, Decision Tree, Naive Bayes.
6. Trong phần Decision Tree, hàm `display_feature_importance()` có vai trò gì?
(a) Hiển thị phân phối nhãn trong tập huấn luyện.
(b) Xếp hạng các đặc trưng quan trọng nhất dựa trên độ đóng góp vào việc phân loại.
(c) Tự động chọn độ sâu tối ưu cho cây.
(d) Chuẩn hóa các đặc trưng đầu vào trước khi huấn luyện.
7. Trong tập dữ liệu Cleveland, thuộc tính `ca` có ý nghĩa gì?
(a) Loại đau ngực (1–4).
(b) Thalassemia (3,6,7).
(c) Điện tâm đồ lúc nghỉ (0–2).
(d) Số mạch máu chính được nhuộm màu (0–3).
8. Đoạn code sau trong phần KNN có mục đích gì?
- 1 `cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)`
- (a) Khởi tạo mô hình KNN với `k=5`.
(b) Chia dữ liệu thành 5 folds cân bằng theo nhãn để cross-validation, giữ kết quả nhất quán nhờ seed.
(c) Ngăn chặn overfitting bằng cách bỏ bớt dữ liệu.
(d) Chuẩn hóa các đặc trưng về cùng khoảng giá trị.
9. Trong phần Naive Bayes, lệnh `classification_report(y_val, nb_pred)` trả về:
(a) Precision, Recall, F1-score cho từng lớp (mắc bệnh tim / không mắc bệnh tim).

- (b) Chỉ duy nhất Accuracy trên toàn bộ tập validation.
- (c) Biểu đồ histogram của dự đoán.
- (d) Giá trị K tối ưu được chọn.
10. Trong triển khai K-means, đoạn code:

```
1 cluster_class_mapping = {  
2     cluster_id: Counter(y_train[train_clusters == cluster_id]).  
3                         most_common(1)[0][0]  
4     for cluster_id in np.unique(train_clusters)  
5 }
```

có ý nghĩa là:

- (a) Ánh xạ mỗi cụm thành nhãn “có bệnh tim” hoặc “không bệnh” dựa trên nhãn chiếm đa số trong cụm train.
- (b) Chuẩn hóa dữ liệu trước khi phân cụm.
- (c) Tự động chọn số cụm tối ưu.
- (d) Loại bỏ các điểm ngoại lai trong tập train.

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
 2. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
 3. **Demo:** Chương trình demo cho nội dung trong bài có thể được truy cập tại [đây](#).
 4. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#).
5. **Rubric:**

Mục	Kiến Thức	Đánh Giá
I.	<ul style="list-style-type: none"> - Hiểu bối cảnh bệnh tim mạch và tầm quan trọng của chẩn đoán sớm. - Nắm được thông tin về bộ dữ liệu Cleveland Heart Disease. - Biết các đặc trưng y tế chính (age, sex, cp, chol, trestbps, ...). 	<ul style="list-style-type: none"> - Trình bày được lý do chọn bộ dữ liệu Cleveland. - Nhận biết và giải thích ý nghĩa các đặc trưng trong bộ dữ liệu. - Hiểu được mối liên hệ giữa dữ liệu y tế và mô hình học máy.
II.	<ul style="list-style-type: none"> - Kiến thức về xử lý đặc trưng (feature engineering). - Kiến thức về các mô hình cơ bản: Naive Bayes, KNN, Decision Tree. - Kiến thức về mô hình không giám sát K-means. - Kiến thức về Ensemble Learning (Stacking). 	<ul style="list-style-type: none"> - Hiểu giả định của Naive Bayes và khi nào phù hợp. - Nhận biết vai trò của K trong KNN và độ sâu trong Decision Tree. - Giải thích nguyên lý K-means và lý do chọn $k = 2$. - Trình bày lợi ích của Ensemble trong việc kết hợp nhiều mô hình.
III.	<ul style="list-style-type: none"> - Kiến thức về triển khai code với scikit-learn. - Kỹ năng sử dụng cross-validation để tìm tham số tối ưu. - Biết cách đánh giá mô hình bằng accuracy, classification report, biểu đồ. 	<ul style="list-style-type: none"> - Viết và hiểu được các hàm <code>evaluate_val</code>, <code>evaluate_test</code>. - Hiểu mục đích của StratifiedKFold trong tìm K tối ưu. - Giải thích được ý nghĩa kết quả accuracy và các chỉ số Precision, Recall, F1. - Biết so sánh hiệu quả giữa tập dữ liệu gốc và tập đã xử lý đặc trưng.

Phần IV

**Module 4: Máy học (tập trung
cho dữ liệu bảng và time-series)**

Chương 16

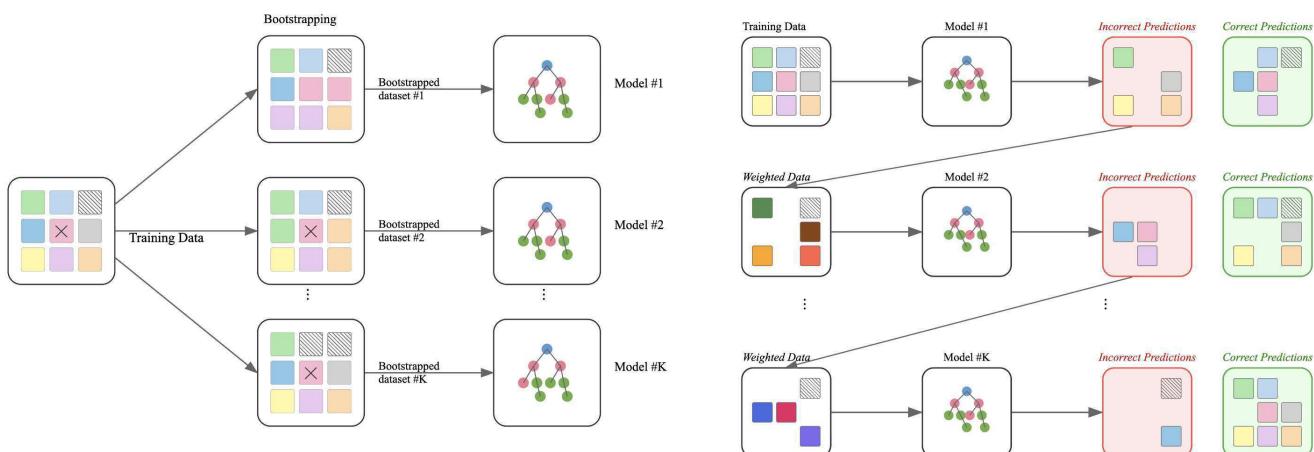
Random Forest và AdaBoost

16.1 Giới thiệu

Ông bà ta có câu:

*"Một cây làm chảng nên non
Ba cây chụm lại nên hòn núi cao."*

Qua đó, chúng ta thấy được rằng sức mạnh của tập thể luôn là chìa khoá để nâng cao tính hiệu quả của công việc. Và đó cũng chính là những hình dung đầu tiên về ý nghĩa quan trọng khi kết hợp nhiều thực thể lại để cùng hoàn thành công việc trong các mô hình học máy (Kỹ thuật ensemble learning). Trong nội dung phần này, chúng ta sẽ tìm hiểu về kỹ thuật học tập tổ hợp (Ensemble learning) thông qua hai mô hình: random forest (rừng ngẫu nhiên) và adaboost.



Hình 16.1: Các kỹ thuật huấn luyện trong Ensemble Learning: Bagging (bên trái) và Boosting (bên phải).

Trong học máy, một mô hình đơn lẻ thường phải đánh đổi giữa thiên lệch (bias) và phương sai (variance). Mô hình phức tạp có thể khớp dữ liệu huấn luyện tốt (thiên lệch nhỏ) nhưng dễ dao động trước mẫu mới (phương sai lớn); mô hình đơn giản thì ngược lại. Ensemble learning đề xuất một

con đường thứ ba: kết hợp nhiều người học yếu theo những nguyên lý thống kê/thuật toán để vừa giảm phương sai (bằng trung bình hóa trên các người học ít tương quan) vừa giảm thiên lệch (bằng học tuần tự, thích nghi vào điểm khó), đồng thời mở rộng biên quyết định—yếu tố đã được chứng minh liên hệ chặt chẽ với năng lực khái quát hóa.

Hai họ kỹ thuật kinh điển là:

- Bagging (Bootstrap Aggregating): huấn luyện song song nhiều người học trên các mẫu bootstrap, rồi bình quân/biểu quyết → chủ yếu giảm phương sai.
- Boosting: huấn luyện tuần tự, thích nghi; mỗi người học mới tập trung vào các ví dụ mà người học trước làm sai → thường giảm thiên lệch, đồng thời tăng margin.

Trong thực hành dữ liệu bảng (tabular data), cây quyết định là base learner ưa dùng do chi phí huấn luyện thấp, mô hình hóa phi tuyến tốt và dễ đa dạng hóa. Chúng ta sẽ nhắc qua về mô hình nền tảng cho kỹ thuật này đó là cây quyết định (Decision Tree).

Decision Tree (DT) là một thuật toán Machine Learning theo kiểu Supervised Learning có thể dùng để giải quyết cho hai dạng bài toán là Regression và Classification. DT xây dựng một cấu trúc dạng cây, với các node đại diện cho quyết định dựa trên một điều kiện của đặc trưng nào đó. Sau khi trải qua một loạt quyết định, kết quả dự đoán cuối cùng chính là giá trị mà node lá nắm giữ.

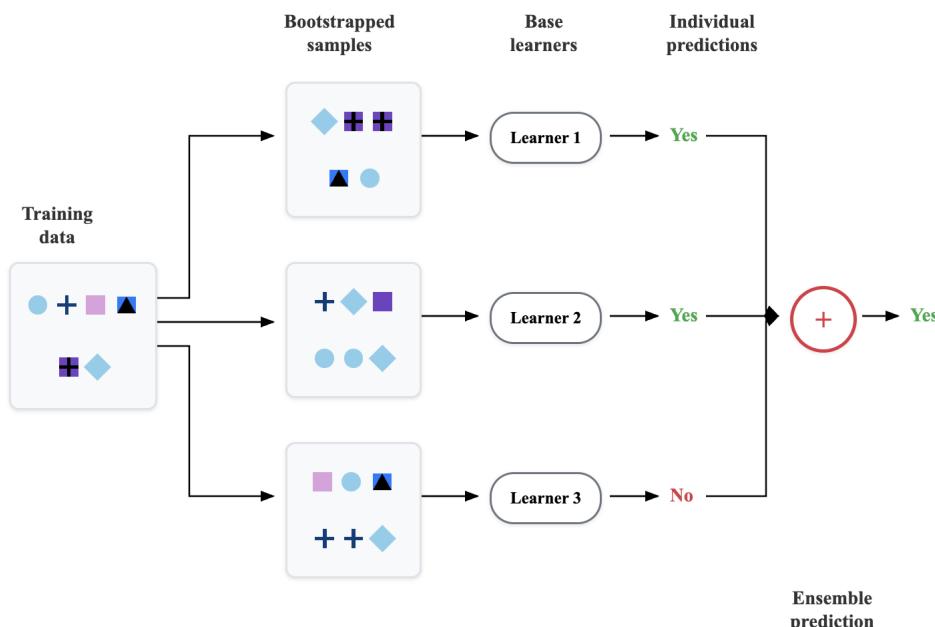
Phần tiếp theo chúng ta sẽ nói về kỹ thuật ensemble learning.

16.2 Kỹ thuật Ensemble learning

16.2.1 Tổng quan về phương pháp

Ensemble Learning là một phương pháp học máy mạnh mẽ dựa trên nguyên lý kết hợp nhiều mô hình (được gọi là base learners) để tạo ra một mô hình tổng hợp có khả năng dự đoán chính xác hơn.

Quá trình thực hiện gồm các bước sau:



Hình 16.2: Kỹ thuật học tập tổ hợp (Ensemble Learning).

1. Bootstrapped Samples (Mẫu Bootstrap)

Quy trình bắt đầu từ việc tạo ra các mẫu bootstrap từ tập dữ liệu huấn luyện gốc. Bootstrap sampling là kỹ thuật lấy mẫu có hoán lại, cho phép tạo ra nhiều tập con khác nhau từ cùng một tập dữ liệu. Mỗi mẫu bootstrap sẽ có kích thước bằng tập dữ liệu gốc nhưng chứa các quan sát khác nhau do quá trình lấy mẫu ngẫu nhiên.

Từ tập huấn luyện \mathcal{D} gồm n quan sát, ta sinh ra B mẫu bootstrap $\mathcal{D}_1, \dots, \mathcal{D}_B$ bằng lấy mẫu có hoán (mỗi \mathcal{D}_b cũng có kích thước n nhưng có thể lặp phần tử). Mục tiêu của bước này là tạo đa dạng dữ liệu để các người học cơ sở nhìn thấy những lát cắt hơi khác nhau của cùng một phân bố.

Điều này tạo ra sự đa dạng trong dữ liệu huấn luyện, giúp các base learners học được những khía cạnh khác nhau của vấn đề, từ đó giảm thiểu hiện tượng overfitting và tăng khả năng tổng quát hóa.

2. Base Learners (Mô hình Cơ sở)

Từ mỗi \mathcal{D}_b , ta huấn luyện một "người học cơ sở" (base learner) h_b .

Các base learners này có thể là:

- Cùng loại thuật toán (ví dụ: tất cả đều là decision trees)
- Khác loại thuật toán (ví dụ: kết hợp decision trees, SVM, neural networks)

3. Individual Predictions (Dự đoán Cá nhân)

Với điểm mới x , từng h_b đưa ra dự đoán (Yes/No cho phân loại, giá trị số cho hồi quy).

Trong hình minh họa, ta thấy:

- Learner 1 dự đoán: "Yes"
- Learner 2 dự đoán: "Yes"
- Learner 3 dự đoán: "No"

4. Ensemble Prediction (Dự đoán Tổng hợp)

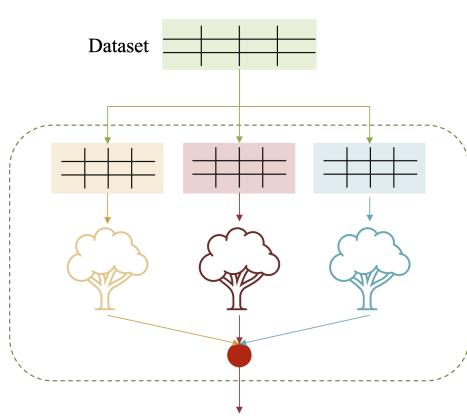
Bước cuối cùng là kết hợp các dự đoán cá nhân thành một dự đoán chung. Có nhiều phương pháp kết hợp:

- Voting (Bỏ phiếu)
 - Hard voting: Chọn nhãn được dự đoán nhiều nhất
 - Soft voting: Tính trung bình xác suất dự đoán
- Weighted voting: Gán trọng số khác nhau cho các base learners dựa trên hiệu suất của chúng.

Trong ví dụ minh họa, với 2 'Yes' và 1 'No', ensemble prediction sẽ là 'Yes' theo nguyên tắc majority voting.

16.2.2 Rừng ngẫu nhiên (Random Forest)

Random Forest là một trong những thuật toán ensemble learning phổ biến nhất, được phát triển bởi Leo Breiman vào năm 2001. Đây là sự kết hợp của kỹ thuật Bootstrap Aggregating (Bagging) với việc lựa chọn ngẫu nhiên các đặc trưng (random feature selection), tạo ra một ‘rừng’ gồm nhiều cây quyết định (decision trees) hoạt động độc lập và kết hợp kết quả để đưa ra dự đoán cuối cùng.



Hình 16.3: Rừng ngẫu nhiên (Random Forest)

Random Forest (RF) là một thuật toán Machine Learning theo kiểu Ensemble Learning có thể dùng để giải quyết cho hai dạng bài toán là Regression và Classification. Ý tưởng của RF liên quan đến việc sử dụng nhiều DT, mỗi cây sẽ học trên một tập con của một bộ training dataset (Bootstrap). Khi thực hiện dự đoán, mỗi cây sẽ đưa ra kết quả dự đoán của mình, sau đó tổng hợp toàn bộ kết quả lại theo một cách để trả về kết quả cuối cùng.

Random Forest sử dụng hai lớp tính ngẫu nhiên:

- Random Sampling: Mỗi cây được huấn luyện trên một mẫu bootstrap khác nhau
- Random Feature Selection: Tại mỗi node của cây, chỉ xem xét một tập con ngẫu nhiên các đặc trưng

Các bước thực hiện của rừng ngẫu nhiên

- Chuẩn bị dữ liệu

- Tập dữ liệu huấn luyện $D = (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$
- Số lượng cây trong rừng: N_{trees}
- Số lượng đặc trưng được xem xét tại mỗi split: m

- Bootstrap Sampling

Bootstrap Sampling tạo ra sự đa dạng cần thiết giữa các cây trong rừng. Đây là kỹ thuật thống kê cho phép chúng ta tạo ra nhiều phiên bản khác nhau của cùng một tập dữ liệu, mỗi phiên bản sẽ “dạy” cho một cây những khía cạnh khác nhau của vấn đề. Điều quan trọng của bootstrap là việc “có hoàn lại nghĩa là một mẫu có thể được chọn nhiều lần trong cùng một bootstrap sample, trong khi một số mẫu khác có thể không được chọn lần nào. Điều này tạo ra sự biến đổi tự nhiên trong dữ liệu mà mỗi cây sẽ nhìn thấy.

Quy trình:

For $i = 1$ to N_{trees} :

1. Tạo mẫu bootstrap D_i bằng cách:
 - Lấy mẫu có hoàn lại từ tập D
 - Kích thước mẫu $= |D|$ (thường là n mẫu)
 - Mỗi mẫu có xác suất được chọn $= 1/n$
2. Kết quả: 63.2% mẫu gốc được chọn, 36.8% không được chọn (Out-of-Bag)

Ví dụ:

Tập gốc: [A, B, C, D, E]
 Bootstrap 1: [A, A, C, D, B]
 Bootstrap 2: [B, C, C, E, A]
 Bootstrap 3: [D, A, E, E, C]

- Xây dựng từng cây quyết định

Với mỗi bộ dữ liệu bootstrap sẽ được huấn luyện để xây dựng một cây quyết định tương ứng. Dựa trên các phương pháp tinh chỉnh và tối ưu cây quyết định.

Ví dụ:

Bootstrap Sample 1: [1,1,3,4,2]
 Random features tại root: [Chiều cao, Tuổi]
 Best split: Chiều cao < 172.5

Bootstrap Sample 2: [2,3,4,5,5]
 Random features tại root: [Cân nặng, Tuổi]
 Best split: Cân nặng < 62.5

Bootstrap Sample 3: [1,2,2,4,5]
 Random features tại root: [Chiều cao, Cân nặng]
 Best split: Cân nặng < 67.5

- Tạo các dự đoán Với mỗi mẫu mới, ta đẩy qua các mô hình dự đoán và tổng hợp kết quả thông qua tính trung bình hoặc voting. So sánh hard Voting và Soft Voting

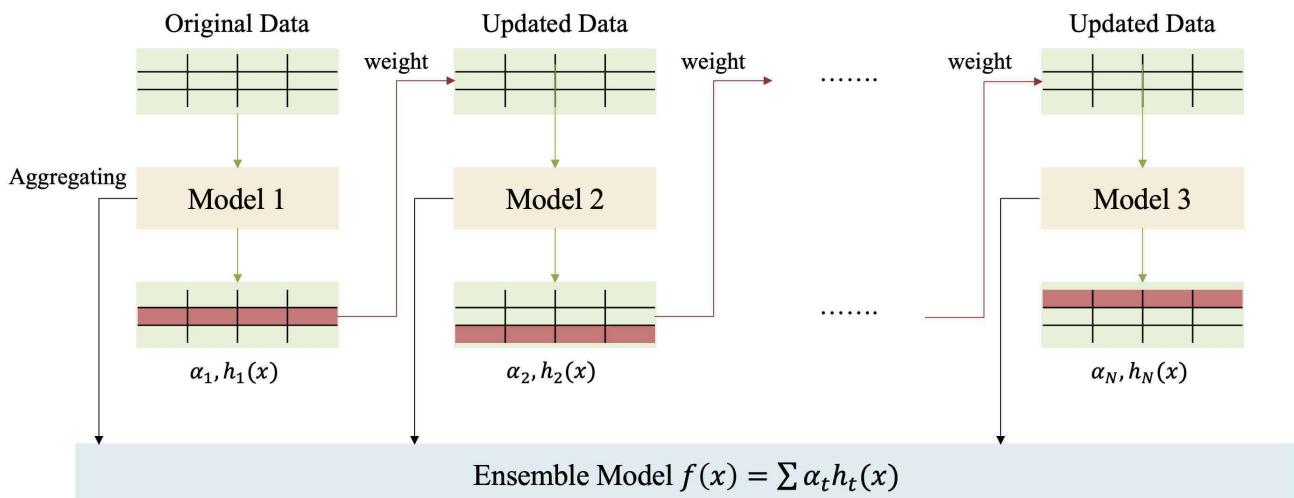
For mỗi cây T_i in Random Forest:
 |— prediction_i = $T_i.predict(x_{new})$
 - Hard Voting: Chọn class được vote nhiều nhất
 $final_prediction = mode([prediction_1, prediction_2, \dots, prediction_N])$
 - Soft Voting: Trung bình xác suất
 $P(class_j) = (1/N) \times \sum(P_i(class_j))$
 $final_prediction = argmax(P(class_j))$

16.2.3 Mô hình AdaBoost

AdaBoost (Adaptive Boosting) là một trong những thuật toán ensemble learning, được phát triển bởi Yoav Freund và Robert Schapire vào năm 1995. Khác với Random Forest tập trung vào việc tạo sự đa dạng thông qua lấy mẫu và lựa chọn đặc trưng ngẫu nhiên, AdaBoost có phương pháp học hoàn toàn khác: học từ sai lầm. Thay vì huấn luyện các mô hình độc lập, AdaBoost xây dựng một chuỗi các weak learner, mỗi learner được thiết kế để sửa chữa những sai lầm của các learner trước đó.

Thuật toán liên tục điều chỉnh trọng số của các mẫu huấn luyện, tập trung nhiều hơn vào những mẫu khó phân loại. Quá trình này tạo ra một hiệu ứng dây chuyền: mỗi mô hình mới được tối ưu hóa để xử lý những trường hợp mà các mô hình trước đó gặp khó khăn, dần dần xây dựng một sự hiểu biết toàn diện về không gian vấn đề.

Quá trình thực hiện gồm các bước sau:



Hình 16.4: AdaBoost (Ensemble Learning).

- Chuẩn bị dữ liệu

AdaBoost bắt đầu với một phân phối đều trên các mẫu huấn luyện, với giả định ban đầu rằng tất cả mẫu đều quan trọng như nhau. Giai đoạn

thiết lập cũng bao gồm việc chọn các weak learner phù hợp - thường là decision stump (cây chỉ có một phép chia) hoặc cây nông.

```

Training_Data = (x1, y1), (x2, y2), ..., (xn, yn)
Initial_Weights = [1/n, 1/n, ..., 1/n]
Number_of_Rounds = T
Weak_Learners = []
Model_Weights = []

```

- Quá trình huấn luyện

Ý tưởng chính AdaBoost là trong quá trình cải tiến lặp. Mỗi lần lặp bao gồm một số bước được điều phối cẩn thận được thiết kế để cải thiện hiệu suất ensemble một cách tiến bộ.

Tại vòng lặp t , weak learner h_t được huấn luyện bằng cách sử dụng dữ liệu huấn luyện có trọng số. Phân phối trọng số W_t phản ánh “độ khó” hiện tại của mỗi mẫu từ góc nhìn của ensemble hiện có.

Trọng số mô hình α_t xác định ảnh hưởng của weak learner h_t trong ensemble cuối cùng.

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

Trong đó:

- Z_t là hằng số chuẩn hóa: $Z_t = \sum_{i=1}^n W_t(i) \cdot \exp(-\alpha_t y_i h_t(x_i))$

- $y_i h_t(x_i) = +1$ nếu phân loại đúng, -1 nếu phân loại sai

Sau đó thực hiện phép đo lỗi để kiểm tra hiệu suất của các learners.

Sau đó tính toán và cập nhật trọng số.

$$W_{t+1}(i) = \frac{W_t(i) \cdot \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

- Dự đoán

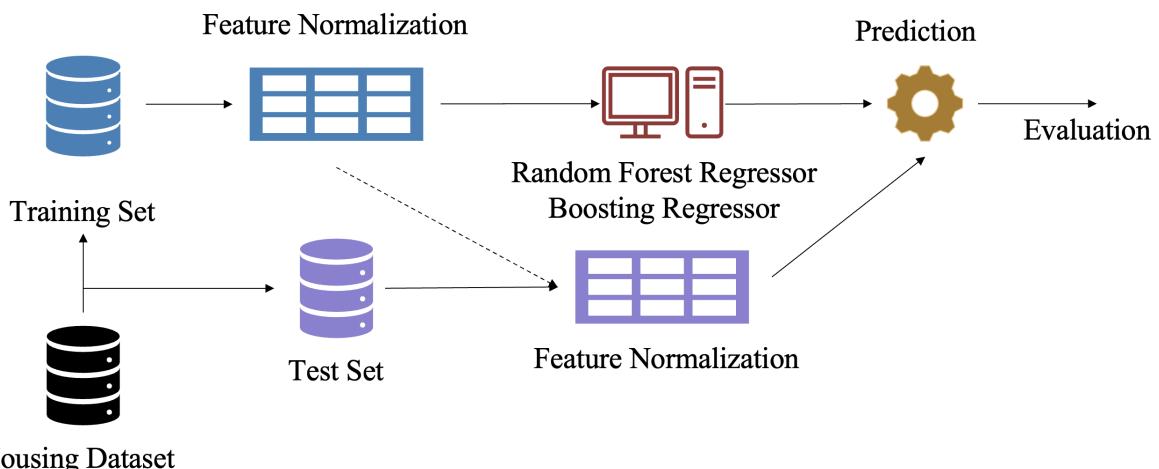
Ensemble cuối cùng kết hợp tất cả weak learner bằng cách sử dụng bỏ phiếu đa số có trọng số, trong đó ảnh hưởng của mỗi learner tỷ lệ với hiệu suất của nó trong quá trình huấn luyện.

Hàm dự đoán cuối cùng được định nghĩa như sau:

$$f(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right)$$

16.3 Huấn luyện mô hình trên bộ dữ liệu Housing

Trong phần này chúng ta sẽ huấn luyện các mô hình này dựa trên bộ dữ liệu ‘Housing.csv’



Hình 16.5: Huấn luyện mô hình Random Forest, AdaBoost dựa vào bộ dữ liệu Housing.

Dựa vào đoạn code gợi ý để hoàn thiện quá trình huấn luyện và đánh giá các mô hình. Ngoài mô hình Random Forest, AdaBoost trong phần này chúng ta sẽ sử dụng thêm mô hình Gradient Boosting.

16.3.1 Chuẩn bị thư viện

Chúng ta cài đặt các công cụ từ thư viện ‘sklearn’ để thực thi phần code.

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 from sklearn.ensemble import RandomForestRegressor,
5     AdaBoostRegressor,
6     GradientBoostingRegressor
7
8 from sklearn.preprocessing import OrdinalEncoder
  
```

```

6 from sklearn.preprocessing import StandardScaler
7 from sklearn.model_selection import train_test_split
8 from sklearn.metrics import mean_absolute_error, mean_squared_error

```

16.3.2 Chuẩn bị dữ liệu

Bộ dữ liệu Housing được sử dụng trong thực nghiệm này là một ví dụ điển hình của bài toán regression trong thực tế, có các đặc trưng số và categorical.

```

1 # Download dataset
2 !gdown 1qeJqFtRdjHqExbWJcgKy0yJbczTTAE3
3
4 # Load dataset
5 dataset_path = "./Housing.csv"
6 df = pd.read_csv(dataset_path)
7 df

```

Khi đó, ta có thể thấy nội dung của bảng dữ liệu có dạng như sau:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

Hình 16.6: Một vài mẫu dữ liệu của bộ dữ liệu Housing.

16.3.3 Mã hóa thuộc tính văn bản

Ordinal Encoding được áp dụng cho các biến categorical, chuyển đổi chúng thành dạng số mà các thuật toán machine learning có thể xử lý. Việc lựa chọn Ordinal Encoder thay vì One-Hot Encoding giúp giảm chiều dữ liệu, đặc biệt phù hợp với các mô hình dựa trên cây quyết định.

```

1 categorical_cols = df.select_dtypes(include=["object"]).columns.
    to_list()
2
3 ordinal_encoder = OrdinalEncoder()
4 encoded_categorical_cols = ordinal_encoder.fit_transform(df[
    categorical_cols])
5 encoded_categorical_df = pd.DataFrame(
6     encoded_categorical_cols,
7     columns=categorical_cols
8 )
9 numerical_df = df.drop(categorical_cols, axis=1)
10 encoded_df = pd.concat(
11     [numerical_df, encoded_categorical_df], axis=1
12 )
13 encoded_df

```

Khi đã hoàn tất, ta được một DataFrame mới có dạng như sau:

	price	area	bedrooms	bathrooms	stories	parking	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea	furnishingstatus
0	13300000	7420	4	2	3	2	1.0	0.0	0.0	0.0	1.0	1.0	0.0
1	12250000	8960	4	4	4	3	1.0	0.0	0.0	0.0	1.0	0.0	0.0
2	12250000	9960	3	2	2	2	1.0	0.0	1.0	0.0	0.0	1.0	1.0
3	12215000	7500	4	2	2	3	1.0	0.0	1.0	0.0	1.0	1.0	0.0
4	11410000	7420	4	1	2	2	1.0	1.0	1.0	0.0	1.0	0.0	0.0

Hình 16.7: Bộ dữ liệu Housing sau khi đã chuyển đổi toàn bộ các đặc trưng Categorical về dạng số.

StandardScaler được sử dụng để chuẩn hóa dữ liệu, đảm bảo tất cả các đặc trưng có cùng scale. Mặc dù Random Forest tương đối không nhạy cảm với scale của dữ liệu, việc chuẩn hóa vẫn quan trọng để đảm bảo công bằng giữa các đặc trưng và tăng tốc độ hội tụ của các thuật toán.

```

1 # Normalization
2 normalizer = StandardScaler()
3 dataset_arr = # Your code here
4
5 # Train: Test split
6 X, y = dataset_arr[:, 1:], dataset_arr[:, 0]
7 test_size = 0.3

```

```

8 random_state = 1
9 is_shuffle = True
10 X_train, X_val, y_train, y_val = train_test_split(
11     # Your code here
12 )
13
14 print(f"Number of training samples: {X_train.shape[0]}")
15 print(f"Number of val samples: {X_val.shape[0]}")

```

16.3.4 Huấn luyện và đánh giá các mô hình

Mô hình Random Forest

Random Forest có một số tham số quan trọng ảnh hưởng trực tiếp đến hiệu suất:

- n_estimators: Số lượng cây trong rừng là tham số quan trọng nhất. Giá trị càng cao thường cho kết quả càng ổn định, nhưng cũng tăng chi phí tính toán. Với bộ dữ liệu Housing, việc tăng từ 10 lên 100 estimators có thể cải thiện đáng kể hiệu suất.

- max_depth: Độ sâu tối đa của mỗi cây quyết định tính từ root đến leaf. Tham số này kiểm soát độ phức tạp của từng cây. Nếu quá nhỏ, mô hình có thể underfitting; nếu quá lớn, có thể dẫn đến overfitting mặc dù Random Forest có cơ chế chống overfitting. Giá trị mặc định None (unlimited depth) thường hoạt động tốt.

- max_features: Số lượng features được xem xét tại mỗi split, đây chính là yếu tố tạo tính ngẫu nhiên thứ hai. Với regression, giá trị mặc định là $\sqrt{\text{tổng số features}}$, còn với classification là $\log_2(\text{tổng số features})$. Tham số này cân bằng giữa diversity và accuracy của các cây.

- min_samples_split và min_samples_leaf: Kiểm soát một node có thể được chia tiếp hoặc trở thành leaf. Các giá trị cao hơn giúp chống overfitting nhưng có thể làm giảm khả năng học pattern phức tạp.

```

1 regressor = RandomForestRegressor(
2     # Your code here
3 )
4 regressor.fit(X_train, y_train)

```

```
5 y_pred = regressor.predict(X_val)
6 mae = mean_absolute_error(y_val, y_pred)
7 mse = mean_squared_error(y_val, y_pred)
8
9 print("Evaluation results on validation set:")
10 print(f"Mean Absolute Error: {mae}")
11 print(f"Mean Squared Error: {mse}")
```

Kết quả nhận được sau khi huấn luyện mô hình:

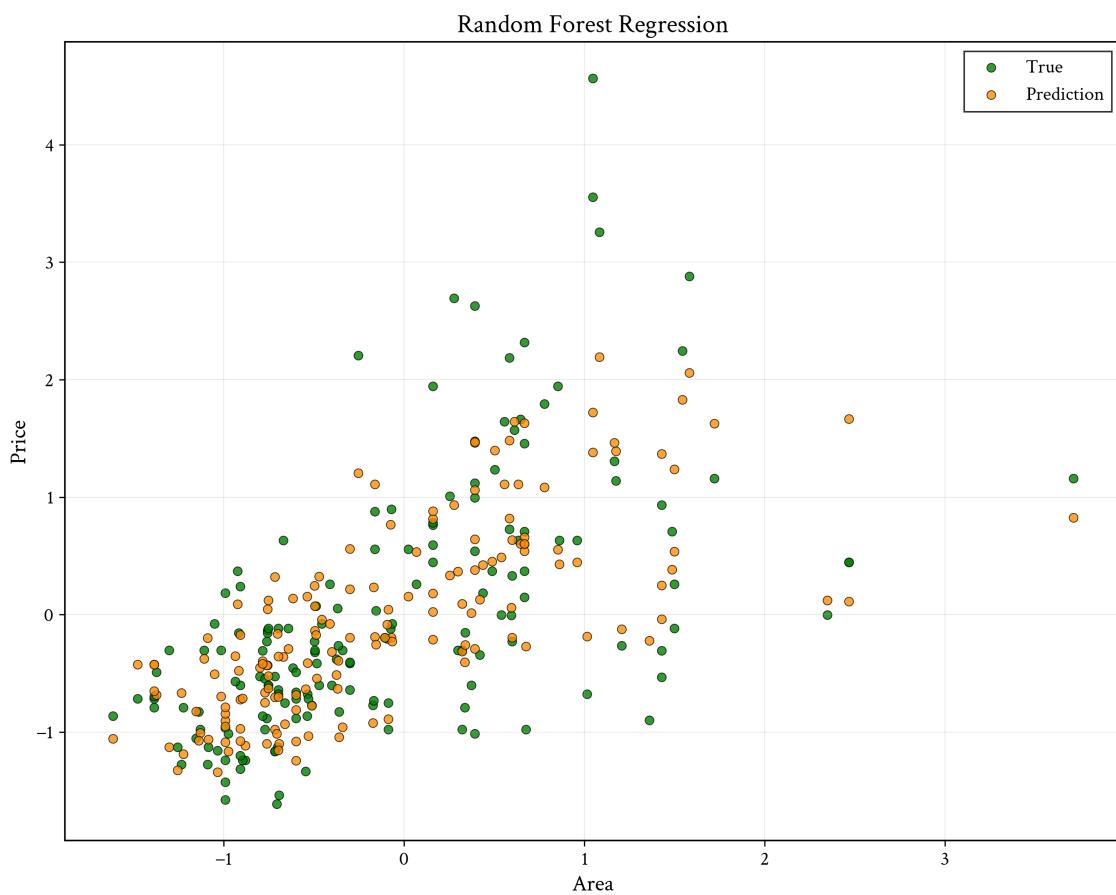
```
Evaluation results on validation set:
Mean Absolute Error: 0.46093873321571177
Mean Squared Error: 0.37944418523089524
```

Phân bô kết quả các điểm dữ liệu thực tế và dự đoán:

Mô hình AdaBoost

AdaBoost có những tham số đặc trưng khác biệt so với Random Forest:

- n_estimators: Tương tự Random Forest nhưng ý nghĩa khác biệt. Trong AdaBoost, đây là số vòng boosting. Quá nhiều có thể dẫn đến overfitting nghiêm trọng vì AdaBoost học tuần tự và có thể "ghi nhớ" noise. Thông thường 50-200 estimators là phù hợp.
- learning_rate: Tham số này điều chỉnh contribution của mỗi weak learner vào kết quả cuối cùng. Giá trị nhỏ hơn (0.1-0.3) thường cho kết quả tốt hơn nhưng cần nhiều estimators hơn. Có trade-off giữa learning_rate và n_estimators: learning_rate thấp cần n_estimators cao và ngược lại.
- base_estimator: Weak learner được sử dụng, mặc định là DecisionTreeClassifier với max_depth=1 (decision stump). Với regression, thường sử dụng DecisionTreeRegressor với depth nhỏ. Việc sử dụng weak learner quá mạnh có thể làm mất đi bản chất "weak" và dẫn đến overfitting.



Hình 16.8: Phân bố các điểm dự đoán và thực tế.

```

1 regressor = AdaBoostRegressor(
2     # Your code here
3 )
4 regressor.fit(X_train, y_train)
5
6 y_pred = regressor.predict(X_val)
7
8 mae = mean_absolute_error(y_val, y_pred)
9 mse = mean_squared_error(y_val, y_pred)
10
11 print("Evaluation results on validation set:")
12 print(f"Mean Absolute Error: {mae}")

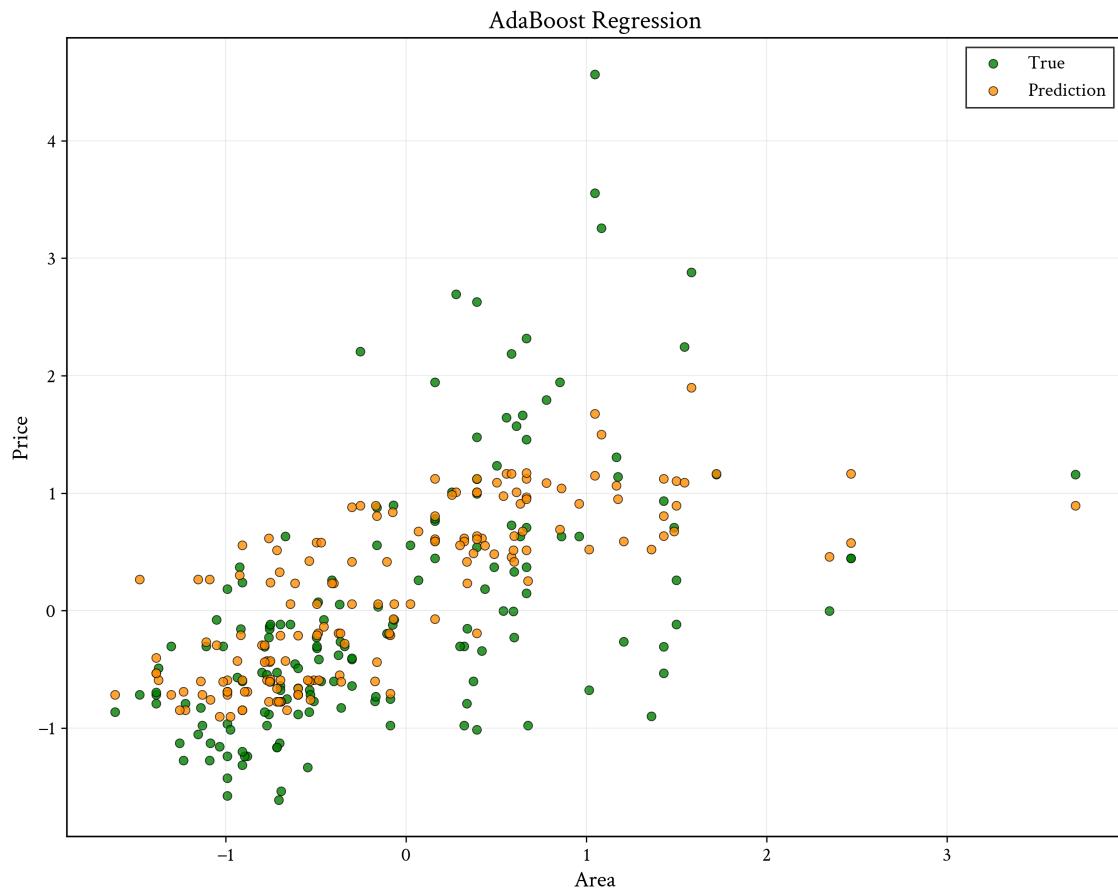
```

```
13 print(f"Mean Squared Error: {mse}")
```

Kết quả nhận được sau khi huấn luyện mô hình:

Evaluation results on validation set:
 Mean Absolute Error: 0.567680019897059
 Mean Squared Error: 0.5739244030038942

Phân bố kết quả các điểm dữ liệu thực tế và dự đoán:



Hình 16.9: Phân bố các điểm dự đoán và thực tế.

Mô hình Gradient Boosting

Gradient Boosting có bộ tham số phức tạp nhất:

- n_estimators và learning_rate: Cặp tham số này có mối quan hệ tương tự AdaBoost. Learning_rate (thường 0.01-0.3) kiểm soát step size trong gradient descent. Giá trị thấp cần nhiều estimators nhưng thường cho kết quả tốt hơn và ít overfitting hơn.
- max_depth: Thường sử dụng cây nông (3-8 levels) vì Gradient Boosting dựa vào việc kết hợp nhiều weak learner. Cây quá sâu có thể làm mỗi estimator quá mạnh và dẫn đến overfitting.
- subsample: Tỷ lệ mẫu được sử dụng để huấn luyện mỗi estimator (stochastic gradient boosting). Giá trị 0.8-1.0 thường tốt. Subsample < 1.0 có thể giảm overfitting và tăng tốc độ huấn luyện.
- min_samples_split, min_samples_leaf: Tương tự Random Forest nhưng quan trọng hơn trong Gradient Boosting vì mô hình dễ overfitting hơn. Giá trị cao hơn giúp tạo ra weak learners thực sự "weak".
- loss function: Với regression có các lựa chọn 'squared_error', 'absolute_error', 'huber'. Huber loss ít nhạy cảm với outliers hơn squared error, có thể phù hợp với dữ liệu housing có các giá trị ảnh hưởng mạnh.

```

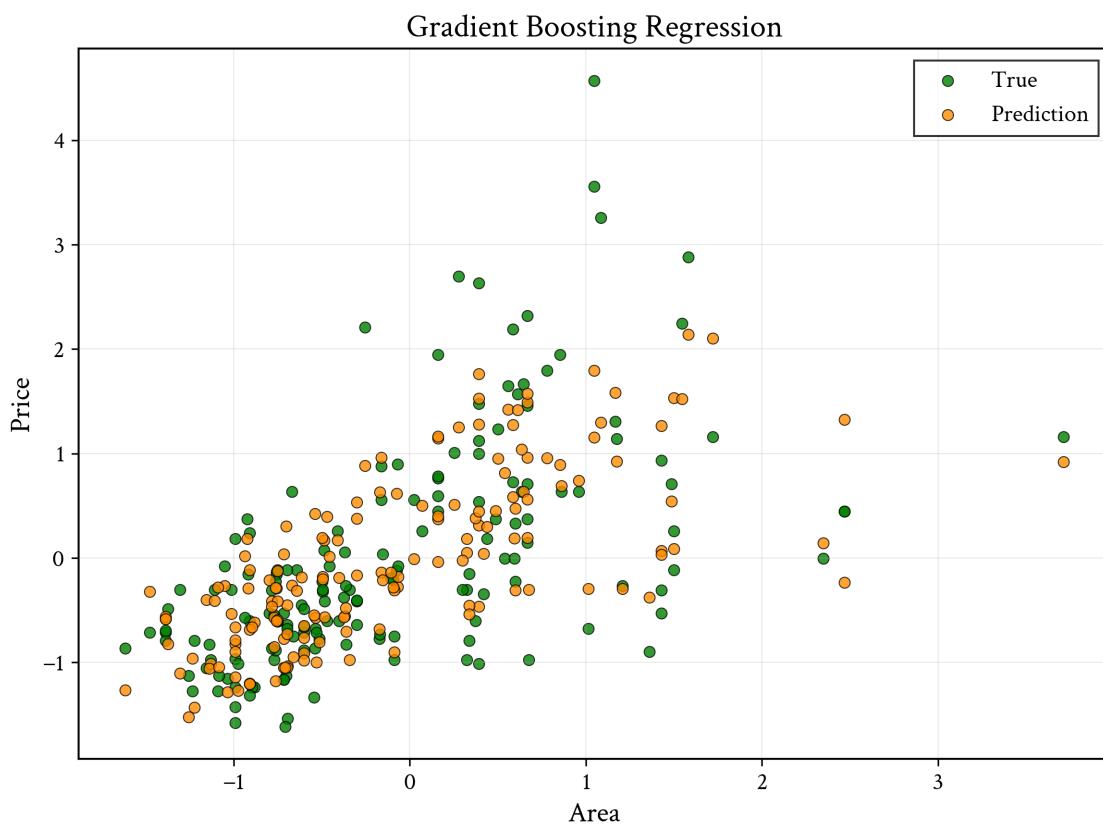
1 regressor = GradientBoostingRegressor(
2     # Your code here
3 )
4 regressor.fit(X_train, y_train)
5
6 y_pred = regressor.predict(X_val)
7
8 mae = mean_absolute_error(y_val, y_pred)
9 mse = mean_squared_error(y_val, y_pred)
10
11 print("Evaluation results on validation set:")
12 print(f"Mean Absolute Error: {mae}")
13 print(f"Mean Squared Error: {mse}")

```

Kết quả nhận được sau khi huấn luyện mô hình:

Evaluation results on validation set:
Mean Absolute Error: 0.4516626127750995
Mean Squared Error: 0.39610445936979427

Phân bố kết quả các điểm dữ liệu thực tế và dự đoán:



Hình 16.10: Phân bố các điểm dự đoán và thực tế.

16.4 Câu hỏi trắc nghiệm

1. Giá trị khởi tạo cho giá trị dự đoán trong thuật toán Gradient Boosting được xác định dựa vào?
 - a) Giá trị lớn nhất
 - b) Giá trị nhỏ nhất
 - c) Giá trị trung bình
 - d) Tổng các giá trị
2. Thuật ngữ nào sau đây không thuộc Decision Tree?
 - a) Logistic Regression
 - b) Gini Index
 - c) Pruning
 - d) Splitting Criterion
3. Công thức nào sau đây là đúng về Mean Squared Error (MSE)?
 - a) $\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)$
 - b) $\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$
 - c) $\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$
 - d) $\frac{1}{n} \sum_{i=1}^n \frac{y_i}{\hat{y}_i}$
4. Trong regression tree, thành phần nào đóng vai trò là điểm khởi đầu trong việc đưa ra quyết định?
 - a) Internal Node
 - b) Leaf Node
 - c) Branch
 - d) Root Node
5. Trong regression tree, thành phần nào đóng vai trò chứa kết quả dự đoán và không phân nhánh?
 - a) Internal Node
 - b) Leaf Node

- c) Branch
- d) Root Node

6. Cho một bộ dataset có nội dung như sau:

X	Y
3	12
5	20
8	28
10	32
12	36

Dựa theo lý thuyết về Decision Tree, các bạn hãy trả lời một số câu hỏi sau:

- 6.1 Điều kiện nào sau đây có thể là điều kiện phân nhánh đầu tiên cho cây, sử dụng độ đo MSE làm splitting criterion?
- a) $X \leq 3$
 - b) $X \leq 8$
 - c) $X \leq 5$
 - d) $X \leq 10$
- 6.2 Dựa vào kết quả đạt được ở câu 6.a, giả sử với giá trị đầu vào X là 2, kết quả dự đoán của cây là?
- a) 14
 - b) 16
 - c) 18
 - d) 20
- 6.3 Dựa vào kết quả đạt được ở câu 6.a, giả sử với giá trị đầu vào X là 15, kết quả dự đoán của cây là?
- a) 20
 - b) 24
 - c) 30
 - d) 32
7. Lý do chính của việc sử dụng bootstrap dataset trong Random Forest là?
- a) Tăng tốc độ huấn luyện
 - b) Ưu tiên các đặc trưng quan trọng
 - c) Khắc phục missing values
 - d) Tránh overfitting
8. Từ nào sau đây được dùng để miêu tả số lượng cây trong Random Forest?
- a) max_depth
 - b) max_features
 - c) n_estimators
 - d) criterion

9. Trong Random Forest, khái niệm Bagging được hiểu là?

- a) Bootstrap Aggregating
- b) Binary Aggregating
- c) Balanced Algorithm Grouping
- d) Best Algorithm for Generalization

10. Cho một bộ dataset có nội dung như sau:

X	Y
2	4
1	3
3	5
2	6

Dựa trên bộ dataset này, bạn sẽ xây dựng một mô hình random forest với số cây là 2. Độ sâu cho mỗi cây $max_depth = 1$ và giá trị dự đoán cuối cùng là trung bình các giá trị dự đoán của mỗi cây. Từ đây, các bạn hãy trả lời một số câu hỏi sau:

- 10.1 Giả sử cả hai cây đều tính toán trên toàn bộ dataset trên. Cây 1 chia nhánh với điều kiện $X \geq 2$, cây 2 chia nhánh với điều kiện $X \geq 3$. Khi đó, kết quả dự đoán của mô hình với $X = 2$ là (lấy giá trị gần nhất)?
- a) 4.0
 - b) 4.5
 - c) 5.0
 - d) 5.5
- 10.2 Giả sử bộ dữ liệu bootstrap của hai cây lần lượt là (hiển thị theo chỉ mục của các mẫu trong bộ dữ liệu gốc):
- **Cây 1:** Gồm các mẫu 0, 1, 2.
 - **Cây 2:** Gồm các mẫu 1, 2, 3.
- Cây 1 chia nhánh với điều kiện $X \geq 2$, cây 2 chia nhánh với điều kiện $X \geq 3$. Khi đó, kết quả dự đoán của mô hình với $X = 1$ là?
- a) 3.25
 - b) 3.50
 - c) 3.75
 - d) 4.0

- 1. Hint:** Các file code gợi ý và dữ liệu nếu có được lưu trong thư mục có thể được tải [tại đây](#).
- 2. Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. Rubric:

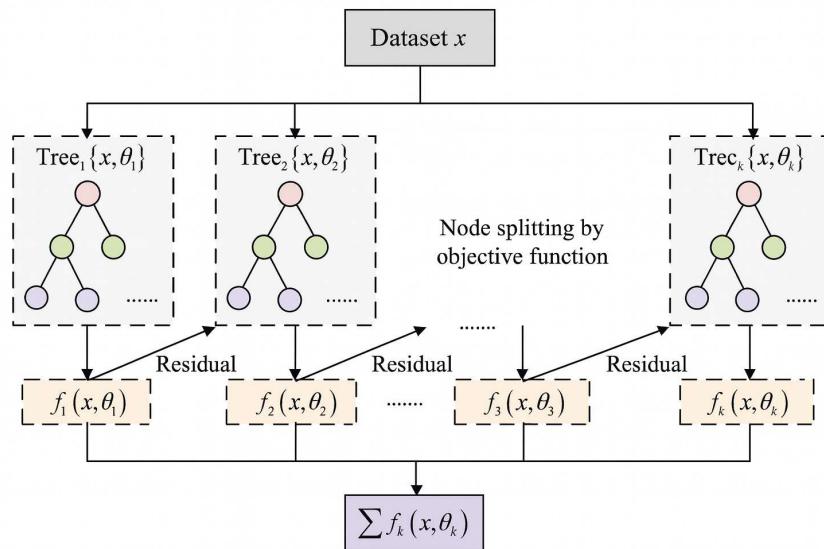
Mục	Kiến Thức	Đánh Giá
II. A	<ul style="list-style-type: none"> - Kiến thức về thư viện sklearn. - Kiến thức về cách sử dụng thư viện sklearn để huấn luyện mô hình Decision Tree và Random Forest. 	<ul style="list-style-type: none"> - Hiểu được cách xây dựng và huấn luyện mô hình Decision Tree và Random Forest cho bài Regression sử dụng thư viện sklearn.
II. B	<ul style="list-style-type: none"> - Kiến thức cơ bản về Decision Tree. - Kiến thức cơ bản về Random Forest. - Kiến thức cơ bản về AdaBoost. - Kiến thức cơ bản về Gradient Boosting. - Các khái niệm khác có liên quan đến Decision Tree và Random Forest (MAE, MSE...) 	<ul style="list-style-type: none"> - Nắm được cách thực thi các thuật toán Decision Tree và Random Forest, AdaBoost, Gradient Boosting.

Chương 17

Gradient Boosting và XGBoost

17.1 Giới thiệu

Các phương pháp ensemble learning như random forest hay adaboost đều đạt được độ chính xác cao trong thực tế. Trong phần nội dung này chúng ta tiếp tục đi tìm hiểu những phương pháp nâng cao hơn – đó chính là Gradient Boosting và XGBoost - hay còn gọi là Extreme Gradient Boosting. Được giới thiệu lần đầu tiên trong bài báo của Friedman với tiêu đề ‘Greedy Function Approximation: A Gradient Boosting Machine’, XGBoost đã nhanh chóng ghi dấu ấn mạnh mẽ trong việc giải quyết các bài toán học có giám sát.



Hình 17.1: Thuật toán XGBoost - Extreme Gradient Boosting.

Với khả năng xử lý những bộ dữ liệu phức tạp và nhiều thuộc tính, XGBoost trở thành công cụ lý tưởng để dự báo các biến kết quả, từ đó đưa ra những quyết định chính xác và đáng tin cậy. Kể từ khi được ra mắt vào 2014, XGBoost không chỉ trở thành công cụ ưa thích trong giới nghiên cứu mà còn là một thuật toán hàng đầu trong các cuộc thi phân tích dữ liệu.

Điều khiến XGBoost trở nên đặc biệt là nhờ vào khả năng tối ưu hóa hiệu quả qua từng bước, không ngừng học hỏi từ các sai số và cải thiện qua từng

lần huấn luyện. Đây là lý do tại sao thuật toán này vẫn duy trì sức hút mạnh mẽ trong học máy – nó không chỉ nổi bật trong các bài toán phân loại mà còn đã từng được ứng dụng trong thực tiễn từ tài chính đến y tế.

17.2 Mô hình Gradient Boosting

17.2.1 Tổng quan

Gradient Boosting là một kỹ thuật học máy ensemble thuộc nhóm boosting, được phát triển bởi Jerome H. Friedman vào năm 1999. Phương pháp này đã trở thành một trong những công cụ quan trọng nhất trong lĩnh vực học máy, đặc biệt trong các cuộc thi dữ liệu và ứng dụng thực tế.

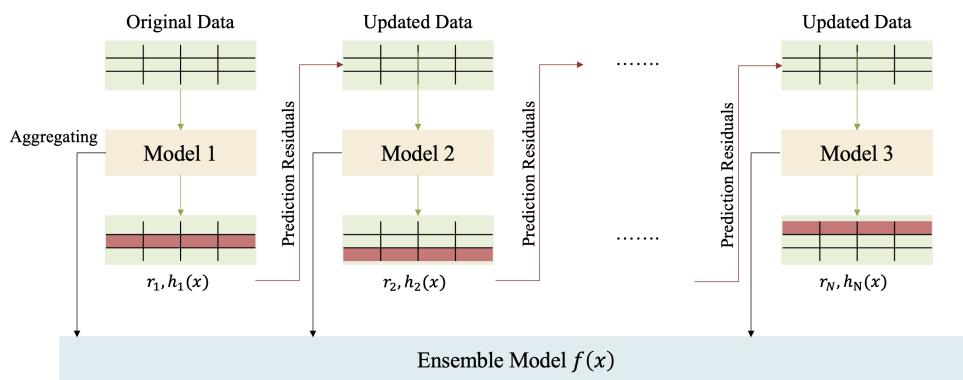
Phương pháp xây dựng mô hình dự đoán bằng cách kết hợp tuần tự nhiều mô hình yếu (weak learners), thường là cây quyết định nông (shallow decision trees). Điểm đặc biệt của phương pháp này là mỗi mô hình mới được huấn luyện để tối ưu hóa gradient của hàm mất mát dựa trên dự đoán của các mô hình trước đó.

Ý tưởng chính của Gradient Boosting dựa trên hai nguyên tắc:

- **Boosting:** Xây dựng mô hình mạnh từ việc kết hợp nhiều mô hình yếu
- **Gradient Descent:** Sử dụng gradient descent trong không gian hàm để tối ưu hóa hàm mất mát

17.2.2 Thuật toán Gradient Boosting

Các bước thực hiện của thuật toán được mô tả trong hình dưới đây:



Hình 17.2: Thuật toán Gradient Boosting.

Khởi tạo: Initial Model Prediction

Mô hình được khởi tạo với giá trị dự đoán ban đầu là giá trị trung bình của tất cả các quan sát:

$$\mu = \frac{1}{N} \sum_{i=1}^N y_i \quad (17.1)$$

Trong đó:

- N là số lượng mẫu trong tập huấn luyện
- y_i là giá trị thực tế của quan sát thứ i
- μ là giá trị dự đoán ban đầu (thường là mean cho regression, log-odds cho classification)

Bước 1: Calculate Residuals

Tính toán phần dư (residuals) - sự khác biệt giữa giá trị thực tế và giá trị dự đoán hiện tại:

$$r_1 = y - \hat{y} = y - \mu \quad (\text{Lần lặp đầu tiên}) \quad (17.2)$$

$$r_2 = y - y_{new} \quad (\text{Các lần lặp tiếp theo}) \quad (17.3)$$

Phần dư này đại diện cho "sai số" mà mô hình hiện tại chưa giải thích được và sẽ là mục tiêu cho mô hình tiếp theo học.

Bước 2: Build a Decision Tree $h(x)$

Xây dựng một cây quyết định (decision tree) để dự đoán phần dư:

- **Input:** Features X và residuals r từ bước 1
- **Output:** Một hàm $h(x)$ mapping từ features đến predicted residuals
- **Mục tiêu:** Minimize squared error giữa actual residuals và predicted residuals

Cây quyết định này thường được giới hạn độ sâu (shallow tree) để tránh overfitting:

$$h(x) = \arg \min_h \sum_{i=1}^N (r_i - h(x_i))^2 \quad (17.4)$$

Bước 3: Compute Residual Outputs

Tính toán output của cây quyết định cho mỗi leaf node. Với mỗi leaf j :

$$\gamma_j = \arg \min_{\gamma} \sum_{x_i \in \text{leaf}_j} L(y_i, F_{m-1}(x_i) + \gamma) \quad (17.5)$$

Trong đó:

- γ_j là giá trị tối ưu cho leaf j
- L là loss function
- F_{m-1} là mô hình tại iteration trước

Với squared loss, γ_j đơn giản là trung bình của các residuals trong leaf j .

Bước 4: Update Predictions

Cập nhật dự đoán bằng cách thêm dự đoán mới (được scale bởi learning rate) vào mô hình hiện tại:

$$y_{new} = \hat{y} + lr \times \hat{r} \quad (17.6)$$

Trong đó:

- \hat{y} là dự đoán hiện tại
- $lr = 0.1$ là learning rate (như trong hình)
- \hat{r} là predicted residuals từ decision tree $h(x)$
- y_{new} là dự đoán được cập nhật

Learning rate nhỏ (0.1) giúp:

- Tránh overfitting
- Cải thiện generalization
- Cho phép fine-tuning tốt hơn

Vòng lặp và hội tụ

Quá trình 4 bước này được lặp lại cho đến khi:

1. Đạt số lượng iterations (trees) đã định trước
2. Residuals đủ nhỏ (convergence criterion)
3. Validation error không cải thiện (early stopping)

Mô hình cuối cùng là tổng của tất cả các predictions:

$$F_M(x) = \mu + \sum_{m=1}^M lr \times h_m(x) \quad (17.7)$$

17.2.3 Huấn luyện mô hình trên bộ dữ liệu Housing

Trong phần này chúng ta sẽ huấn luyện các mô hình này dựa trên bộ dữ liệu ‘Housing.csv’. Dựa vào đoạn code gợi ý để hoàn thiện quá trình huấn luyện và đánh giá các mô hình.

Chuẩn bị thư viện

Chúng ta cài đặt các công cụ từ thư viện ‘sklearn’ để thực thi phần code.

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 from sklearn.ensemble import GradientBoostingRegressor
5 from sklearn.preprocessing import OrdinalEncoder
6 from sklearn.preprocessing import StandardScaler

```

```

7 from sklearn.model_selection import train_test_split
8 from sklearn.metrics import mean_absolute_error, mean_squared_error

```

Chuẩn bị dữ liệu

Bộ dữ liệu Housing được sử dụng trong thực nghiệm này là một ví dụ điển hình của bài toán regression trong thực tế, có các đặc trưng số và categorical.

```

1 # Download dataset
2 !gdown 1qeJqFtRdjHqExbWJcgKy0yJbczTTAE3
3
4 # Load dataset
5 dataset_path = "./Housing.csv"
6 df = pd.read_csv(dataset_path)
7 df

```

Mã hóa thuộc tính văn bản

Ordinal Encoding được áp dụng cho các biến categorical, chuyển đổi chúng thành dạng số mà các thuật toán machine learning có thể xử lý. Việc lựa chọn Ordinal Encoder thay vì One-Hot Encoding giúp giảm chiều dữ liệu, đặc biệt phù hợp với các mô hình dựa trên cây quyết định.

```

1 categorical_cols = df.select_dtypes(include=["object"]).columns.
2                               to_list()
3
4 ordinal_encoder = OrdinalEncoder()
5 encoded_categorical_cols = ordinal_encoder.fit_transform(df[
6                                         categorical_cols])
7 encoded_categorical_df = pd.DataFrame(
8     encoded_categorical_cols,
9     columns=categorical_cols
10)
11 numerical_df = df.drop(categorical_cols, axis=1)
12 encoded_df = pd.concat(
13     [numerical_df, encoded_categorical_df], axis=1

```

```

12 )
13 encoded_df

```

StandardScaler được sử dụng để chuẩn hóa dữ liệu, đảm bảo tất cả các đặc trưng có cùng scale. Mặc dù Random Forest tương đối không nhạy cảm với scale của dữ liệu, việc chuẩn hóa vẫn quan trọng để đảm bảo công bằng giữa các đặc trưng và tăng tốc độ hội tụ của các thuật toán.

```

1 # Normalization
2 normalizer = StandardScaler()
3 dataset_arr = # Your code here
4
5 # Train: Test split
6 X, y = dataset_arr[:, 1:], dataset_arr[:, 0]
7 test_size = 0.3
8 random_state = 1
9 is_shuffle = True
10 X_train, X_val, y_train, y_val = train_test_split(
11     # Your code here
12 )
13
14 print(f"Number of training samples: {X_train.shape[0]}")
15 print(f"Number of val samples: {X_val.shape[0]}")

```

Mô hình Gradient Boosting

Gradient Boosting có bộ tham số phức tạp nhất:

- n_estimators và learning_rate: Cặp tham số này có mối quan hệ tương tự AdaBoost. Learning_rate (thường 0.01-0.3) kiểm soát step size trong gradient descent. Giá trị thấp cần nhiều estimators nhưng thường cho kết quả tốt hơn và ít overfitting hơn.
- max_depth: Thường sử dụng cây nồng (3-8 levels) vì Gradient Boosting dựa vào việc kết hợp nhiều weak learner. Cây quá sâu có thể làm mỗi estimator quá mạnh và dẫn đến overfitting.
- subsample: Tỷ lệ mẫu được sử dụng để huấn luyện mỗi estimator

(stochastic gradient boosting). Giá trị 0.8-1.0 thường tốt. Subsample < 1.0 có thể giảm overfitting và tăng tốc độ huấn luyện.

- min_samples_split, min_samples_leaf: Tương tự Random Forest nhưng quan trọng hơn trong Gradient Boosting vì mô hình dễ overfitting hơn. Giá trị cao hơn giúp tạo ra weak learners thực sự "weak".
- loss function: Với regression có các lựa chọn 'squared_error', 'absolute_error', 'huber'. Huber loss ít nhạy cảm với outliers hơn squared error, có thể phù hợp với dữ liệu housing có các giá trị ảnh hưởng mạnh.

```

1 regressor = GradientBoostingRegressor(
2     # Your code here
3 )
4 regressor.fit(X_train, y_train)
5
6 y_pred = regressor.predict(X_val)
7
8 mae = mean_absolute_error(y_val, y_pred)
9 mse = mean_squared_error(y_val, y_pred)
10
11 print("Evaluation results on validation set:")
12 print(f"Mean Absolute Error: {mae}")
13 print(f"Mean Squared Error: {mse}")

```

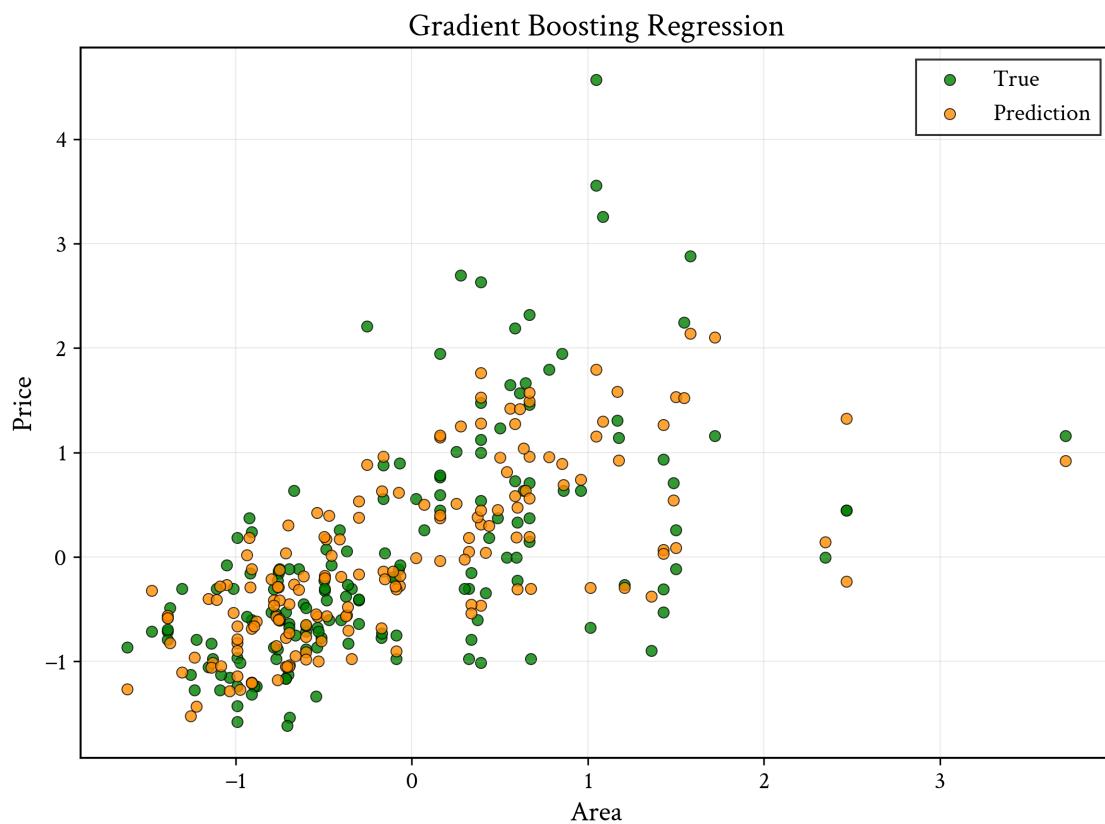
Kết quả nhận được sau khi huấn luyện mô hình:

```

Evaluation results on validation set:
Mean Absolute Error: 0.4516626127750995
Mean Squared Error: 0.39610445936979427

```

Phân bố kết quả các điểm dữ liệu thực tế và dự đoán:

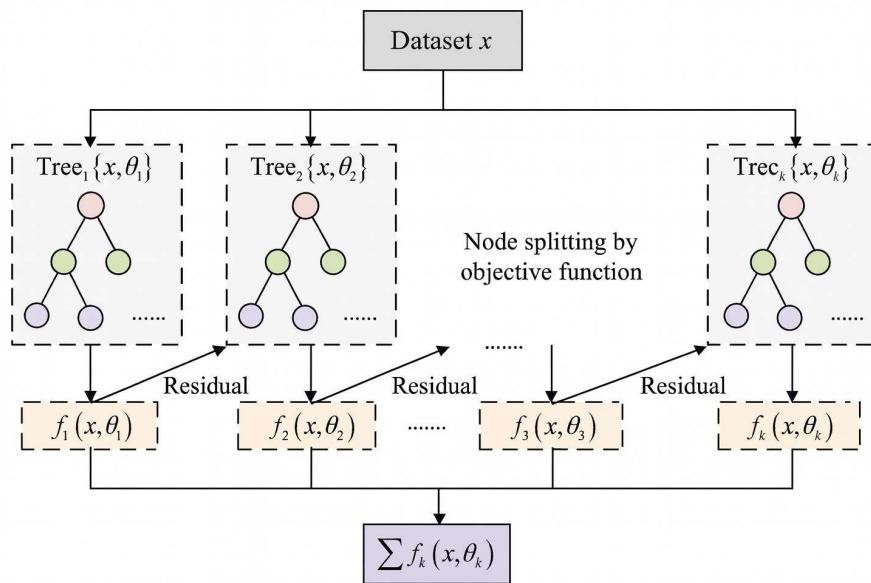


Hình 17.3: Phân bố các điểm dự đoán và thực tế.

17.3 Mô hình XGBoost cho Classification

17.3.1 Tổng quan

XGBoost rất hiệu quả trong việc giải quyết các bài toán phân loại, tức là bài toán mà mục tiêu là phân loại các mẫu dữ liệu vào các nhóm khác nhau.



Hình 17.4: Mô hình XGBoost

Trong bài toán phân loại, XGBoost giúp dự đoán xác suất mà một mẫu dữ liệu thuộc về một nhóm nào đó. Cụ thể, nó sẽ tính toán xác suất mẫu đó thuộc về lớp 1 hay lớp 0 (phân loại nhị phân). Sau khi có xác suất này, chúng ta sẽ so sánh nó với một ngưỡng (thường là 0.5 cho phân loại nhị phân). Nếu xác suất cao hơn 0.5, mẫu dữ liệu sẽ được phân loại vào lớp 1, còn nếu xác suất nhỏ hơn hoặc bằng 0.5, mẫu sẽ được phân loại vào lớp 0.

Ví dụ, nếu XGBoost dự đoán xác suất một bệnh nhân thuộc lớp có bệnh là 0.8, chúng ta sẽ phân loại bệnh nhân này là có bệnh, vì xác suất lớn hơn 0.5. Ngược lại, nếu xác suất là 0.3, bệnh nhân sẽ được phân loại là không có bệnh.

17.3.2 Các bước trong XGBoost cho Classification

Trong XGBoost, để giải quyết bài toán phân loại, thuật toán sử dụng một chuỗi các bước nhằm tối ưu hóa mô hình phân loại dựa trên các cây quyết định. Dưới đây là các bước chi tiết trong quá trình huấn luyện mô hình XGBoost cho bài toán phân loại:

Bước 1: Khởi tạo giá trị dự đoán ban đầu (f_0)

Ban đầu, XGBoost khởi tạo một giá trị dự đoán f_0 cho tất cả các mẫu trong bộ dữ liệu. Đối với bài toán phân loại nhị phân, giá trị này thường là 0.5. Việc khởi tạo này giúp mô hình bắt đầu với một "điểm xuất phát" trung lập.

Bước 2: Tính toán Similarity Score của root

Tại bước này, thuật toán tính toán Similarity Score cho root (nút gốc của cây quyết định). Đây là một phép đo cho biết độ "tương đồng" giữa các mẫu trong tập dữ liệu. Similarity Score được tính theo công thức:

$$\text{Similarity Score} = \frac{(\text{Sum of Residuals})^2}{\sum [\text{Previous Probability} \times (1 - \text{Previous Probability})] + \lambda}$$

- Sum of Residuals: là tổng các sai số giữa giá trị thực tế và giá trị dự đoán của mẫu ($Y - f_0$).
- Previous Probability: là xác suất dự đoán của mô hình trước khi huấn luyện cây mới. Đối với phân loại nhị phân, đây là giá trị xác suất mà mô hình dự đoán mẫu thuộc về lớp 1.
- λ (lambda): là tham số điều chỉnh (regularization), dùng để giảm sự phức tạp của mô hình và ngăn ngừa hiện tượng overfitting.

Công thức này giúp tính toán mức độ phù hợp của các dự đoán và tìm ra cách tốt nhất để chia dữ liệu vào các nhánh trong cây quyết định.

Bước 3: Chọn điều kiện root và tính Similarity Score cho các node

Có nhiều cách chọn điều kiện root, cơ bản nhất là lấy trung bình của 2 sample liền kề. Sau đó tính Similarity Score cho các node trong nhánh trái và nhánh phải.

Bước 4: Tính Gain cho từng điều kiện root đã chọn

Sau khi có các điều kiện chia nhánh, XGBoost tính toán Gain cho mỗi điều kiện (split). Gain giúp đánh giá mức độ cải thiện mô hình khi chia nhánh tại một điểm cụ thể. Gain được tính bằng công thức:

$\text{Gain} = \text{Left Similarity Score} + \text{Right Similarity Score} - \text{Root Similarity Score}$

XGBoost sẽ chọn điều kiện chia nhánh có Gain lớn nhất, tức là chia nhánh sao cho mô hình cải thiện độ chính xác nhiều nhất.

Bước 5: Lặp lại quá trình cho các cây quyết định

XGBoost sẽ tiếp tục lặp lại quá trình trên cho mỗi cây quyết định trong mô hình. Mỗi cây mới học từ các sai số của cây trước đó, giúp mô hình dần dần cải thiện khả năng phân loại. Mỗi cây mới sẽ đưa ra một Output giúp cập nhật giá trị dự đoán.

Output được tính theo công thức:

$$\text{Output} = \frac{\text{Sum of Residuals}}{\sum [\text{Previous Probability} \times (1 - \text{Previous Probability})]}$$

Đây là giá trị giúp điều chỉnh dự đoán của mô hình tại mỗi bước.

Bước 6: Dự đoán kết quả cho toàn bộ tập dữ liệu

Cuối cùng, sau khi các cây quyết định được xây dựng, XGBoost sẽ tính toán xác suất dự đoán cho từng mẫu dữ liệu trong tập huấn luyện. Điều này được thực hiện bằng cách thay thế giá trị f_0 ban đầu và tính toán xác suất mới.

Để dự đoán xác suất cuối cùng của một mẫu, XGBoost sử dụng công thức:

$$\text{Probability} = \frac{e^{\text{LogPrediction}}}{1 + e^{\text{LogPrediction}}}$$

Trong đó, LogPrediction được tính bằng công thức:

$$\text{LogPrediction} = \log \left(\frac{\text{Previous Probability}}{1 - \text{Previous Probability}} \right) + \text{lr} \times \text{Output}$$

- log: là Natural logarithm, còn gọi là ln.

- lr: là learning rate, tham số điều chỉnh tốc độ học của mô hình.
- Output: là giá trị đã tính được ở các bước trước.

Cuối cùng, mô hình phân loại sẽ quyết định nhãn (label) cho mẫu dữ liệu dựa trên xác suất đã tính được.

17.3.3 Huấn luyện mô hình cho bộ dữ liệu phân loại chất lượng rượu vang

Bài tập này chúng ta sẽ tập trung vào việc triển khai và đào tạo mô hình XGBoost. Mục tiêu chính của chúng ta là sử dụng mô hình này để giải quyết một bài toán classification, cụ thể là phân loại chất lượng rượu vang (3 classes: 0, 1, 2). Để thực hiện điều này, chúng ta sẽ tuân theo một số bước cụ thể mà sẽ được trình bày sau đây.

Bước 1: Import thư viện cần thiết

Trong bước này, chúng ta sẽ import các thư viện hỗ trợ phân tích dữ liệu, trực quan hóa và xây dựng mô hình:

- numpy, pandas: xử lý dữ liệu dạng số và bảng.
- matplotlib.pyplot: vẽ biểu đồ, trực quan hóa dữ liệu.
- xgboost: thư viện chính để xây dựng mô hình XGBoost.
- sklearn.metrics: cung cấp hàm accuracy_score để đánh giá độ chính xác.
- sklearn.model_selection: dùng để chia tập dữ liệu thành train/test.

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import xgboost as xgb
5 from sklearn.metrics import accuracy_score
6 from sklearn.preprocessing import LabelEncoder
7 from sklearn.model_selection import train_test_split
8 from sklearn.datasets import load_wine
9 from sklearn.preprocessing import StandardScaler
```

Bước 2: Load dữ liệu

Chúng ta sử dụng phương thức `read_csv()` của pandas để đọc dữ liệu từ một tệp .csv. (Tải data tại [đây](#)). Khi dữ liệu được load thành công, chúng ta có một `DataFrame` với các cột đặc trưng (features) và một cột nhãn (target).

```

1 # Load dataset
2 wine = load_wine()
3 df = pd.DataFrame(wine.data, columns=wine.feature_names)
4 df[["target"]] = wine.target
5
6 df

```

Khi đó, ta có thể thấy nội dung của bảng dữ liệu có dạng như sau:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	Target
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0

Hình 17.5: Một vài mẫu dữ liệu trong data.

Trong bài toán này:

- Cột "Target" là biến mục tiêu (label), thể hiện chất lượng rượu vang với ba mức (0, 1, 2).
- Các cột còn lại là các đặc trưng hóa học của rượu vang, được dùng làm dữ liệu đầu vào (X).

Bước 3: Tách dữ liệu thành X và y

Ở bước này, chúng ta tách dữ liệu thành hai phần:

- X: chứa tất cả các đặc trưng (feature).
- y: chứa nhãn (label) cần dự đoán, ở đây là cột cuối cùng "Target".

```
1 X, y = # Your code here
```

Bước 4: Chia dữ liệu train/test

Để đánh giá mô hình một cách khách quan, dữ liệu được chia thành 2 tập:

- **train**: dùng để huấn luyện mô hình.
- **test**: dùng để kiểm tra khả năng tổng quát hóa của mô hình.

Chúng ta chọn tỷ lệ 70% train và 30% test.

```
1 X_train, X_test, y_train, y_test = train_test_split(  
2     # Your code here  
3 )
```

Bước 5: Xây dựng mô hình XGBoost Classifier

Sau khi có tập train/test, chúng ta tạo một mô hình **XGBClassifier**. Tham số **seed=7** được dùng để cố định tính ngẫu nhiên, giúp kết quả có thể tái lập lại.

```
1 xg_class = # Your code here  
2 xg_class.fit(X_train, y_train)
```

Bước 6: Dự đoán trên tập test

Khi mô hình đã được huấn luyện, ta có thể sử dụng nó để dự đoán nhãn cho các mẫu dữ liệu mới (ở đây là tập test).

```
1 pred = xg_class.predict(X_test)
```

Bước 7: Đánh giá mô hình

Trong bài toán phân loại (Classification), một độ đo phổ biến là Accuracy:

$$\text{Accuracy} = \frac{\text{Số mẫu phân loại đúng}}{\text{Tổng số mẫu}}$$

Accuracy cho biết tỷ lệ dự đoán đúng của mô hình. Đây là chỉ số đơn giản nhưng hữu ích để đánh giá ban đầu. Trong các trường hợp dữ liệu mất cân bằng (một lớp có nhiều mẫu hơn các lớp khác), chúng ta cần bổ sung thêm các chỉ số khác như Precision, Recall, F1-score, ROC, AUC.

```
1 train_acc = accuracy_score(y_train, xg_class.predict(X_train))
2 test_acc = accuracy_score(y_test, pred)
3
4 print(f"Train ACC: {train_acc}")
5 print(f"Test ACC: {test_acc}")
```

17.4 Câu hỏi trắc nghiệm

1. Trong phân loại nhị phân bằng XGBoost, giá trị khởi tạo f_0 (theo xác suất) mặc định trong thư viện XGBoost là?
 - a) 0.0
 - b) 0.5
 - c) 1.0
 - d) Tỷ lệ lớp dương trong dữ liệu

2. Cho nhãn $Y = [1, 0, 1, 0]$, $f_0 = 0.5$, $\lambda = 0$, $Previous\ Probability = 0.5$ cho mọi mẫu. Hãy tính **Similarity Score** của **root**:

$$\text{Similarity} = \frac{\left(\sum(y - 0.5) \right)^2}{\sum[0.5(1 - 0.5)] + \lambda}.$$

- a) 0.00
 - b) 0.25
 - c) 1.00
 - d) 2.00
3. Cho $X = [23, 24, 25, 27, 28, 30]$ và $Y = [1, 1, 0, 1, 0, 1]$, với $f_0 = 0.5$, $Previous\ Probability = 0.5$, $\lambda = 0$. Xét split $X < 26.5$. **Similarity** của **nhánh trái** bằng:

$$\text{Left Similarity} = \frac{\left(\sum(y - 0.5) \right)^2}{\sum[0.5(1 - 0.5)]}.$$

- a) ≈ 0.00
 - b) ≈ 0.33
 - c) ≈ 0.50
 - d) ≈ 0.75
4. Vẫn với dữ liệu ở trên, **Similarity(root)** ≈ 0.67 . Với split $X < 25$, ta có **Similarity(left)** = 2.00, **Similarity(right)** = 0.00. **Gain** của split này là:

$$\text{Gain} = \text{Left} + \text{Right} - \text{Root}.$$

- a) ≈ 0.67
- b) ≈ 1.00
- c) ≈ 1.33
- d) ≈ 2.00

5. Với một node có 2 mẫu [1, 1], giả định *Previous Probability* = 0.5 cho mọi mẫu, $\lambda = 0$. **Output** của node theo công thức bên dưới là bao nhiêu?

$$\text{Output} = \frac{\sum(y - 0.5)}{\sum[0.5(1 - 0.5)]}$$

- a) 0.5
- b) 1.0
- c) 2.0
- d) 4.0

6. Một mẫu có *Previous Probability* = 0.5, rơi vào lá với **Output** = 2.0; dùng $lr = 0.3$. Tính:

$$\text{LogPrediction} = \log\left(\frac{0.5}{1 - 0.5}\right) + 0.3 \times 2.0, \quad \text{Probability} = \frac{e^{\text{LogPrediction}}}{1 + e^{\text{LogPrediction}}}.$$

- a) ≈ 0.58
- b) ≈ 0.65
- c) ≈ 0.73
- d) ≈ 0.80

7. Khi tăng λ trong công thức **Similarity** (nằm ở mẫu số), tác động chính là:

- 8. Ưu tiên các node nhỏ (ít mẫu)
- 9. Giảm điểm **Similarity** mạnh hơn ở node nhỏ, gián tiếp ưu tiên node lớn

10. Không ảnh hưởng đến việc chọn split
11. Chỉ ảnh hưởng đến xác suất sau cùng
12. Dữ liệu cực kỳ mất cân bằng: 90% âm, 10% dương. Giá trị gợi ý của `scale_pos_weight` là:
 - a) ≈ 0.1
 - b) ≈ 1
 - c) ≈ 9
 - d) ≈ 10
13. Mô hình có Train ACC = 0.99, Test ACC = 0.85. Phương án nào **không** phải là cách điển hình để giảm overfitting?
14. Giảm `max_depth`, dùng `early_stopping_rounds`
15. Giảm `learning_rate`, tăng `n_estimators` hợp lý
16. Tăng `max_depth` và bỏ regularization
17. Dùng `subsample`, `colsample_bytree`
18. Trên tập test 100 mẫu, confusion matrix: TP= 18, FP= 7, TN= 65, FN= 10. **Accuracy** là:

$$\text{ACC} = \frac{TP + TN}{TP + FP + TN + FN}.$$

- a) 0.78
- b) 0.83
- c) 0.85
- d) 0.90

- 1. Hint:** Các file code gợi ý và dữ liệu nếu có được lưu trong thư mục có thể được tải [tại đây](#).
- 2. Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. Rubric:

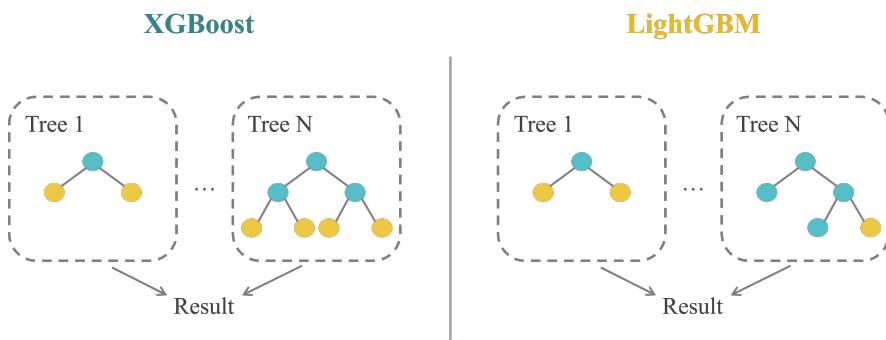
Mục	Kiến Thức	Đánh Giá
II. A	<ul style="list-style-type: none"> - Nguyên lý XGBoost trong bài toán Classification. - Các bước chính: khởi tạo f_0, tính Similarity, chọn split, tính Gain và Output. - Ý nghĩa của tham số λ, learning rate, depth. 	<ul style="list-style-type: none"> - Nắm quy trình tổng quát của XGBoost. - Hiểu vai trò của từng bước và từng tham số.
II. B	<ul style="list-style-type: none"> - Công thức toán học: Similarity Score, Gain, Output, Probability. - Ảnh hưởng của tham số đến quá trình học. - Hiện tượng overfitting và cách khắc phục. 	<ul style="list-style-type: none"> - Tính toán được ví dụ minh họa. - Giải thích ảnh hưởng của tham số. - Nhận biết và xử lý overfitting.

Chương 18

XGBoost và LightGBM

18.1 Giới thiệu

Trong cuộc sống, sóng sau luôn đẩy sóng trước - machine learning cũng không ngừng tiến hóa với những thế hệ mô hình mới. Random Forest mở đường với việc kết hợp nhiều cây quyết định, AdaBoost học từ những sai lầm để cải thiện, Gradient Boosting sử dụng gradient để tối ưu hóa từng bước. Đến năm 2016-2017, khi dữ liệu ngày càng lớn và yêu cầu về tốc độ xử lý trở nên cấp thiết hơn, XGBoost và LightGBM xuất hiện như câu trả lời cho những thách thức mới. Chúng không chỉ đơn thuần là sự cải tiến về mô hình, mà còn thể hiện hai triết lý khác nhau trong cách tiếp cận bài toán machine learning.



Hình 18.1: Khác biệt giữa 2 mô hình XGBoost và LightGBM.

XGBoost, do Tianqi Chen phát triển năm 2016, nhanh chóng trở thành mô hình được ưa chuộng nhờ cân bằng giữa tốc độ và độ chính xác. Điểm đáng chú ý là XGBoost có nhiều cơ chế regularization giúp tránh overfitting, đồng thời tối ưu cách tính toán để chạy nhanh hơn so với nhiều mô hình boosting trước đó.

Ra mắt sau đó một năm, LightGBM của Microsoft lại đi theo hướng khác: thay vì xây cây theo tầng, nó phát triển theo lá để tập trung vào những nhánh cải thiện mô hình nhiều nhất. Kết hợp với chiến lược lấy mẫu thông minh, LightGBM vừa nhanh vừa tiết kiệm bộ nhớ, đặc biệt hiệu quả với dữ liệu lớn.

XGBoost có thể ví như một học sinh chăm chỉ - luôn kiểm tra kỹ lưỡng để tránh sai sót và không ngừng tối ưu hóa cách học. Trong khi đó, LightGBM giống như một học sinh thông minh biết cách ‘lười đúng chỗ’ - tập trung vào những điều quan trọng nhất để đạt kết quả tốt với thời gian ngắn nhất.

18.2 Mô hình XGBoost cho bài toán hồi quy

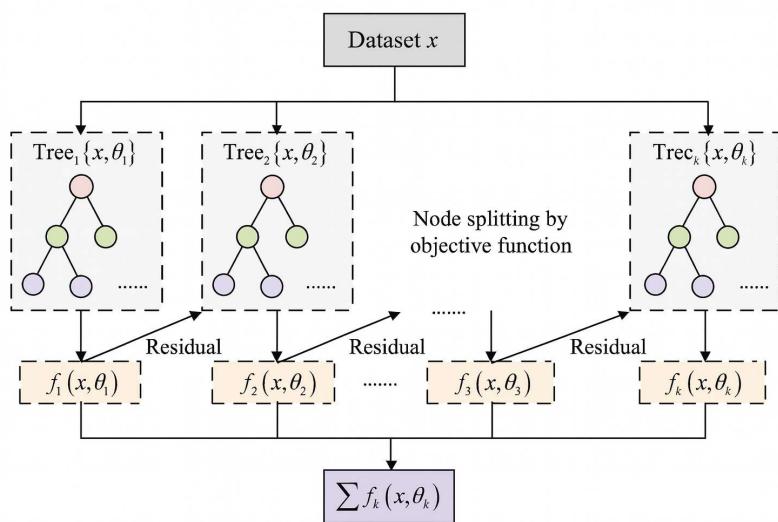
18.2.1 Tổng quan

XGBoost không chỉ mạnh trong phân loại mà còn rất hiệu quả cho các bài toán hồi quy, nơi mục tiêu là dự đoán một giá trị liên tục thay vì gán nhãn vào các nhóm. Trong hồi quy, mô hình được huấn luyện để tối thiểu hóa sai số giữa giá trị dự đoán và giá trị thực tế, thường thông qua các hàm mất mát như Mean Squared Error (MSE) hoặc Mean Absolute Error (MAE). Cơ chế boosting của XGBoost sẽ lần lượt xây dựng các cây quyết định nhỏ, mỗi cây tập trung vào việc ‘sửa chữa’ những phần sai số mà các cây trước đó chưa khắc phục được. Nhờ vậy, sau nhiều vòng lặp, mô hình dần cải thiện và đưa ra kết quả ngày càng chính xác hơn.

Ví dụ, khi dự đoán giá một căn nhà, giá thực tế là 3 tỷ nhưng mô hình XGBoost ban đầu chỉ dự đoán 2.5 tỷ. Mỗi cây tiếp theo sẽ ‘sửa’ phần sai số còn lại, khiến kết quả dần tiệm cận 3 tỷ. Nhờ cách tinh chỉnh liên tục này, XGBoost đặc biệt hiệu quả trong các bài toán hồi quy như dự đoán giá nhà, nhu cầu hay doanh thu.

Ý tưởng chính của Gradient Boosting dựa trên ba nguyên tắc:

- Dự đoán dần dần (Boosting theo sai số)
- Giảm thiểu sai số bằng hàm mất mát
- Regularization để tránh overfitting



Hình 18.2: Mô hình XGBoost.

18.2.2 Các bước trong XGBoost cho bài toán hồi quy

Step 1: Khởi tạo

Khởi tạo giá trị f_0 dự đoán của model bằng cách lấy trung bình của Y (giá trị của target).

Step 2: Tính Similarity Score của root

$$\text{Similarity Score} = \frac{(\text{Sum of Residuals})^2}{\text{Number of Residuals} + \lambda}$$

- Sum of Residuals = $\sum(Y - f_0)$: tổng các giá trị $Y - f_0$.
- Number of Residuals : số lượng sample (còn có thể ký hiệu n hoặc $|\text{node}|$ tại node đang xét).
- λ : regularization parameter giúp giảm overfitting.

Step 3: Chọn điều kiện root và tính Similarity Score cho các node con

Có nhiều cách chọn điều kiện root, cơ bản nhất là lấy trung bình của 2 sample liên tiếp nhau. Sau đó tính Similarity Score cho các node trong nhánh trái và nhánh phải theo công thức ở Step2 (ứng với từng tập mẫu trái/phải).

Step 4: Tính Gain cho từng điều kiện root và chọn lớn nhất

$\text{Gain} = \text{Left Similarity Score} + \text{Right Similarity Score} - \text{Root Similarity Score}$

- Left Similarity Score : Similarity Score tính trên nhánh trái vừa chia.
- Right Similarity Score : Similarity Score tính trên nhánh phải vừa chia.
- Root Similarity Score : Similarity Score tại node gốc trước khi chia.
- Chọn điều kiện có Gain lớn nhất.

Step 5: Lặp lại quá trình và tính Output cho node được chọn

Tùy vào điều kiện độ sâu của tree mà ta sẽ thực hiện chia nhánh bằng cách lặp lại Step2 đến Step4. Sau đó ta đi tìm **Output** cho root theo điều kiện có gain lớn nhất:

$$\text{Output} = \frac{\text{Sum of Residuals}}{\text{Number of Residuals} + \lambda}$$

- Sum of Residuals = $\sum(Y - f_0)$: tổng residuals trong node tương ứng.
- Number of Residuals : số lượng sample trong node tương ứng.

Step 6: Cập nhật dự đoán cho toàn bộ training sample và tiếp tục lặp

Dùng công thức sau để dự đoán kết quả (thay thế cho f_0) và tiếp tục thực hiện Step2 đến Step5 cho đến khi thoả mãn điều kiện dừng:

$$\hat{y} = f_0 + lr \times \text{Output}$$

- f_0 : giá trị khởi tạo ở Step1, sau đó được cập nhật dần.
- lr : learning rate.
- Output : giá trị lá (node) thu được từ Step5, áp dụng cho các sample rơi vào node đó.

18.2.3 Huấn luyện mô hình cho bộ dữ liệu dự đoán giá thuê phòng Airbnb

Bài tập này chúng ta sẽ tập trung vào việc triển khai và đào tạo mô hình XGBoost. Mục tiêu chính là sử dụng mô hình này để giải quyết một bài toán regression, cụ thể là dự đoán giá niêm yết của các chỗ ở trên Airbnb dựa trên những đặc trưng như loại phòng, số người ở, tiện nghi, chính sách huỷ, v.v. Để thực hiện điều này, chúng ta sẽ tuân theo một số bước cụ thể được trình bày sau đây.

Bước 1: Import thư viện cần thiết

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 from sklearn.preprocessing import LabelEncoder, StandardScaler
6 from sklearn.model_selection import train_test_split
7 from sklearn import metrics
8 from xgboost import XGBRegressor
```

Chúng ta sẽ sử dụng các thư viện sau:

- `pandas`, `numpy`: xử lý và tính toán dữ liệu.
- `matplotlib`: vẽ biểu đồ.
- `LabelEncoder`: mã hóa dữ liệu phân loại.
- `StandardScaler`: chuẩn hóa dữ liệu đầu vào.
- `train_test_split`: chia dữ liệu thành tập huấn luyện và kiểm tra.
- `metrics`: đánh giá mô hình.
- `XGBRegressor`: mô hình hồi quy XGBoost.

Bước 2: Tải và đọc dữ liệu

```
1 !gdown 1A6Gf0TBjv19maDutZX_rqslsFi4fg0BF
2 !unzip Airbnb_Data.zip
3
4 df = pd.read_csv("../content/Airbnb_Data.csv")
5 df.head()
```

Trong bước này, chúng ta tải về bộ dữ liệu từ Google Drive, giải nén tệp ZIP và sau đó đọc dữ liệu vào DataFrame bằng pandas. Hàm `df.head()` hiển thị 5 dòng đầu tiên của bộ dữ liệu để kiểm tra.

	id	log_price	property_type	room_type	amenities	accommodates	bathrooms	bed_type	cancellation_policy	cleaning_fee	...	latitude	longitude	name	neighbourhood
0	6901257	5.010635	Apartment	Entire home/apt	{"Wireless Internet","Air conditioning","Kitchen", "Balcony"}	3	1.0	Real Bed	strict	True	...	40.696524	-73.991617	Beautiful brownstone 1-bedroom	Brooklyn Heights
1	6304928	5.129699	Apartment	Entire home/apt	{"Wireless Internet","Air conditioning","Kitchen", "Balcony"}	7	1.0	Real Bed	strict	True	...	40.766115	-73.989040	Superb 3BR Apt Located Near Times Square	Hell's Kitchen
2	7919400	4.976734	Apartment	Entire home/apt	{"TV","Cable TV","Wireless Internet","Air conditioning"}	5	1.0	Real Bed	moderate	True	...	40.808110	-73.943756	The Garden Oasis	Harlem
3	13418779	6.620073	House	Entire home/apt	{"TV","Cable TV","Internet","Wireless Internet","Kitchen", "Balcony"}	4	1.0	Real Bed	flexible	True	...	37.772004	-122.431619	Beautiful Flat in the Heart of SF!	Lower Haight
4	3808709	4.744932	Apartment	Entire home/apt	{"TV,Internet","Wireless Internet","Air conditioning"}	2	1.0	Real Bed	moderate	True	...	38.925627	-77.034596	Great studio in midtown DC	Columbia Heights

Hình 18.3: Một vài mẫu dữ liệu trong bộ data Airbnb.

Bước 3: Khám phá dữ liệu

```
1 df.shape
2 df.info()
3 df = df.drop(["id", "name", "description", "thumbnail_url", "zipcode",
4                 "neighbourhood"], axis=1)
5 df = df.dropna()
categorical_cols = df.select_dtypes(include=["object", "bool"]).
    columns.to_list()
```

Trong bước này, chúng ta thực hiện các thao tác sau:

- `df.shape`: Kiểm tra số lượng dòng và cột của dữ liệu.

- `df.info()`: Xem thông tin chi tiết về các cột và kiểu dữ liệu, bao gồm số lượng giá trị thiếu.
- `df.drop(...)`: Loại bỏ các cột không cần thiết như `id`, `name`, `description`, `thumbnail_url`, `zipcode`, `neighbourhood`, các cột này chứa nhiều giá trị null, và thông tin không hữu ích cho mô hình.
- `df.dropna()`: Loại bỏ các dòng có giá trị thiếu (null) để đảm bảo dữ liệu sạch khi huấn luyện mô hình.
- `df.select_dtypes(include=["object", "bool"])`: Lọc các cột có kiểu dữ liệu là `object` (dữ liệu phân loại) và `bool` (boolean), rồi lưu tên các cột này vào danh sách `categorical_cols`.

Bước 4: Tiền xử lý dữ liệu

```

1 # B1. Chuyển cột 'host_since', 'first_review', 'last_review' thành
      datetime (nếu chưa)
2 df['host_since'] = pd.to_datetime(df['host_since'], errors='coerce')
3 df['first_review'] = pd.to_datetime(df['first_review'], errors='
      coerce')
4 df['last_review'] = pd.to_datetime(df['last_review'], errors='coerce
      ')
5
6 # Tính số ngày từ 'host_since', 'first_review', 'last_review' đến ngày
      hiện tại
7 df['host_since'] = (pd.to_datetime('today') - df['host_since']).dt.
      days
8 df['first_review'] = (pd.to_datetime('today') - df['first_review']).
      dt.days
9 df['last_review'] = (pd.to_datetime('today') - df['last_review']).dt
      .days
10
11 #B2. Loại bỏ ký tự '%' và chuyển thành số cho 'host_response_rate'
12 df['host_response_rate'] = df['host_response_rate'].str.rstrip('%').
      astype(float) / 100.0
13 #B3. Tách và đếm số lượng tiện ích
14 df['amenities'] = df['amenities'].apply(lambda x: len(x.strip('{}')).
      split(','))
15 #B4. Khởi tạo LabelEncoder
16 le = LabelEncoder()

```

```
17 categorical_cols_new=['property_type',
18 'room_type',
19 'amenities',
20 'bed_type',
21 'cancellation_policy',
22 'cleaning_fee',
23 'city',
24 'host_has_profile_pic',
25 'host_identity_verified',
26 'instant_bookable']
27
28 # Sử dụng LabelEncoder cho các cột phân loại
29 for col in categorical_cols_new:
30     df[col] = le.fit_transform(df[col].astype(str))
```

Trong bước này, chúng ta thực hiện các thao tác sau:

- **Chuyển đổi cột ngày tháng:**

Chuyển các cột ‘host_since’, ‘first_review’, và ‘last_review’ thành kiểu dữ liệu datetime nếu chưa phải.

Tính toán số ngày kể từ ngày hiện tại đến ngày trong các cột này, giúp mô hình có dữ liệu dạng số (thay vì dữ liệu kiểu chuỗi).

- **Xử lý cột tỷ lệ phản hồi của host:**

Loại bỏ ký tự % từ cột ‘host_response_rate’ và chuyển đổi giá trị sang dạng số thực, rồi chia cho 100 để có tỷ lệ từ 0 đến 1.

- **Đếm số lượng tiện ích:**

Tách và đếm số lượng tiện ích trong cột ‘amenities’ bằng cách loại bỏ dấu ngoặc và phân tách theo dấu phẩy, rồi tính số lượng tiện ích.

- **Mã hóa các cột phân loại:**

Sử dụng LabelEncoder để chuyển các cột phân loại như ‘property_type’, ‘room_type’, ‘city’... thành dạng số cho mô hình học máy. Mỗi giá trị duy nhất trong các cột này sẽ được mã hóa thành một số nguyên.

	property_type	room_type	amenities	bed_type	cancellation_policy	cleaning_fee	city	first_review	host_has_profile_pic	host_identity_verified	host_response_rate	host_since	instant_bookable	last_review	
1	0	0	6	4		2	1	4	2958	1	0	1.0	3005	1	2909
2	0	0	10	4		1	1	4	3055	1	1	1.0	3242	1	2918
4	0	0	3	4		1	1	2	3774	1	1	1.0	3846	1	3153
5	0	1	1	4		2	1	5	2936	1	1	1.0	3017	1	2927
6	0	0	13	4		1	1	3	3106	1	0	1.0	3113	1	3064

Hình 18.4: Một vài mẫu dữ liệu đã được tiền xử lý.

Bước 5: Chia dữ liệu train/test

```

1 x = # Your code here
2 y = # Your code here
3 x_train, x_test, y_train, y_test = # Your code here
4
5 normalizer = StandardScaler()
6 x_train_scaled = normalizer.fit_transform(x_train)
7 x_test_scaled = normalizer.transform(x_test)

```

Trong bước này, chúng ta sẽ tách dữ liệu thành hai phần:

- **x:** Chứa tất cả các cột trong DataFrame ngoại trừ cột ‘log_price’, vì chúng ta chỉ muốn sử dụng các đặc trưng (features) để dự đoán giá.
- **y:** Chứa giá trị mà mô hình cần dự đoán, chính là cột ‘log_price’.

Chúng ta chọn tỷ lệ chia dữ liệu là 80% cho tập huấn luyện (**train**) và 20% cho tập kiểm tra (**test**). Tham số **random_state=42** giúp cố định tính ngẫu nhiên để kết quả có thể tái lặp lại.

Việc chuẩn hóa dữ liệu bằng **StandardScaler** giúp các đặc trưng có cùng thang đo, tránh việc một số đặc trưng có giá trị lớn chi phối mô hình, từ đó có thể tăng hiệu quả học và độ chính xác của dự đoán.

Bước 6: Xây dựng mô hình XGBoost Regressor

```

1 xgb = # Your code here
2 xgb.fit(x_train_scaled, y_train)

```

Sau khi chia dữ liệu thành train/test, chúng ta xây dựng mô hình XGBRegressor. Tham số `seed=42` giúp cố định tính ngẫu nhiên để kết quả có thể tái lặp lại.

- Tạo mô hình XGBRegressor với các tham số sau: `seed=42, learning_rate=0.3, n_estimators=100, max_depth=6`.
- `xgb.fit(x_train_scaled, y_train)`: Huấn luyện mô hình với tập huấn luyện đã chuẩn hóa.

Bước 7: Dự đoán trên tập test

```
1 y_pred_xgb = xgb.predict(x_test_scaled)
```

Chúng ta sử dụng mô hình XGBoost đã huấn luyện để dự đoán giá trị mục tiêu cho các mẫu dữ liệu mới (ở đây là tập test đã chuẩn hóa).

Bước 8: Đánh giá mô hình

```
1 mae_xgb = metrics.mean_absolute_error(y_test, y_pred_xgb)
2 mse_xgb = metrics.mean_squared_error(y_test, y_pred_xgb)
3 rmse_xgb = np.sqrt(metrics.mean_squared_error(y_test, y_pred_xgb))
4 r2_xgb = metrics.r2_score(y_test, y_pred_xgb)
5
6 print('Mean Absolute Error of XGBoost Regressor : ', mae_xgb)
7 print('Mean Squared Error of XGBoost Regressor : ', mse_xgb)
8 print('Root Mean Squared Error of XGBoost Regressor : ', rmse_xgb)
9 print('R2 Score of XGBoost Regressor : ', r2_xgb)
```

Sau khi huấn luyện mô hình, chúng ta sử dụng các chỉ số sau để đánh giá hiệu quả của mô hình:

- **MAE (Mean Absolute Error)**: Đo lường sai số trung bình giữa giá trị thực và giá trị dự đoán. MAE càng thấp, mô hình càng chính xác.
- **MSE (Mean Squared Error)**: Tính trung bình bình phương sai số giữa giá trị thực và dự đoán. MSE phạt sai số lớn.

- **RMSE (Root Mean Squared Error):** Căn bậc hai của MSE, giúp đo sai số với đơn vị giống dữ liệu.
- **R2 Score (R-squared):** Đo khả năng giải thích sự biến động của dữ liệu, giá trị gần 1 thể hiện mô hình giải thích tốt.

18.3 Mô hình LightGBM

18.3.1 Tổng quan

LightGBM là một biến thể của GBDT (Gradient-Based Decision Tree), được Microsoft giới thiệu với trọng tâm tối ưu hoá tốc độ và khả năng mở rộng.

Nền tảng của LightGBM được xây dựng dựa trên hai kỹ thuật then chốt: **Gradient-based One-Side Sampling (GOSS)** và **Exclusive Feature Bundling (EFB)**. Trong đó:

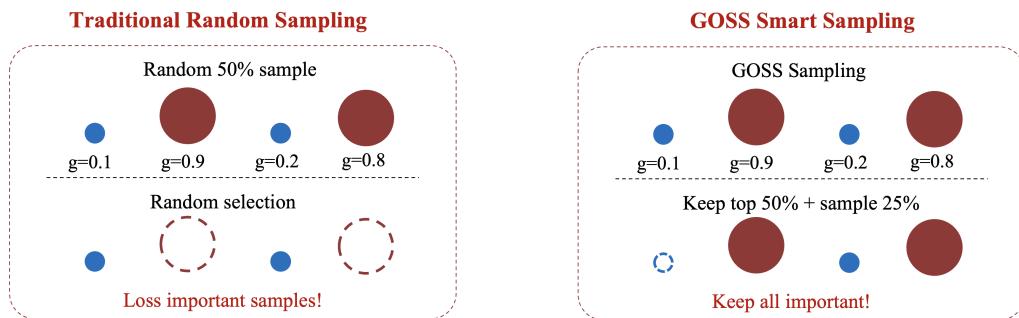
- **GOSS**: ưu tiên giữ lại các mẫu có độ lớn gradient cao, lấy mẫu ngẫu nhiên một phần từ phần còn lại, rồi tái trọng số khi ước lượng độ lợi thông tin (information gain) để hạn chế sai lệch.
- **EFB**: thay vì dùng các phép biến đổi như PCA để giảm chiều, LightGBM hạ chi phí theo số đặc trưng bằng cách gộp những đặc trưng hầu như không cùng kích hoạt (hiếm khi đồng thời có giá trị khác 0) thông qua bài toán tô màu đồ thị (graph coloring).

So với XGBoost, LightGBM cho kết quả huấn luyện thường nhanh hơn và tiết kiệm bộ nhớ hơn mà vẫn đảm bảo kết quả tương đương.

18.3.2 GOSS và EFB

Cách hoạt động của **LightGBM** nhìn chung khá tương đồng với **XGBoost** nhưng có một vài kỹ thuật bổ sung sau:

GOSS



Hình 18.5: Minh họa kỹ thuật GOSS

Ý tưởng: Độ dốc (gradient) của từng mẫu dữ liệu thể hiện mức độ “chưa được giải quyết” của mô hình. Những điểm có gradient nhỏ đóng góp rất ít vào cập nhật tham số, nên có thể giảm tần suất xuất hiện của chúng mà vẫn giữ nguyên hướng học. LightGBM khai thác điều này bằng **Gradient-based One-Side Sampling (GOSS)**: giữ trọn các mẫu có gradient lớn, chỉ lấy ngẫu nhiên một phần nhỏ trong nhóm gradient thấp, rồi hiệu chỉnh trọng số để không làm sai lệch phép đánh giá split gain.

Quy trình:

1. Sắp xếp theo độ lớn gradient $|g_i|$, giữ lại $a \cdot n$ điểm có $|g_i|$ lớn nhất.
2. Lấy ngẫu nhiên $b \cdot n$ điểm từ phần còn lại.
3. Nhân trọng số các mẫu ở bước (2) với hệ số

$$\frac{1-a}{b}.$$

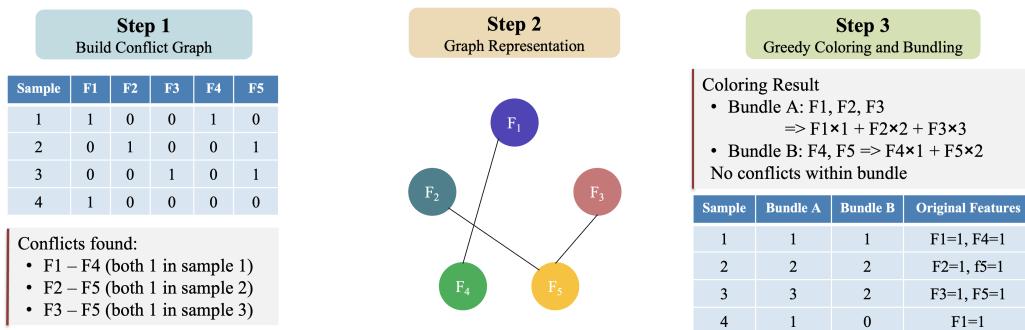
để tăng cường các dữ liệu đó, tránh mất cân bằng khi huấn luyện.

4. Huấn luyện cây quyết định trên tập dữ liệu gộp lại.

Lợi ích: giảm mạnh số mẫu tham gia xây histogram nhưng vẫn giữ gần đúng tốt cho ước lượng gain.

EFB

Ý tưởng: nhiều đặc trưng rời rạc (sparse) hầu như không cùng khác 0 (ví dụ: one-hot). Ta gom chúng vào một “bundle” duy nhất, sao cho mỗi bản ghi chỉ kích hoạt đúng một (hoặc rất ít) thành phần trong bundle.



Hình 18.6: Minh họa kỹ thuật EFB với graph coloring

Quy trình:

1. Dụng đồ thị xung đột giữa các đặc trưng.
2. Thực hiện *greedy bundling* cho phép xung đột nhỏ γ .
3. Map các đặc trưng trong cùng một bundle vào các dải bin không chồng lấn.

Lợi ích: Độ phức tạp xây histogram được giảm từ

$$O(N \cdot F) \text{ thành } O(N \cdot B), \text{ với } B \ll F,$$

trong đó N là số mẫu, F là số đặc trưng ban đầu và B là số bundle sau khi gom.

18.3.3 Huấn luyện mô hình LightGBM cho bài toán hồi quy - Bộ dữ liệu Airbnb

Các bước trong LightGBM cho bài toán hồi quy

Step 0: Tiết xử lý (đặc trưng của LightGBM)

- Lượng tử hoá mỗi đặc trưng liên tục vào K bin.
- (Tuỳ chọn) EFB: gộp các đặc trưng “hiếm khi cùng khác 0” để giảm số chiều.

Step 1: Khởi tạo

$$f^{(0)} = \frac{1}{n} \sum_{i=1}^n y_i$$

Khởi tạo dự đoán bằng trung bình Y (với hồi quy L2).

Step 2: Tính gradient và hessian Với L2 loss:

$$g_i = \hat{y}_i - y_i, \quad h_i = 1.$$

Tại node T :

$$G_T = \sum_{i \in T} g_i, \quad H_T = \sum_{i \in T} h_i.$$

Step 3: (tuỳ chọn): GOSS

1. Giữ tất cả điểm có $|g_i|$ lớn nhất (top a).
2. Lấy ngẫu nhiên b phần trăm từ phần còn lại.
3. Tái trọng số nhóm nhỏ-gradient bằng $\frac{1-a}{b}$.

Step 4: Xây histogram và chọn điều kiện tách

$$\text{Gain} = \frac{1}{2} \left(\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G_T^2}{H_T + \lambda} \right) - \gamma.$$

Chọn điều kiện tách có Gain lớn nhất.

Step 5: Mở rộng cây leaf-wise (best-first)

- Đưa node con vào hàng đợi ưu tiên theo gain tiềm năng.
- Lặp Step 2–4 trên node có gain lớn nhất.
- Dừng khi hết gain dương hoặc chạm ràng buộc (`num_leaves`, `max_depth`, ...).

Step 6: Tính giá trị lá (Newton step)

$$v_L = -\frac{G_L}{H_L + \lambda}.$$

Step 7: Cập nhật dự đoán

$$f^{(m)}(x) = f^{(m-1)}(x) + \eta \cdot v_{L(x)},$$

trong đó $v_{L(x)}$ là giá trị lá mà x rơi vào, η là learning rate.

Step 8: Dự đoán cuối cùng

$$\hat{y}(x) = f^{(0)}(x) + \sum_{m=1}^M \eta v_{L_m(x)}^{(m)}.$$

Huấn luyện mô hình cho bộ dữ liệu dự đoán giá thuê nhà Airbnb

Chúng ta sẽ áp dụng LightGBM vào bài toán hồi quy, cụ thể là bộ dữ liệu dự đoán giá thuê nhà Airbnb.

Bước 1: Import thư viện cần thiết

Trong bước này, chúng ta sẽ import các thư viện hỗ trợ phân tích dữ liệu, trực quan hóa và xây dựng mô hình:

- `numpy`, `pandas`: xử lý dữ liệu dạng số và bảng.
- `matplotlib.pyplot`: vẽ biểu đồ, trực quan hóa dữ liệu.
- `sklearn.metrics`: cung cấp hàm `mean_squared_error`, `mean_absolute_error` để đánh giá dự đoán của mô hình.

- `lightgbm`: thư viện LightGBM
- `sklearn.preprocessing`: cung cấp `LabelEncoder` để chuyển dữ liệu category về số.
- `sklearn.model_selection`: dùng để chia tập dữ liệu thành train/test.

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4
5 import lightgbm as lgb
6 from sklearn.metrics import mean_squared_error, mean_absolute_error
7 from sklearn.preprocessing import LabelEncoder
8 from sklearn.model_selection import train_test_split

```

Bước 2: Khám phá dữ liệu Chúng ta sử dụng `read_csv()` để đọc dữ liệu và hiển thị nội dung của một vài dòng đầu của bộ dữ liệu.

```

1 !gdown 1A6Gf0TBJv19maDutZX_rqslsFi4fgoBF
2 !unzip Airbnb_Data.zip
3
4 dataset_path = path + "/Airbnb_Data.csv"
5 data_df = pd.read_csv(dataset_path)
6 data_df.head()

```

Bước 3: Tiền xử lý dữ liệu Kiểm tra số giá trị null của mỗi đặc trưng

```
1 data_df.isnull().sum()
```

Chúng ta tiến hành điền các giá trị nan cho một số đặc trưng bằng cách propagate giá trị khác nan trước đó bằng đoạn code sau:

```

1 data_df.last_review.fillna(method="ffill", inplace=True)
2 data_df.first_review.fillna(method="ffill", inplace=True)
3 data_df.host_since.fillna(method="ffill", inplace=True)

```

Một số đặc trưng khác với kiểu dữ liệu dạng số, ta thực hiện điền các giá trị nan bằng giá trị trung bình của các dòng khác hoặc bằng giá trị 0.

```

1 data_df["bathrooms"] = data_df["bathrooms"].fillna(round(data_df["bathrooms"].median()))
2 data_df["bedrooms"] = data_df["bedrooms"].fillna((data_df["bedrooms"].median()))
3 data_df["beds"] = data_df["beds"].fillna((data_df["beds"].median()))
4 data_df["review_scores_rating"] = data_df["review_scores_rating"].fillna(0)

```

Với các đặc trưng category, ta sử dụng Label Encoder để chuyển về dạng số.

```

1 categorical_col = []
2 numerical_col = []
3 for column in data_df.columns:
4     if data_df[column].dtypes != "float64" and data_df[column].dtypes != "int64":
5         categorical_col.append(column)
6     else:
7         numerical_col.append(column)
8 le = # Your code here
9 for col in categorical_col:
10    data_df[col] = le.fit_transform(data_df[col])

```

Bước 4: Chia dữ liệu train/test

Để đánh giá mô hình một cách khách quan, dữ liệu được chia thành 2 tập:

- **train**: dùng để huấn luyện mô hình.
- **test**: dùng để kiểm tra khả năng tổng quát hóa của mô hình.

```

1 from sklearn.model_selection import train_test_split
2
3 x = data_df[top_corr_features]
4 y = data_df["log_price"]
5 x_train, x_test, y_train, y_test = # Your code here

```

Bước 5: Huấn luyện mô hình

Khởi tạo mô hình LightGBM và fit vào tập dữ liệu X_train, y_train

```

1 import lightgbm as lgb
2
3 lgb_reg = # Your code here
4 lgb_reg.fit(x_train, y_train)

```

Bước 6: Đánh giá mô hình Chúng ta sẽ đánh giá mô hình bằng cách dự đoán trên tập X_test và tính toán các hàm mục tiêu như Mean Squared Error, Mean Absolute Error.

```

1 y_pred = lgb_reg.predict(x_test)
2 mae = mean_absolute_error(y_test, y_pred)
3 mse = mean_squared_error(y_test, y_pred)
4
5 print('Evaluation results on test set:')
6 print(f'Mean Absolute Error: {mae}')
7 print(f'Mean Squared Error: {mse}')

```

18.3.4 Huấn luyện mô hình cho bài toán phân loại - bộ dữ liệu phân loại chất lượng rượu vang

Bài tập này chúng ta sẽ tập trung vào việc triển khai và đào tạo mô hình LightGBM. Mục tiêu chính của chúng ta là sử dụng mô hình này để giải quyết một bài toán classification, cụ thể là phân loại chất lượng rượu vang (3 classes: 0, 1, 2). Để thực hiện điều này, chúng ta sẽ tuân theo một số bước cụ thể mà sẽ được trình bày sau đây.

Bước 1: Import thư viện cần thiết

Trong bước này, chúng ta sẽ import các thư viện hỗ trợ phân tích dữ liệu, trực quan hóa và xây dựng mô hình:

- numpy, pandas: xử lý dữ liệu dạng số và bảng.
- matplotlib.pyplot: vẽ biểu đồ, trực quan hóa dữ liệu.

- `sklearn.metrics`: cung cấp hàm `accuracy_score` để đánh giá độ chính xác.
- `lightgbm`: thư viện LightGBM
- `sklearn.model_selection`: dùng để chia tập dữ liệu thành train/test.

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import lightgbm as lgb
5 from sklearn.datasets import load_wine
6 from sklearn.metrics import accuracy_score
7 from sklearn.preprocessing import LabelEncoder
8 from sklearn.model_selection import train_test_split

```

Bước 2: Load dữ liệu

Chúng ta sử dụng phương thức `read_csv()` của pandas để đọc dữ liệu từ một tệp .csv. (Tải data tại [đây](#)). Khi dữ liệu được load thành công, chúng ta có một DataFrame với các cột đặc trưng (features) và một cột nhãn (target).

```

1 # Load dataset
2 wine = load_wine()
3 data_df = pd.DataFrame(wine.data, columns=wine.feature_names)
4 data_df["target"] = wine.target
5 data_df

```

Khi đó, ta có thể thấy nội dung của bảng dữ liệu có dạng như sau:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	Target	
0	14.23	1.71	2.43		15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
1	13.20	1.78	2.14		11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
2	13.16	2.36	2.67		18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
3	14.37	1.95	2.50		16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
4	13.24	2.59	2.87		21.0	116.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0

Hình 18.7: Một vài mẫu dữ liệu trong data.

Trong bài toán này:

- Cột "Target" là biến mục tiêu (label), thể hiện chất lượng rượu vang với ba mức (0, 1, 2).

- Các cột còn lại là các đặc trưng hóa học của rượu vang, được dùng làm dữ liệu đầu vào (X).

Bước 3: Khám phá dữ liệu

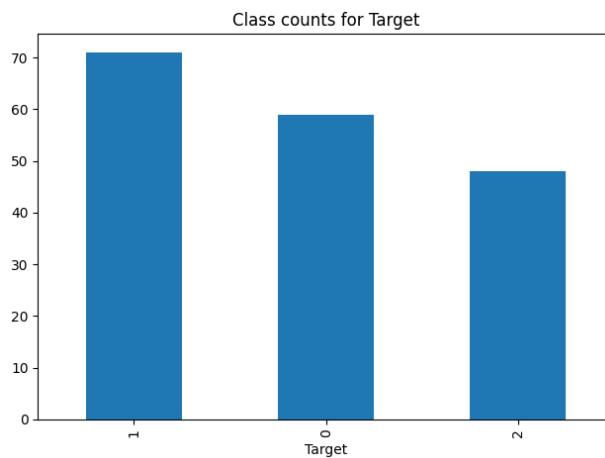
Ở bước này, chúng ta sẽ thực hiện kiểm tra các đặc trưng của dữ liệu:

Ở đây để đơn giản (data cũng không nhiều), chúng ta sẽ kiểm tra sự cân bằng giữa các class

```

1 def class_balance(df: pd.DataFrame, target: str):
2     print(f"Target column: {target}")
3     vc = df[target].value_counts(dropna=False).to_frame("count")
4     vc["pct"] = (vc["count"] / len(df) * 100).round(2)
5     print(vc)
6     # bar plot
7     vc["count"].plot(kind="bar")
8     plt.title(f"Class counts for {target}")
9     plt.tight_layout()
10    plt.show()
11    return vc
12
13 class_balance(data_df, "Target")

```



Hình 18.8: Class Balance.

Hình ảnh cho thấy các dữ liệu chủ yếu là class 0 và ít nhất là class 2. Tuy

nhiên do bộ dữ liệu này không có quá nhiều data nên sự chênh lệch này là có thể chấp nhận.

Bước 4: Chia dữ liệu train/test

Để đánh giá mô hình một cách khách quan, dữ liệu được chia thành 2 tập:

- **train**: dùng để huấn luyện mô hình.
- **test**: dùng để kiểm tra khả năng tổng quát hóa của mô hình.

Chúng ta chọn tỷ lệ 70% train và 30% test.

```
1 X_train, X_test, y_train, y_test = train_test_split(  
2     # Your code here  
3 )
```

Bước 5: Xây dựng mô hình LightGBM Classifier

Sau khi có tập train/test, chúng ta tạo một mô hình LightGBM. Tham số `seed=7` được dùng để cố định tính ngẫu nhiên, giúp kết quả có thể tái lặp lại.

```
1 lgb_class = # Your code here  
2 lgb_class.fit(X_train, y_train)
```

Bước 6: Dự đoán trên tập test

Khi mô hình đã được huấn luyện, ta có thể sử dụng nó để dự đoán nhãn cho các mẫu dữ liệu mới (ở đây là tập test).

```
1 preds = lgb_class.predict(X_test)
```

Bước 7: Đánh giá mô hình

Trong bài toán phân loại (Classification), một độ đo phổ biến là Accuracy:

$$\text{Accuracy} = \frac{\text{Số mẫu phân loại đúng}}{\text{Tổng số mẫu}}$$

Accuracy cho biết tỷ lệ dự đoán đúng của mô hình. Đây là chỉ số đơn giản nhưng hữu ích để đánh giá ban đầu. Trong các trường hợp dữ liệu mất cân bằng (một lớp có nhiều mẫu hơn các lớp khác), chúng ta cần bổ sung thêm các chỉ số khác như Precision, Recall, F1-score, ROC, AUC.

```
1 train_acc = accuracy_score(y_train, lgb_class.predict(X_train))
2 test_acc = accuracy_score(y_test, preds)
3
4 print(f"Train ACC: {train_acc}")
5 print(f"Test ACC: {test_acc}")
```

18.4 Câu hỏi trắc nghiệm

1. Mục tiêu chính của mỗi cây quyết định mới được thêm vào trong thuật toán XGBoost cho bài toán hồi quy là gì?
 - (a) Mô hình hóa trực tiếp biến mục tiêu Y.
 - (b) Sửa chữa sai số (phần dư) của các cây trước đó.
 - (c) Tăng độ chêch (bias) cho toàn bộ mô hình.
 - (d) Giảm số lượng đặc trưng được sử dụng.
2. Cho một tập dữ liệu với các giá trị mục tiêu $Y = [10, 20, 30, 40]$. Giá trị dự đoán khởi tạo f_0 của mô hình XGBoost là bao nhiêu?
 - (a) 20
 - (b) 25
 - (c) 30
 - (d) 100
3. Với giá trị mục tiêu $Y = [2, 4, 8]$, giá trị khởi tạo $f_0 = 5$ và tham số điều chỉnh $\lambda = 1$. Hãy tính Similarity Score của node gốc (root).
 - (a) -3.0
 - (b) 2.25
 - (c) 0.25
 - (d) 1.0
4. Giả sử Similarity Score của node gốc là 12.0. Sau khi phân chia theo một điều kiện, Similarity Score của node con bên trái là 7.5 và node con bên phải là 8.5. Gain thu được từ việc phân chia này là bao nhiêu?
 - (a) 1.0
 - (b) 4.0
 - (c) 16.0
 - (d) 28.0

5. Một node lá trong cây chứa các mẫu có phần dư (residuals) tương ứng là $[1.5, 2.5, -1.0]$ với $\lambda = 0$. Giá trị Output của node lá này là bao nhiêu?
 - (a) 3.0
 - (b) 1.5
 - (c) 1.0
 - (d) 0.5
6. GOSS (Gradient-based One-Side Sampling) trong LightGBM thực hiện chức năng nào?
 - (a) Lấy mẫu ngẫu nhiên toàn bộ dữ liệu rồi huấn luyện trên phân đoạn này.
 - (b) Giữ mọi mẫu có $|g_i|$ lớn và lấy mẫu ngẫu nhiên từ phần còn lại, sau đó tái trọng số các mẫu nhỏ-gradient.
 - (c) Gộp các đặc trưng thừa thành các bundle.
 - (d) Thay thế histogram bằng exact split search.
7. Bạn có một tập dữ liệu nhiều feature one-hot (khoảng 10k categories \rightarrow 10k cột), nhưng mỗi bản ghi chỉ có vài cột khác 0. Để giảm dùng bộ nhớ và tăng tốc, bạn nên:
 - (a) Dùng EFB để gom các cột sparse vào bundle.
 - (b) Bỏ một nửa các cột ngẫu nhiên.
 - (c) Chuyển tất cả sang float32 và train như bình thường.
 - (d) Thay tree bằng logistic regression.
8. Với tổng số mẫu $n = 10000$, chọn tham số GOSS với $a = 0.2$ và $b = 0.1$. Hỏi LightGBM giữ bao nhiêu mẫu có gradient lớn và chọn bao nhiêu mẫu từ phần còn lại?
 - (a) 2000 và 1000
 - (b) 800 và 2000
 - (c) 1000 và 2000
 - (d) 2000 và 800

9. Cho gradients $g = [0.9, 0.2, 0.1, 0.05]$ (4 mẫu). Dùng GOSS với $a = 0.5$ và $b = 0.25$. Giả sử từ phần còn lại ta chọn mẫu có gradient 0.1. Hệ số tái trọng số là $(1 - a)/b$. Tổng gradient hiệu dụng sau GOSS (tính trên các mẫu được giữ, với các gradient nhỏ được nhân hệ số tái trọng số) là bao nhiêu?
- (a) 1.40
 - (b) 1.20
 - (c) 1.30
 - (d) 1.10
10. EFB chỉ được áp dụng cho các đặc trưng one-hot và không thể áp dụng cho các đặc trưng liên tục.
- (a) Đúng
 - (b) Sai

- 1. Hint:** Các file code gợi ý và dữ liệu nếu có được lưu trong thư mục có thể được tải [tại đây](#).
- 2. Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. Rubric:

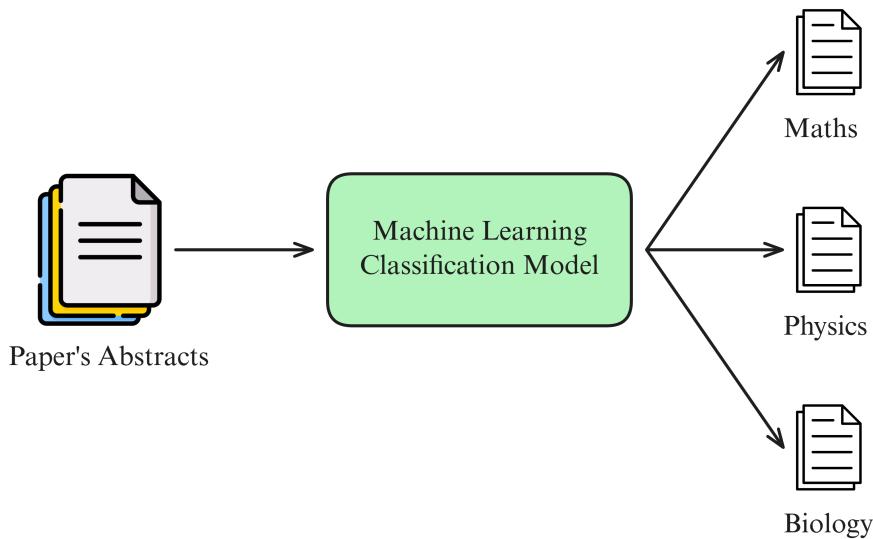
Mục	Kiến Thức	Đánh Giá
II. A	<ul style="list-style-type: none"> - Nguyên lý XGBoost trong bài toán Regression. - Các bước chính: khởi tạo f_0, tính Similarity, chọn split, tính Gain và Output. - Ý nghĩa của tham số λ, learning rate, depth. 	<ul style="list-style-type: none"> - Nắm quy trình tổng quát của XGBoost. - Hiểu vai trò của từng bước và từng tham số.
II. B	<ul style="list-style-type: none"> - Thuật toán LightGBM áp dụng cho phân loại và hồi quy - Ảnh hưởng của tham số đến quá trình học. - Hiện tượng overfitting và cách khắc phục. 	<ul style="list-style-type: none"> - Tính toán được ví dụ minh họa. - Giải thích ảnh hưởng của tham số. - Nhận biết và xử lý overfitting.

Chương 19

Project 1: Phân loại khả năng mắc bệnh tim dựa vào các triệu trứng (dùng các giải thuật đã học)

19.1 Giới thiệu

Text Classification (Tạm dịch: Phân loại văn bản) là một trong những bài toán cơ bản và quan trọng trong lĩnh vực Xử lý ngôn ngữ tự nhiên (NLP). Mục tiêu của bài toán này là phân loại các đoạn văn bản vào các nhóm hoặc nhãn khác nhau dựa trên nội dung của chúng. Các ứng dụng phổ biến của Text Classification bao gồm phân loại email (spam vs. ham), phân loại tin tức, phân loại cảm xúc (sentiment analysis), và nhiều ứng dụng khác.



Hình 19.1: Minh họa ứng dụng phân loại topic của paper dựa trên abstract.

Trong project này, chúng ta sẽ xây dựng một chương trình Text Classification liên quan đến việc phân loại một abstract của publication (bài báo khoa học)

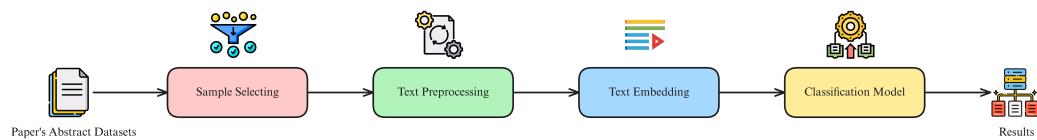
thành các topic khác nhau. Chương trình sẽ được xây dựng trên nền tảng Python và sử dụng các thư viện học máy phổ biến như scikit-learn, numpy, v.v.

Theo đó, Input/Output chung của chương trình sẽ bao gồm:

- **Input:** Một abstract của publication (bài báo khoa học).
- **Output:** Topic của abstract đó (ví dụ: vật lý, toán học, khoa học máy tính, v.v.).

19.2 Xây dựng chương trình phân loại topic của publication abstract

Trong phần này, ta sẽ tiến hành cài đặt chương trình phân loại topic của publication abstract. Chương trình sẽ được xây dựng dựa trên ba phương pháp mã hóa văn bản khác nhau, bao gồm Bag-of-Words (BoW), TF-IDF và Sentence Embeddings. Mỗi phương pháp sẽ được áp dụng với một mô hình phân loại khác nhau, bao gồm Naive Bayes, KNN và Decision Tree.



Hình 19.2: Pipeline

19.2.1 Cài đặt và Import thư viện

Chúng ta import các thư viện cần thiết để thực hiện các bước tiền xử lý, trích xuất đặc trưng, huấn luyện mô hình và đánh giá mô hình. Các thư viện này bao gồm:

```

1 from collections import Counter, defaultdict
2 from typing import List, Dict, Literal, Union
3
4 import re
5 import math
6 import numpy as np
7 import matplotlib.pyplot as plt
8 import seaborn as sns
9
10 from datasets import load_dataset
11 from sentence_transformers import SentenceTransformer
12
13 from sklearn.model_selection import train_test_split
14 from sklearn.metrics import accuracy_score, classification_report,
15                                         confusion_matrix
16 from sklearn.feature_extraction.text import CountVectorizer,
17                                         TfidfVectorizer

```

```
16 from sklearn.cluster import KMeans
17 from sklearn.neighbors import KNeighborsClassifier
18 from sklearn.tree import DecisionTreeClassifier
19 from sklearn.naive_bayes import GaussianNB
20
21 import warnings
22
23 warnings.filterwarnings("ignore")
24
25 CACHE_DIR = "./cache"
```

Trước tiên, chúng ta cùng điểm qua một số thư viện chính được sử dụng trong bài:

- **re:** Thư viện để làm việc với biểu thức chính quy (regular expressions), giúp xử lý và phân tích chuỗi văn bản.
- **datasets:** Thư viện để tải và làm việc với các bộ dữ liệu từ Hugging Face.
- **sentence-transformers:** Thư viện để tạo và sử dụng các mô hình sentence embeddings, giúp chuyển đổi văn bản thành các vector ngữ nghĩa.
- **matplotlib.pyplot:** Thư viện để vẽ đồ thị và biểu đồ, hỗ trợ trực quan hóa dữ liệu.
- **seaborn:** Thư viện vẽ đồ thị thống kê, cung cấp các hàm tiện ích để trực quan hóa dữ liệu một cách đẹp mắt.
- **numpy:** Cung cấp các đối tượng mảng đa chiều và các hàm toán học để làm việc với các mảng này.
- **scikit-learn:** Thư viện học máy phổ biến, giúp xây dựng và triển khai các mô hình học máy phức tạp một cách nhanh chóng.

19.2.2 Đọc và khám phá bộ dữ liệu

Trong bài này, chúng ta sẽ sử dụng bộ dữ liệu [UniverseTBD/arxiv-abstracts-large](#) được cung cấp trên HuggingFace. Bộ dữ liệu này bao gồm các abstract

(tóm tắt) của các bài báo khoa học được đăng trên arXiv, một kho lưu trữ trực tuyến cho các bài báo khoa học trong nhiều lĩnh vực khác nhau. Bộ dữ liệu này có thể được sử dụng để phân loại các abstract theo các chủ đề khác nhau hoặc để phân tích nội dung của các bài báo khoa học, bao gồm các thông tin như tiêu đề, tác giả, ngày cập nhật, v.v. Bộ dữ liệu này có kích thước lớn với hơn 2 triệu bản ghi.

Chúng ta load bộ dữ liệu này bằng thư viện `datasets` với hàm `load_dataset` và in ra một số thông tin cơ bản về bộ dữ liệu như sau:

```
1 ds = load_dataset("UniverseTBD/arxiv-abstracts-large")
2 ds
```

Nội dung được in ra màn hình như sau:

```
DatasetDict(
  train: Dataset(
    features: ['id', 'submitter', 'authors', 'title', 'comments', 'journal-ref',
               'doi', 'report-no', 'categories', 'license', 'abstract', 'versions', 'update_-_
date', 'authors_parsed'],
    num_rows: 2292057
  )
)
```

Các fields của bộ dữ liệu bao gồm:

- **id:** ID của paper trên arXiv.
- **submitter:** Người nộp bài báo.
- **authors:** Danh sách các tác giả của bài báo.
- **title:** Tiêu đề của bài báo.
- **comments:** Các bình luận liên quan đến bài báo.
- **journal-ref:** Tham chiếu đến tạp chí nơi bài báo được xuất bản.
- **doi:** Digital Object Identifier, mã định danh duy nhất cho bài báo.

- **report-no:** Số báo cáo liên quan đến bài báo.
- **categories:** Các chủ đề hoặc lĩnh vực mà bài báo thuộc về.
- **license:** Giấy phép sử dụng của bài báo.
- **abstract:** Tóm tắt nội dung của bài báo (đây là field chúng ta sẽ sử dụng để phân loại).
- **versions:** Phiên bản của bài báo.
- **update_date:** Ngày cập nhật của bài báo.
- **authors_parsed:** Danh sách các tác giả đã được phân tích và chuẩn hóa.

Chúng ta sẽ sử dụng field **abstract** để làm input cho mô hình phân loại, và field **categories** để làm nhãn (label) cho mô hình.

Để quan sát dữ liệu, chúng ta in ra màn hình 3 bản ghi đầu tiên của bộ dữ liệu như sau:

```

1 # print three first examples
2 for i in range(3):
3     print(f"Example {i+1}:")
4     print(ds['train'][i]['abstract'])
5     print(ds['train'][i]['categories'])
6     print("---" * 20)

```

Dựa vào dữ liệu ta thấy dữ liệu abstract của chúng ta có chứa nhiều kí tự đặc biệt như dấu chấm, dấu phẩy, dấu ngoặc kép, v.v. Ngoài ra, các abstract này cũng có độ dài khác nhau, có thể từ vài câu đến vài đoạn văn. Các nhãn (categories) của các abstract này cũng rất đa dạng, bao gồm nhiều lĩnh vực khác nhau như vật lý, toán học, khoa học máy tính, v.v. Chúng ta in toàn bộ các categories của abstract để xem xét các nhãn này có phù hợp với bài toán phân loại hay không như sau:

```

1 all_categories = ds['train']['categories']
2 print(set(all_categories))

```

Categories của abstract tuân theo format <primary category> <secondary category> ..., ví dụ như hep-ph hay math.CO cs.CG. Chúng ta sẽ sử dụng field `abstract` để làm input cho mô hình phân loại, và phần primary category (category đầu tiên) trong field `categories` để làm nhãn (label) cho mô hình. Để quan sát số lượng primary categories trong bộ dữ liệu, chúng ta sẽ đếm số lượng các nhãn này và in ra như sau:

```
1 all_categories = ds['train']['categories']
2 category_set = set()
3
4 # Collect unique labels
5 for category in all_categories:
6     parts = category.split(' ')
7     for part in parts:
8         topic = part.split('.')[0]
9         category_set.add(topic)
10
11 # Sort the labels and print them
12 sorted_categories= sorted(list(category_set), key=lambda x: x.lower
13                           ())
13 print(f'There are {len(sorted_categories)} unique primary categories
14   in the dataset:')
14 for category in sorted_categories:
15     print(category)
```

Kết quả in ra sẽ là danh sách các primary categories trong bộ dữ liệu như sau:

There are 38 unique primary categories in the dataset:

acc-phys
adap-org
alg-geom
ao-sci
astro-ph
atom-ph
bayes-an
chao-dyn
chem-ph

```

cmp-lg
comp-gas
cond-mat
cs
...

```

Chúng ta sẽ lấy 1000 samples, các samples này có primary là một trong 5 categories sau: `astro-ph`, `cond-mat`, `cs`, `math`, `physics`.

```

1 # load 1000 samples with single label belonging to specific
   categories
2 samples = []
3 CATEGORIES_TO_SELECT = ['astro-ph', 'cond-mat', 'cs', 'math', 'physics']
4 for s in ds['train']:
5     if len(s['categories'].split(' ')) != 1:
6         continue
7
8     cur_category = s['categories'].strip().split('.')[0]
9     if cur_category not in CATEGORIES_TO_SELECT:
10        continue
11
12    samples.append(s)
13
14    if len(samples) >= 1000:
15        break
16 print(f"Number of samples: {len(samples)}") # Number of samples:
                                                1000

```

19.2.3 Tiề̂n xử lý dữ liệu

Tiề̂n xử lý dữ liệu đặc trưng: Nội dung của abstract có chứa nhiều kí tự đặc biệt (newline, trailing space, v.v.), viết hoa, cũng như dấu câu. Do đó, chúng ta cần thực hiện các bước tiền xử lý để chuẩn hóa dữ liệu.

Đối với mỗi sample trong danh sách 1000 samples chúng ta đã chọn ra, chúng ta sẽ thực hiện các bước tiền xử lý như sau:

- Loại bỏ các kí tự \n và khoảng trắng ở đầu và cuối chuỗi.
- Loại bỏ các kí tự đặc biệt (dấu câu, kí tự không phải chữ cái hoặc số).

- Loại bỏ các chữ số.
- Chuyển đổi tất cả các chữ cái thành chữ thường (lowercase).
- Lấy nhãn (label) là primary category (phần đầu tiên) trong field categories.

Các bước tiền xử lý trên tương ứng với đoạn code sau:

```

1 preprocessed_samples = []
2 for s in samples:
3     abstract = s['abstract']
4
5     # Remove \n characters in the middle and leading/trailing spaces
6     abstract = abstract.strip().replace("\n", " ")
7
8     # Remove special characters
9     abstract = re.sub(r'^\w\s', ' ', abstract)
10
11    # Remove digits
12    abstract = re.sub(r'\d+', ' ', abstract)
13
14    # Remove extra spaces
15    abstract = re.sub(r'\s+', ' ', abstract).strip()
16
17    # Convert to lower case
18    abstract = abstract.lower()
19
20    # for the label, we only keep the first part
21    parts = s['categories'].split(' ')
22    category = parts[0].split('.')[0]
23
24    preprocessed_samples.append({
25        "text": abstract,
26        "label": category
27    })

```

Tiếp theo, chúng ta cần tạo hai dictionaries để lưu trữ các primary categories dưới dạng số và ngược lại bằng đoạn code sau:

```

1 label_to_id = {label: i for i, label in enumerate(sorted_labels)}
2 id_to_label = {i: label for i, label in enumerate(sorted_labels)}
3

```

```
4 # Print label to ID mapping
5 print("Label to ID mapping:")
6 for label, id_ in label_to_id.items():
7     print(f"{label} --> {id_}")
```

Kết quả in ra sẽ là danh sách các nhãn và ID tương ứng như sau:

```
Label to ID mapping:
astro-ph -> 0
cond-mat -> 1
cs -> 2
math -> 3
physics -> 4
```

Cuối cùng, chúng ta sẽ chia dữ liệu trên thành 2 tập: tập huấn luyện (train) và tập kiểm tra (test) với tỉ lệ 80% cho tập huấn luyện và 20% cho tập kiểm tra. Chúng ta sẽ sử dụng hàm `train_test_split` từ thư viện `sklearn` để thực hiện việc này.

```
1 X_full = [sample['text'] for sample in preprocessed_samples]
2 y_full = [label_to_id[sample['label']] for sample in
            preprocessed_samples]
3
4 X_train, X_test, y_train, y_test = train_test_split(X_full, y_full,
                                                      test_size=0.2, random_state=42,
                                                      stratify=y_full)
5
6 print(f"Training samples: {len(X_train)}")
7 print(f"Test samples: {len(X_test)}")
```

Số lượng mẫu trong tập huấn luyện và tập kiểm tra sẽ được in ra như sau:

```
Training samples: 800
Test samples: 200
```

19.2.4 Mã hóa văn bản

Văn bản của chúng ta sau khi tiền xử lý sẽ được mã hóa thành các vector số để có thể sử dụng trong mô hình học máy. Có nhiều phương pháp để mã hóa văn bản, nhưng trong bài này, chúng ta sẽ sử dụng từng phương pháp trong ba phương pháp sau và so sánh chúng với nhau: *Bag-of-Words* (BoW), *TF-IDF* (Term Frequency-Inverse Document Frequency), và *Sentence Embeddings*.

Bag-of-Words (BoW)

Bag-of-Words (BoW) là một kỹ thuật biểu diễn văn bản đơn giản và thường được sử dụng. Nó chuyển đổi văn bản thành một vector có độ dài cố định bằng cách đếm số lần xuất hiện của mỗi từ trong văn bản. Phương pháp này bỏ qua ngữ pháp và thứ tự từ nhưng vẫn giữ lại tần suất của các từ.

Ví dụ về cách sử dụng BoW để mã hóa văn bản với class `CountVectorizer` từ thư viện `sklearn` như sau:

```
1 docs = [
2     "I am going to school to study for the final exam.",
3     "The weather is nice today and I feel happy.",
4     "I love programming in Python and exploring new libraries.",
5     "Data science is an exciting field with many opportunities.",
6 ]
7
8 bow = CountVectorizer()
9 vectors = bow.fit_transform(docs)
10
11 for i, vec in enumerate(vectors):
12     print(f"Document {i+1}: {vec.toarray()}")
```

Đoạn code trên khai báo một danh sách các văn bản (`docs`) và sử dụng class `CountVectorizer` để mã hóa chúng thành các vector BoW với hàm `fit_transform`. Sau đó, nó in ra các vector tương ứng với từng văn bản như sau:

Vector BoW khi sử dụng CountVectorizer

```
Document 1: [[1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 2 0
0 0]]
Document 2: [[0 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1
1 0]]
Document 3: [[0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0]]
Document 4: [[0 1 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 1]]
```

Chúng ta thấy được mỗi văn bản được mã hóa thành một vector với kích thước bằng với số lượng từ trong từ điển. Mỗi phần tử trong vector tương ứng với tần suất xuất hiện của từ đó trong văn bản.

TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) là một phương pháp mã hóa văn bản nâng cao hơn BoW, nó không chỉ tính tần suất của từ trong văn bản mà còn xem xét tần suất của từ trong toàn bộ tập dữ liệu. Điều này giúp giảm trọng số của các từ phổ biến và tăng trọng số của các từ hiếm gặp, từ đó cải thiện khả năng phân loại. Ví dụ về cách sử dụng TF-IDF để mã hóa văn bản với class `TfidfVectorizer` từ thư viện `sklearn` như sau:

```
1 docs = [
2     "I am going to school to study for the final exam.",
3     "The weather is nice today and I feel happy.",
4     "I love programming in Python and exploring new libraries.",
5     "Data science is an exciting field with many opportunities.",
6 ]
7
8 vectorizer = TfidfVectorizer()
9 tfidf_vectors = vectorizer.fit_transform(docs)
10
11 for i, vec in enumerate(tfidf_vectors):
12     print(f"TF-IDF for Document {i+1}:")
13     print(vec.toarray())
```

Đoạn code trên khai báo một danh sách các văn bản (docs) và sử dụng

class `TfidfVectorizer` để mã hóa chúng thành các vector TF-IDF với hàm `fit_transform`. Sau đó, nó in ra các vector tương ứng với từng văn bản như sau:

Vector TF-IDF khi sử dụng `TfidfVectorizer`

TF-IDF for Document 1:

```
[[0.29333722 0. 0. 0. 0.29333722 0. 0. 0. 0.29333722 0.29333722  
0.29333722 0. 0. 0. 0. 0. 0. 0. 0. 0.29333722 0. 0.29333722  
0.23127044 0.58667444 0. 0. 0. ]]
```

TF-IDF for Document 2:

```
[[0. 0. 0.30091213 0. 0. 0. 0.38166888 0. 0. 0. 0. 0.38166888 0.  
0.30091213 0. 0. 0. 0. 0.38166888 0. 0. 0. 0. 0. 0.30091213 0.  
0.38166888 0.38166888 0. ]]
```

TF-IDF for Document 3:

```
[[0. 0. 0.2855815 0. 0. 0. 0.36222393 0. 0. 0. 0. 0. 0.36222393 0.  
0.36222393 0.36222393 0. 0.36222393 0. 0. 0.36222393 0.36222393 0.  
0. 0. 0. 0. 0. 0. ]]
```

TF-IDF for Document 4:

```
[[0. 0.34056989 0. 0.34056989 0. 0.34056989 0. 0. 0.34056989 0. 0. 0.  
0. 0.26850921 0. 0. 0.34056989 0. 0. 0.34056989 0. 0. 0. 0.34056989 0.  
0. 0. 0. 0.34056989]]
```

Khác với BoW, các vector TF-IDF có trọng số khác nhau cho từng từ, do đó các vector này biểu diễn từng trọng số với kiểu số thực (float) thay vì số nguyên (int). Điều này giúp mô hình phân loại có thể nhận biết được tầm quan trọng của từng từ trong văn bản.

Sentence Embeddings

Sentence Embeddings là một phương pháp mã hóa văn bản nâng cao hơn, nó chuyển đổi toàn bộ câu hoặc đoạn văn thành một vector có kích thước cố định. Các vector này thường được huấn luyện trên các tập dữ liệu lớn và có thể nắm bắt được ngữ nghĩa của câu, từ đó cải thiện khả năng phân loại.

Chúng ta sẽ implement class `EmbeddingVectorizer` để mã hóa văn bản thành các vector embeddings. Class này sẽ sử dụng mô hình pre-trained từ thư viện `sentence-transformers` để chuyển đổi văn bản thành vector.

Xây dựng EmbeddingVectorizer để mã hóa văn bản

```

1  class EmbeddingVectorizer:
2      def __init__(
3          self,
4          model_name: str = 'intfloat/multilingual-e5-base',
5          normalize: bool = True
6      ):
7          self.model = SentenceTransformer(model_name)
8          self.normalize = normalize
9
10     def _format_inputs(
11         self,
12         texts: List[str],
13         mode: Literal['query', 'passage']
14     ) -> List[str]:
15         if mode not in {"query", "passage"}:
16             raise ValueError("Mode must be either 'query' or 'passage'")
17         return [f"{mode}: {text.strip()}" for text in texts]
18
19     def transform(
20         self,
21         texts: List[str],
22         mode: Literal['query', 'passage'] = 'query'
23     ) -> List[List[float]]:
24         if mode == 'raw':
25             inputs = texts
26         else:
27             inputs = self._format_inputs(texts, mode)
28
29         embeddings = self.model.encode(inputs, normalize_embeddings=
```

```

30             self.normalize)
31     return embeddings.tolist()
32
32     def transform_numpy(
33         self,
34         texts: List[str],
35         mode: Literal['query', 'passage'] = 'query'
36     ) -> np.ndarray:
37         return np.array(self.transform(texts, mode=mode))

```

Class `EmbeddingVectorizer` được xây dựng với các methods sau:

- `__init__`: Khởi tạo mô hình SentenceTransformer với tên mô hình và tùy chọn chuẩn hóa.
- `_format_inputs`: Định dạng đầu vào cho mô hình, thêm tiền tố "query" hoặc "passage" vào văn bản.
- `transform`: Chuyển đổi danh sách văn bản thành các vector embeddings.
- `transform_numpy`: Chuyển đổi danh sách văn bản thành các vector embeddings dưới dạng numpy array.

Tương tự hai phương pháp trên, chúng ta sẽ sử dụng class `EmbeddingVectorizer` để mã hóa văn bản trong bộ dữ liệu của mình. Đoạn code sau sẽ thực hiện việc này:

```

1 docs = [
2     "I am going to school to study for the final exam.",
3     "The weather is nice today and I feel happy.",
4     "I love programming in Python and exploring new libraries.",
5     "Data science is an exciting field with many opportunities.",
6 ]
7
8 vectorizer = EmbeddingVectorizer()
9 embeddings = vectorizer.transform(docs)
10
11 for i, emb in enumerate(embeddings):
12     print(f"Embedding for Document {i+1}:")
13     print(emb[:3]) # Print first 10 dimensions for brevity

```

Đoạn code trên sẽ mã hóa các văn bản trong danh sách `docs` thành các vector embeddings và in ra 10 chiều đầu tiên của mỗi vector như sau:

Vector Embeddings khi sử dụng EmbeddingVectorizer	
Embedding for Document 1:	-0.014805682934820652, 0.031276583671569824, -0.01615864224731922
Embedding for Document 2:	0.011917386204004288, 0.03327357769012451, -0.025739021599292755
Embedding for Document 3:	0.012662884779274464, 0.03936118632555008, -0.024181174114346504
Embedding for Document 4:	0.0063935937359929085, 0.04922199621796608, -0.028402965515851974

Các vectors này tương tự như các vectors TF-IDF, nhưng chúng có kích thước cố định và có thể nắm bắt được ngữ nghĩa của câu. Bên cạnh đó, có rất ít từ trong các vectors này có giá trị bằng 0, điều này cho thấy các vectors embeddings có thể biểu diễn tốt hơn ngữ nghĩa của văn bản so với các phương pháp mã hóa khác.

Khai báo và khởi tạo các vectorizers

Chúng ta sẽ khai báo và khởi tạo các vectorizers và huấn luyện chúng trên tập dữ liệu của chúng ta cho từng phương pháp mã hóa văn bản đã đề cập ở trên. Đoạn code sau sẽ thực hiện việc này:

```

1 bow_vectorizer = CountVectorizer()
2 X_train_bow = # Your code here
3 X_test_bow = bow_vectorizer.transform(X_test)
4
5 tfidf_vectorizer = TfidfVectorizer()
6 X_train_tfidf = # Your code here
7 X_test_tfidf = tfidf_vectorizer.transform(X_test)
8
9 embedding_vectorizer = EmbeddingVectorizer()
10 X_train_embeddings = # Your code here
11 X_test_embeddings = embedding_vectorizer.transform(X_test)
12

```

```

13 # convert all to numpy arrays for consistency
14 X_train_bow, X_test_bow = np.array(X_train_bow.toarray()), np.array(
15                                     X_test_bow.toarray())
16 X_train_tfidf, X_test_tfidf = np.array(X_train_tfidf.toarray()), np.
17                                     array(X_test_tfidf.toarray())
18 X_train_embeddings, X_test_embeddings = np.array(X_train_embeddings)
19                                     , np.array(X_test_embeddings)
20
21 # Print shapes of the transformed datasets
22 print(f"Shape of X_train_bow: {X_train_bow.shape}\n")
23 print(f"Shape of X_test_bow: {X_test_bow.shape}\n")
24 print(f"Shape of X_train_tfidf: {X_train_tfidf.shape}\n")
25 print(f"Shape of X_test_tfidf: {X_test_tfidf.shape}\n")
26 print(f"Shape of X_train_embeddings: {X_train_embeddings.shape}\n")
27 print(f"Shape of X_test_embeddings: {X_test_embeddings.shape}\n")

```

Kết quả in ra màn hình lần lượt sẽ là kích thước của các tập dữ liệu đã được mã hóa:

Kích thước của các tập dữ liệu đã được mã hóa

Shape of X_train_bow: (800, 10373)
 Shape of X_test_bow: (200, 10373)

Shape of X_train_tfidf: (800, 10373)
 Shape of X_test_tfidf: (200, 10373)

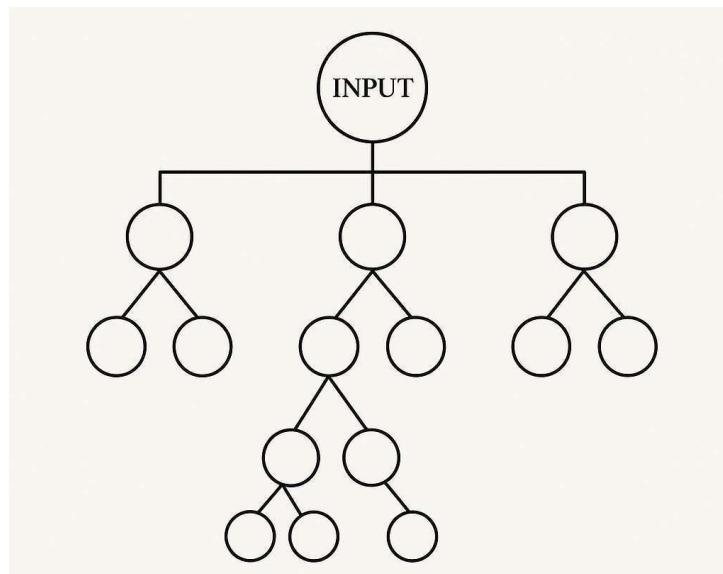
Shape of X_train_embeddings: (800, 768)
 Shape of X_test_embeddings: (200, 768)

Chúng ta thấy rằng các vector BoW và TF-IDF có kích thước bằng nhau, tương ứng với số lượng từ trong từ điển (10373 từ), trong khi các vector embeddings có kích thước cố định là 768, cho thấy chúng được mã hóa thành các vector có kích thước nhỏ hơn nhiều so với BoW và TF-IDF.

19.2.5 Huấn luyện và đánh giá mô hình phân loại

Random Forest

Random Forest là một thuật toán học máy mạnh mẽ, nó cải thiện độ chính xác dự đoán bằng cách kết hợp đầu ra của nhiều decision trees độc lập từ các mẫu ngẫu nhiên của dữ liệu huấn luyện thông qua quá trình bootstrapping. Khi đưa ra dự đoán, Random Forest sẽ tổng hợp kết quả của các decision trees này - bằng majority vote cho các bài toán phân loại hoặc bằng cách lấy trung bình cho các bài toán hồi quy.



Hình 19.3: Random Forest Algorithm

Tương tự như các mô hình trước, chúng ta sẽ thiết kế một hàm `train_and_test_random_forest` để huấn luyện mô hình Random Forest và in ra các kết quả phân loại như sau:

Hàm `train_and_test_random_forest` để huấn luyện mô hình Random Forest

```

1 def train_and_test_random_forest(X_train, y_train, X_test, y_test,
2                                 n_estimators: int = 100):
3     rf = RandomForestClassifier(n_estimators=n_estimators,
4
  
```

```

1                                         random_state=42)
2     rf.fit(X_train, y_train)
3
4     # Predict on the test set
5     y_pred = rf.predict(X_test)
6
7     # Calculate accuracy and classification report
8     accuracy = accuracy_score(y_test, y_pred)
9     report = classification_report(y_test, y_pred, target_names=
10                                    sorted_labels, output_dict=True)
11
12    return y_pred, accuracy, report

```

Sau đó, chúng ta sẽ chạy hàm `train_and_test_random_forest` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình Random Forest cho các phương pháp mã hóa

```

1 rf_bow_labels, rf_bow_accuracy, rf_bow_report =
2                                         train_and_test_random_forest(
3                                         X_train_bow, y_train, X_test_bow,
4                                         y_test)
5 rf_tfidf_labels, rf_tfidf_accuracy, rf_tfidf_report =
6                                         train_and_test_random_forest(
7                                         X_train_tfidf, y_train,
8                                         X_test_tfidf, y_test)
9 rf_embeddings_labels, rf_embeddings_accuracy, rf_embeddings_report =
10                                         train_and_test_random_forest(
11                                         X_train_embeddings, y_train,
12                                         X_test_embeddings, y_test)
13
14 # Print Random Forest results
15 print("Accuracies for Random Forest:")
16 print(f"Bag of Words: {rf_bow_accuracy:.4f}")
17 print(f"TF-IDF: {rf_tfidf_accuracy:.4f}")
18 print(f"Embeddings: {rf_embeddings_accuracy:.4f}")

```

Và đây là kết quả độ chính xác của mô hình Random Forest cho từng phương pháp mã hóa:

		Random Forest Confusion Matrix (Bag of Words)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	84 (96.55%)	1 (1.15%)	0 (0.00%)	2 (2.30%)	0 (0.00%)
	cond-mat	11 (23.91%)	34 (73.91%)	0 (0.00%)	1 (2.17%)	0 (0.00%)
	cs	1 (20.00%)	0 (0.00%)	0 (0.00%)	4 (80.00%)	0 (0.00%)
	math	2 (4.65%)	0 (0.00%)	0 (0.00%)	41 (95.35%)	0 (0.00%)
	physics	7 (36.84%)	7 (36.84%)	0 (0.00%)	5 (26.32%)	0 (0.00%)

(a) Confusion Matrix Random Forest với BoW

Độ chính xác của mô hình Random Forest cho từng phương pháp mã hóa

Accuracies for Random Forest:

Bag of Words: 0.7950

TF-IDF: 0.7750

Embeddings: 0.8550

Trong đó, mô hình Random Forest với phương pháp mã hóa embeddings đạt độ chính xác cao nhất là 85.5%, trong khi phương pháp BoW và TF-IDF đạt độ chính xác lần lượt là 79.5% và 77.5%. Điều này cho thấy mô hình Random Forest có thể hoạt động tốt với các phương pháp mã hóa khác nhau, nhưng vẫn có xu hướng hoạt động tốt hơn với phương pháp mã hóa embeddings (giống với các mô hình đã thí nghiệm phía trên).

Cuối cùng, chúng ta có confusion matrices cho từng phương pháp mã hóa trong Hình 19.4.

		Random Forest Confusion Matrix (TF-IDF)				
		Predicted Label				
True Label	astro-ph	84 (96.55%)	0 (0.00%)	0 (0.00%)	3 (3.45%)	0 (0.00%)
	cond-mat	13 (28.26%)	30 (65.22%)	0 (0.00%)	3 (6.52%)	0 (0.00%)
	cs	0 (0.00%)	1 (20.00%)	0 (0.00%)	4 (80.00%)	0 (0.00%)
	math	2 (4.65%)	0 (0.00%)	0 (0.00%)	41 (95.35%)	0 (0.00%)
	physics	7 (36.84%)	7 (36.84%)	0 (0.00%)	5 (26.32%)	0 (0.00%)
	astro-ph	cond-mat	cs	math	physics	

(b) Confusion Matrix Random Forest với TF-IDF

		Random Forest Confusion Matrix (Embeddings)				
		Predicted Label				
True Label	astro-ph	87 (100.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	cond-mat	3 (6.52%)	43 (93.48%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
	cs	2 (40.00%)	0 (0.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	1 (2.33%)	1 (2.33%)	0 (0.00%)	41 (95.35%)	0 (0.00%)
	physics	5 (26.32%)	11 (57.89%)	0 (0.00%)	3 (15.79%)	0 (0.00%)
	astro-ph	cond-mat	cs	math	physics	

(c) Confusion Matrix Random Forest với Embeddings

Hình 19.4: Confusion Matrix cho từng phương pháp mã hóa với mô hình Random Forest

AdaBoost

AdaBoost, hay Adaptive Boosting, là một thuật toán học máy kết hợp nhiều mô hình học đơn giản và "yếu" để tạo ra một mô hình "mạnh" có độ chính xác cao. Nó hoạt động bằng cách huấn luyện lặp đi lặp lại các mô hình yếu này, với mỗi mô hình tiếp theo tập trung nhiều hơn vào các ví dụ bị phân loại sai

từ các mô hình trước đó bằng cách tăng trọng số của chúng. Dự đoán cuối cùng là trung bình có trọng số của các dự đoán từ tất cả các mô hình yếu. Chúng ta xây dựng hàm huấn luyện và kiểm thử mô hình AdaBoost trên bộ dữ liệu của chúng ta như sau:

Hàm train_and_test_adaboost để huấn luyện mô hình AdaBoost

```

1 def train_and_test_adaboost(X_train, y_train, X_test, y_test,
2                             n_estimators: int = 50):
3     ada = AdaBoostClassifier(n_estimators=n_estimators, random_state
4                               =42)
5     ada.fit(X_train, y_train)
6
7     # Predict on the test set
8     y_pred = ada.predict(X_test)
9
10    # Calculate accuracy and classification report
11    accuracy = accuracy_score(y_test, y_pred)
12    report = classification_report(y_test, y_pred, target_names=
13                                    sorted_labels, output_dict=True)
14
15    return y_pred, accuracy, report

```

Sau đó, chúng ta sẽ chạy hàm `train_and_test_adaboost` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình AdaBoost cho các phương pháp mã hóa

```

1 ada_bow_labels, ada_bow_accuracy, ada_bow_report =
2     train_and_test_adaboost(
3         X_train_bow, y_train, X_test_bow,
4         y_test)
5 ada_tfidf_labels, ada_tfidf_accuracy, ada_tfidf_report =
6     train_and_test_adaboost(
7         X_train_tfidf, y_train,
8         X_test_tfidf, y_test)
9 ada_embeddings_labels, ada_embeddings_accuracy,
10    ada_embeddings_report =
11        train_and_test_adaboost(
12            X_train_embeddings, y_train,
13            X_test_embeddings, y_test)

```

```
4 # Print AdaBoost results
5 print("Accuracies for AdaBoost:")
6 print(f"Bag of Words: {ada_bow_accuracy:.4f}")
7 print(f"TF-IDF: {ada_tfidf_accuracy:.4f}")
8 print(f"Embeddings: {ada_embeddings_accuracy:.4f}")
```

Và đây là kết quả độ chính xác của mô hình AdaBoost cho từng phương pháp mã hóa:

Độ chính xác của mô hình AdaBoost cho từng phương pháp mã hóa

Accuracies for AdaBoost:
Bag of Words: 0.6000
TF-IDF: 0.5700
Embeddings: 0.6200

Chúng ta có thể thấy kết quả đánh giá mô hình AdaBoost không được như mong đợi, với độ chính xác chỉ đạt khoảng 60% cho phương pháp BoW và embeddings, và thấp hơn một chút với TF-IDF là 57%. Kết quả này thấp hơn các mô hình machine learning trước đó như Decision Tree, Naive Bayes và SVM. Để cải thiện, chúng ta có thể tối ưu các hyperparameters của mô hình AdaBoost hoặc thử các thuật toán boosting khác như Gradient Boosting hoặc XGBoost.

Cuối cùng, chúng ta có confusion matrices cho từng phương pháp mã hóa trong Hình 19.5.

		AdaBoost Confusion Matrix (Bag of Words)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	61 (70.11%)	21 (24.14%)	0 (0.00%)	4 (4.60%)	1 (1.15%)
	cond-mat	13 (28.26%)	27 (58.70%)	0 (0.00%)	5 (10.87%)	1 (2.17%)
	cs	2 (40.00%)	0 (0.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	3 (6.98%)	9 (20.93%)	0 (0.00%)	31 (72.09%)	0 (0.00%)
	physics	6 (31.58%)	6 (31.58%)	0 (0.00%)	6 (31.58%)	1 (5.26%)

(a) Confusion Matrix AdaBoost với BoW

		AdaBoost Confusion Matrix (TF-IDF)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	77 (88.51%)	3 (3.45%)	0 (0.00%)	7 (8.05%)	0 (0.00%)
	cond-mat	29 (63.04%)	10 (21.74%)	0 (0.00%)	7 (15.22%)	0 (0.00%)
	cs	1 (20.00%)	1 (20.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	13 (30.23%)	3 (6.98%)	0 (0.00%)	27 (62.79%)	0 (0.00%)
	physics	9 (47.37%)	4 (21.05%)	0 (0.00%)	6 (31.58%)	0 (0.00%)

(b) Confusion Matrix AdaBoost với TF-IDF

		AdaBoost Confusion Matrix (Embeddings)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	70 (80.46%)	1 (1.15%)	0 (0.00%)	6 (6.90%)	10 (11.49%)
	cond-mat	0 (0.00%)	30 (65.22%)	0 (0.00%)	9 (19.57%)	7 (15.22%)
	cs	0 (0.00%)	1 (20.00%)	0 (0.00%)	2 (40.00%)	2 (40.00%)
	math	5 (11.63%)	7 (16.28%)	0 (0.00%)	21 (48.84%)	10 (23.26%)
	physics	4 (21.05%)	8 (42.11%)	0 (0.00%)	4 (21.05%)	3 (15.79%)

(c) Confusion Matrix AdaBoost với Embeddings

Hình 19.5: Confusion Matrix cho từng phương pháp mã hóa với mô hình AdaBoost

Gradient Boosting

Gradient boosting là một kỹ thuật học máy tập hợp, xây dựng các cây quyết định (hoặc các mô hình yếu khác) theo trình tự, với mỗi cây mới sửa lỗi của các cây trước đó để tạo thành một mô hình dự đoán mạnh mẽ tổng thể. Nó hoạt động bằng cách lặp đi lặp lại việc tối thiểu hóa một hàm mất mát, sử dụng gradient để hướng dẫn việc huấn luyện mỗi mô hình con mới. Việc sử dụng gradient boosting có thể tồn kém về mặt tính toán và dễ bị overfitting, đòi hỏi việc điều chỉnh hyperparameter cẩn thận.

Chúng ta xây dựng hàm huấn luyện và kiểm thử mô hình Gradient Boosting trên bộ dữ liệu của chúng ta như sau:

Hàm train_and_test_gradient_boosting để huấn luyện mô hình Gradient Boosting

```

1 def train_and_test_gradient_boosting(X_train, y_train, X_test,
2                                     y_test, n_estimators: int = 100):
3     gb = GradientBoostingClassifier(n_estimators=n_estimators,
4                                     random_state=42)
5     gb.fit(X_train, y_train)
6
7     # Predict on the test set

```

```

6     y_pred = gb.predict(X_test)
7
8     # Calculate accuracy and classification report
9     accuracy = accuracy_score(y_test, y_pred)
10    report = classification_report(y_test, y_pred, target_names=
11                                    sorted_labels, output_dict=True)
12
13    return y_pred, accuracy, report

```

Sau đó, chúng ta sẽ chạy hàm `train_and_test_gradient_boosting` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình Gradient Boosting cho các phương pháp mã hóa

```

1 gb_bow_labels, gb_bow_accuracy, gb_bow_report =
2     train_and_test_gradient_boosting(
3         X_train_bow, y_train, X_test_bow,
4         y_test)
5 gb_tfidf_labels, gb_tfidf_accuracy, gb_tfidf_report =
6     train_and_test_gradient_boosting(
7         X_train_tfidf, y_train,
8         X_test_tfidf, y_test)
9 gb_embeddings_labels, gb_embeddings_accuracy, gb_embeddings_report =
10        train_and_test_gradient_boosting(
11            X_train_embeddings, y_train,
12            X_test_embeddings, y_test)
13
14 # Print Gradient Boosting results
15 print("Accuracies for Gradient Boosting:")
16 print(f"Bag of Words: {gb_bow_accuracy:.4f}")
17 print(f"TF-IDF: {gb_tfidf_accuracy:.4f}")
18 print(f"Embeddings: {gb_embeddings_accuracy:.4f}")

```

Và đây là kết quả độ chính xác của mô hình Gradient Boosting cho từng phương pháp mã hóa:

		Gradient Boosting Confusion Matrix (Bag of Words)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	79 (90.80%)	3 (3.45%)	0 (0.00%)	3 (3.45%)	2 (2.30%)
	cond-mat	4 (8.70%)	38 (82.61%)	0 (0.00%)	3 (6.52%)	1 (2.17%)
	cs	1 (20.00%)	1 (20.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	2 (4.65%)	2 (4.65%)	0 (0.00%)	39 (90.70%)	0 (0.00%)
	physics	2 (10.53%)	6 (31.58%)	0 (0.00%)	6 (31.58%)	5 (26.32%)

(a) Confusion Matrix Gradient Boosting với BoW

Độ chính xác của mô hình Gradient Boosting cho từng phương pháp mã hóa

Accuracies for Gradient Boosting:

Bag of Words: 0.8050

TF-IDF: 0.8000

Embeddings: 0.8650

Kết quả đánh giá mô hình Gradient Boosting cho thấy độ chính xác đạt khoảng 80.5% với phương pháp BoW, 80% với TF-IDF và cao nhất là 86.5% với embeddings. Kết quả này khá tương đồng với các mô hình machine learning trước đó như Decision Tree và SVM, đặc biệt là khi sử dụng embeddings. Để cải thiện hơn nữa, chúng ta có thể tối ưu các hyperparameters của mô hình Gradient Boosting hoặc thử các thuật toán boosting khác như XGBoost hoặc LightGBM.

Cuối cùng, chúng ta có confusion matrices cho từng phương pháp mã hóa trong Hình 19.6.

		Gradient Boosting Confusion Matrix (TF-IDF)				
		Predicted Label				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	83 (95.40%)	2 (2.30%)	0 (0.00%)	2 (2.30%)	0 (0.00%)
	cond-mat	6 (13.04%)	34 (73.91%)	0 (0.00%)	3 (6.52%)	3 (6.52%)
	cs	1 (20.00%)	1 (20.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	3 (6.98%)	2 (4.65%)	0 (0.00%)	38 (88.37%)	0 (0.00%)
	physics	3 (15.79%)	7 (36.84%)	0 (0.00%)	4 (21.05%)	5 (26.32%)

(b) Confusion Matrix Gradient Boosting với TF-IDF

		Gradient Boosting Confusion Matrix (Embeddings)				
		Predicted Label				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	85 (97.70%)	0 (0.00%)	0 (0.00%)	2 (2.30%)	0 (0.00%)
	cond-mat	0 (0.00%)	44 (95.65%)	0 (0.00%)	0 (0.00%)	2 (4.35%)
	cs	1 (20.00%)	1 (20.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	1 (2.33%)	2 (4.65%)	0 (0.00%)	40 (93.02%)	0 (0.00%)
	physics	6 (31.58%)	7 (36.84%)	0 (0.00%)	2 (10.53%)	4 (21.05%)

(c) Confusion Matrix Gradient Boosting với Embeddings

Hình 19.6: Confusion Matrix cho từng phương pháp mã hóa với mô hình Gradient Boosting

XGBoost

XGBoost là một thư viện học máy mã nguồn mở, được sử dụng rộng rãi vì cách triển khai tối ưu hóa và mở rộng của thuật toán gradient boosting cho các tác vụ như phân loại, hồi quy và xếp hạng. XGBoost nổi tiếng nhờ tốc

độ, độ chính xác, cũng như khả năng xử lý các tập dữ liệu lớn thông qua xử lý song song và phân tán.

Chúng ta xây dựng hàm huấn luyện và kiểm thử mô hình XGBoost như sau:

Hàm train_and_test_xgboost để huấn luyện mô hình XGBoost

```

1 def train_and_test_xgboost(X_train, y_train, X_test, y_test,
                           n_estimators: int = 100):
2     xgb = XGBClassifier(
3         n_estimators=n_estimators, use_label_encoder=False,
4         eval_metric='mlogloss',
5         random_state=42
6     )
7     xgb.fit(X_train, y_train)
8
9     # Predict on the test set
10    y_pred = xgb.predict(X_test)
11
12    # Calculate accuracy and classification report
13    accuracy = accuracy_score(y_test, y_pred)
14    report = classification_report(y_test, y_pred, target_names=
15                                    sorted_labels, output_dict=True)
16
17    return y_pred, accuracy, report

```

Tiếp theo, chúng ta sẽ chạy hàm `train_and_test_xgboost` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình XGBoost cho các phương pháp mã hóa

```

1 xgb_bow_labels, xgb_bow_accuracy, xgb_bow_report =
2     train_and_test_xgboost(X_train_bow,
3                             y_train, X_test_bow, y_test)
4 xgb_tfidf_labels, xgb_tfidf_accuracy, xgb_tfidf_report =
5     train_and_test_xgboost(
6         X_train_tfidf, y_train,
7         X_test_tfidf, y_test)
8 xgb_embeddings_labels, xgb_embeddings_accuracy,
9     xgb_embeddings_report =
10    train_and_test_xgboost(
11        X_train_embeddings, y_train,
12        X_test_embeddings, y_test)

```

```
4                                     X_test_embeddings, y_test)
5 # Print XGBoost results
6 print("Accuracies for XGBoost:")
7 print(f"Bag of Words: {xgb_bow_accuracy:.4f}")
8 print(f"TF-IDF: {xgb_tfidf_accuracy:.4f}")
9 print(f"Embeddings: {xgb_embeddings_accuracy:.4f}")
```

Và đây là kết quả độ chính xác của mô hình XGBoost cho từng phương pháp mã hóa:

Độ chính xác của mô hình XGBoost cho từng phương pháp mã hóa

Accuracies for XGBoost:
Bag of Words: 0.7850
TF-IDF: 0.7650
Embeddings: 0.8750

Kết quả đánh giá mô hình XGBoost cho thấy độ chính xác đạt khoảng 78.5% với phương pháp BoW, 76.5% với TF-IDF và cao nhất là 87.5% với embeddings. Kết quả này cho thấy XGBoost hoạt động rất tốt với phương pháp mã hóa embeddings, đạt độ chính xác trong top những mô hình hoạt động tốt nhất trong các mô hình đã thử nghiệm. Tuy nhiên, kết quả với BoW và TF-IDF thấp hơn một chút so với các mô hình khác như SVM và Gradient Boosting.

Cuối cùng, chúng ta có confusion matrices cho từng phương pháp mã hóa trong Hình 19.7.

XGBoost Confusion Matrix (Bag of Words)					
True Label	astro-ph	81 (93.10%)	3 (3.45%)	0 (0.00%)	2 (2.30%)
	cond-mat	8 (17.39%)	35 (76.09%)	0 (0.00%)	3 (6.52%)
	cs	0 (0.00%)	2 (40.00%)	0 (0.00%)	2 (40.00%)
	math	1 (2.33%)	0 (0.00%)	0 (0.00%)	41 (95.35%)
	physics	2 (10.53%)	10 (52.63%)	1 (5.26%)	6 (31.58%)
	Predicted Label	astro-ph	cond-mat	cs	math

(a) Confusion Matrix XGBoost với BoW

XGBoost Confusion Matrix (TF-IDF)					
True Label	astro-ph	83 (95.40%)	3 (3.45%)	0 (0.00%)	1 (1.15%)
	cond-mat	9 (19.57%)	32 (69.57%)	0 (0.00%)	3 (6.52%)
	cs	0 (0.00%)	2 (40.00%)	0 (0.00%)	2 (40.00%)
	math	2 (4.65%)	0 (0.00%)	0 (0.00%)	36 (83.72%)
	physics	3 (15.79%)	9 (47.37%)	0 (0.00%)	5 (26.32%)
	Predicted Label	astro-ph	cond-mat	cs	math

(b) Confusion Matrix XGBoost với TF-IDF

		XGBoost Confusion Matrix (Embeddings)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	85 (97.70%)	0 (0.00%)	0 (0.00%)	1 (1.15%)	1 (1.15%)
	cond-mat	0 (0.00%)	44 (95.65%)	0 (0.00%)	1 (2.17%)	1 (2.17%)
	cs	1 (20.00%)	1 (20.00%)	0 (0.00%)	3 (60.00%)	0 (0.00%)
	math	1 (2.33%)	1 (2.33%)	0 (0.00%)	41 (95.35%)	0 (0.00%)
	physics	4 (21.05%)	8 (42.11%)	0 (0.00%)	2 (10.53%)	5 (26.32%)

(c) Confusion Matrix XGBoost với Embeddings

Hình 19.7: Confusion Matrix cho từng phương pháp mã hóa với mô hình XGBoost

LightGBM

LightGBM (Light Gradient Boosting Machine) được xây dựng dựa trên thuật toán Gradient Boosting Decision Tree (GBDT). Đây là một framework học tăng cường theo gradient, tận dụng các thuật toán học dựa trên cây, đặc biệt là decision tree đã được tối ưu hóa.

Đặc điểm và tối ưu hóa nổi bật của LightGBM được liệt kê như sau:

- Phát triển cây theo lá (Leaf-wise, Best-first): Khác với nhiều phương pháp triển khai GBDT khác thường mở rộng cây theo từng tầng (level-wise), LightGBM mở rộng cây theo lá. Điều này có nghĩa là nó sẽ chọn lá có độ giảm tổn thất (delta loss) lớn nhất để chia, giúp mô hình hội tụ nhanh hơn và tiềm năng đạt độ chính xác cao hơn.
- Thuật toán dựa trên histogram: LightGBM sử dụng thuật toán dựa trên histogram để tìm điểm chia tối ưu, từ đó rút ngắn đáng kể thời gian huấn luyện, đặc biệt với các tập dữ liệu lớn.
- Tiết kiệm bộ nhớ: Thiết kế hướng đến hiệu quả sử dụng bộ nhớ, phù hợp để xử lý dữ liệu ở quy mô lớn.
- Hỗ trợ học song song và phân tán: LightGBM hỗ trợ nhiều chiến lược

huấn luyện song song và phân tán, giúp mở rộng khả năng xử lý cho các bài toán Big Data.

Đầu tiên, chúng ta xây dựng hàm huấn luyện và kiểm thử mô hình LightGBM như sau:

Hàm train_and_test_lightgbm để huấn luyện mô hình LightGBM

```

1 def train_and_test_lightgbm(X_train, y_train, X_test, y_test,
2                             n_estimators: int = 100):
3     lgbm = lgb.LGBMClassifier(boosting_type='goss', n_estimators=
4                               n_estimators, random_state=42)
5     lgbm.fit(X_train, y_train)
6
7     # Predict on the test set
8     y_pred = lgbm.predict(X_test)
9
10    # Calculate accuracy and classification report
11    accuracy = accuracy_score(y_test, y_pred)
12    report = classification_report(y_test, y_pred, target_names=
13                                    sorted_labels, output_dict=True)
14
15    return y_pred, accuracy, report

```

Tiếp theo, chúng ta sẽ chạy hàm `train_and_test_lightgbm` cho từng phương pháp mã hóa và in ra kết quả như sau:

Huấn luyện mô hình LightGBM cho các phương pháp mã hóa

```

1 lgbm_bow_labels, lgbm_bow_accuracy, lgbm_bow_report =
2     train_and_test_lightgbm(
3         X_train_bow, y_train, X_test_bow,
4         y_test)
5 lgbm_tfidf_labels, lgbm_tfidf_accuracy, lgbm_tfidf_report =
6     train_and_test_lightgbm(
7         X_train_tfidf, y_train,
8         X_test_tfidf, y_test)
9 lgbm_embeddings_labels, lgbm_embeddings_accuracy,
10    lgbm_embeddings_report =
11    train_and_test_lightgbm(
12        X_train_embeddings, y_train,
13        X_test_embeddings, y_test)

```

```
4 # Print LightGBM results
5 print("Accuracies for LightGBM:")
6 print(f"Bag of Words: {lgbm_bow_accuracy:.4f}")
7 print(f"TF-IDF: {lgbm_tfidf_accuracy:.4f}")
8 print(f"Embeddings: {lgbm_embeddings_accuracy:.4f}")
```

Và đây là kết quả độ chính xác của mô hình LightGBM cho từng phương pháp mã hóa:

Độ chính xác của mô hình LightGBM cho từng phương pháp mã hóa

Accuracies for LightGBM:
Bag of Words: 0.7500
TF-IDF: 0.7750
Embeddings: 0.8800

Kết quả in ra màn hình cho thấy độ chính xác của mô hình LightGBM đạt khoảng 75% với phương pháp BoW, 77.5% với TF-IDF và cao nhất là 88% với embeddings. Tương tự với hầu hết các mô hình khác, mô hình LightGBM hoạt động tốt nhất với phương pháp mã hóa embeddings, đạt độ chính xác trong top những mô hình hoạt động tốt nhất trong các mô hình đã thử nghiệm. Kết quả với BoW và TF-IDF thấp hơn một chút so với các mô hình khác như SVM và Gradient Boosting.

Cuối cùng, chúng ta có confusion matrices cho từng phương pháp mã hóa trong Hình 19.8.

		LightGBM Confusion Matrix (Bag of Words)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	82 (94.25%)	3 (3.45%)	0 (0.00%)	1 (1.15%)	1 (1.15%)
	cond-mat	9 (19.57%)	32 (69.57%)	0 (0.00%)	3 (6.52%)	2 (4.35%)
	cs	0 (0.00%)	2 (40.00%)	0 (0.00%)	1 (20.00%)	2 (40.00%)
	math	2 (4.65%)	1 (2.33%)	0 (0.00%)	36 (83.72%)	4 (9.30%)
	physics	4 (21.05%)	9 (47.37%)	0 (0.00%)	6 (31.58%)	0 (0.00%)

(a) Confusion Matrix LightGBM với BoW

		LightGBM Confusion Matrix (TF-IDF)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	82 (94.25%)	4 (4.60%)	0 (0.00%)	1 (1.15%)	0 (0.00%)
	cond-mat	8 (17.39%)	34 (73.91%)	0 (0.00%)	2 (4.35%)	2 (4.35%)
	cs	0 (0.00%)	1 (20.00%)	0 (0.00%)	2 (40.00%)	2 (40.00%)
	math	2 (4.65%)	0 (0.00%)	0 (0.00%)	36 (83.72%)	5 (11.63%)
	physics	3 (15.79%)	9 (47.37%)	0 (0.00%)	4 (21.05%)	3 (15.79%)

(b) Confusion Matrix LightGBM với TF-IDF

		LightGBM Confusion Matrix (Embeddings)				
		astro-ph	cond-mat	cs	math	physics
True Label	astro-ph	86 (98.85%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	1 (1.15%)
	cond-mat	1 (2.17%)	44 (95.65%)	0 (0.00%)	0 (0.00%)	1 (2.17%)
	cs	1 (20.00%)	0 (0.00%)	0 (0.00%)	3 (60.00%)	1 (20.00%)
	math	0 (0.00%)	1 (2.33%)	0 (0.00%)	41 (95.35%)	1 (2.33%)
	physics	3 (15.79%)	8 (42.11%)	0 (0.00%)	3 (15.79%)	5 (26.32%)

(c) Confusion Matrix LightGBM với Embeddings

Hình 19.8: Confusion Matrix cho từng phương pháp mã hóa với mô hình LightGBM

19.3 Câu hỏi trắc nghiệm

1. Mục tiêu chính của bài toán Text Classification là gì?
 - (a) Dịch văn bản sang ngôn ngữ khác
 - (b) Phân loại đoạn văn bản vào các nhóm/nhãn dựa trên nội dung
 - (c) Tóm tắt văn bản thành một câu
 - (d) Tạo văn bản mới từ dữ liệu huấn luyện
2. Bước tiền xử lý nào **không** được nêu trong tài liệu?
 - (a) Chuyển chữ thường (lowercase)
 - (b) Loại bỏ ký tự đặc biệt và chữ số
 - (c) Loại bỏ ký tự xuống dòng và khoảng trắng dư
 - (d) Stemming/Lemmatization
3. Phát biểu nào đúng về Bag-of-Words (BoW)?
 - (a) Mã hoá dựa trên mô hình ngôn ngữ tiền huấn luyện
 - (b) Bảo toàn thứ tự từ trong câu
 - (c) Biểu diễn bằng đếm tần suất từ, bỏ qua trật tự
 - (d) Luôn cho vector có kích thước 768
4. Vai trò của thành phần IDF trong TF-IDF là gì?
 - (a) Tăng trọng số cho các từ xuất hiện thường xuyên trong mọi tài liệu
 - (b) Giảm trọng số của các từ phổ biến trong toàn bộ tập dữ liệu
 - (c) Chuẩn hoá chiều dài tài liệu về cùng độ dài
 - (d) Loại bỏ hoàn toàn các từ dừng (stopwords)
5. Điểm khác biệt chính giữa BoW/TF-IDF và Sentence Embeddings là gì?
 - (a) Embeddings cho vector dày (dense), kích thước cố định, nắm bắt ngữ nghĩa tốt hơn

- (b) Embeddings tạo vector rời rạc (sparse) có kích thước bằng kích thước từ vựng
- (c) Embeddings yêu cầu thủ công chọn đặc trưng
- (d) Embeddings chỉ hoạt động với văn bản tiếng Anh
6. Khi dùng `train_test_split(..., stratify=y)` với argument `stratify`, ta đang đảm bảo điều gì?
- (a) Kích thước vector đều ra bằng nhau
- (b) Tỉ lệ nhãn giữa tập train/test được bảo toàn
- (c) Huấn luyện nhanh hơn
- (d) Tự động chuẩn hoá dữ liệu
7. Phát biểu nào đúng về Random Forest trong phân loại?
- (a) Mô hình chỉ chứa một cây rất sâu duy nhất để giảm bias
- (b) Tập hợp nhiều cây quyết định huấn luyện trên mẫu bootstrap, bỏ phiếu đa số
- (c) Huấn luyện tuần tự để giảm sai số của cây trước
- (d) Chỉ phù hợp với dữ liệu ảnh
8. Đặc trưng cốt lõi của AdaBoost là gì?
- (a) Chọn ngẫu nhiên đặc trưng tại mỗi nút chia
- (b) Huấn luyện song song nhiều mô hình yếu độc lập
- (c) Tăng trọng số các mẫu bị phân loại sai ở vòng trước để học tuần tự
- (d) Dùng tìm kiếm theo histogram cho điểm chia
9. Ý tưởng chính của Gradient Boosting là gì?
- (a) Lấy trung bình dự đoán của nhiều cây độc lập để giảm phương sai
- (b) Xây nhiều mô hình tuần tự, mỗi mô hình mới học theo hướng gradient của hàm loss
- (c) Nén từ vựng để giảm chiều

- (d) Tối ưu trực tiếp độ chính xác thay vì dùng hàm loss
10. LightGBM khác các triển khai GBDT thông thường ở điểm nào?
- (a) Phát triển cây theo lá (leaf-wise, best-first) và dùng thuật toán histogram để tìm điểm chia
 - (b) Luôn bắt buộc độ sâu cây cố định bằng 1
 - (c) Không hỗ trợ huấn luyện song song
 - (d) Chỉ hoạt động với dữ liệu ảnh độ phân giải cao

1. Hint: Các file code gợi ý và dữ liệu được lưu trong thư mục có thể được tải [tại đây](#).

2. Solution: Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. Kiến thức cần cho project:

Để hoàn thành và hiểu rõ phần project một cách hiệu quả nhất, các học viên cần nắm rõ các kiến thức được trang bị

4. Đề xuất phương pháp cải tiến project:

Project tiếp cận bài toán phân tích chủ đề văn bản (topic modeling) theo hướng sử dụng các phương pháp biểu diễn vector như Bag of Words (BoW), TF-IDF, S-BERT kết hợp với các thuật toán KNN, K-Means clustering và Decision Tree. Để tiếp tục cải tiến, tối ưu hệ thống; nhóm tác biên soạn gợi ý một số phương pháp như sau:

- **Tiền xử lý nâng cao:** Sử dụng các kỹ thuật tiền xử lý chuyên sâu hơn như loại bỏ từ ngữ ít xuất hiện (rare words), lemmatization thay vì stemming, phát hiện và xử lý stopwords theo ngữ cảnh, và phát hiện cụm từ (phrase detection) để tăng chất lượng vector biểu diễn.
- **Thử nghiệm nhiều phương pháp biểu diễn:**
 - So sánh hiệu quả giữa BoW, TF-IDF và mô hình embedding hiện đại như S-BERT.
 - Kết hợp nhiều phương pháp biểu diễn (feature fusion) để tận dụng ưu điểm của từng phương pháp.
- **Tối ưu các thuật toán cây/ensemble:** bằng cách tinh chỉnh các siêu tham số mô hình
- **Điển giải mô hình (Model interpretability):**
 - Feature importance (split/gain/Permutation), SHAP cho cả cục bộ và toàn cục.
 - Partial Dependence / ICE để kiểm tra hiệu ứng đơn biến; *monotonic constraints* (XGB/LGBM) để áp điều kiện nghiệp vụ.

- **Tăng cường dữ liệu (Data Augmentation):** Áp dụng các phương pháp như thay thế từ đồng nghĩa, paraphrasing hoặc back-translation để làm phong phú tập dữ liệu huấn luyện.

Trên đây là một số gợi ý, giúp học viên có thể cải tiến thêm hệ thống để đạt hiểu quả tốt hơn về cả tốc độ xử lý và độ chính xác.

5. Rubric:

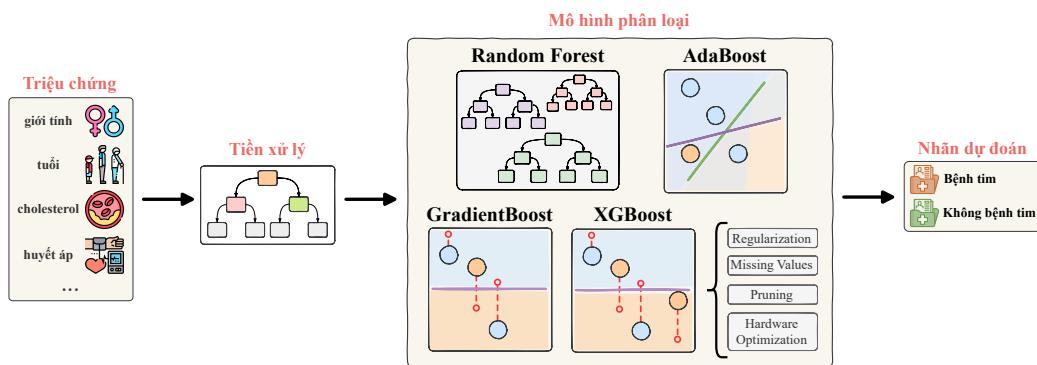
Mục	Kiến Thức	Đánh Giá
II.	<ul style="list-style-type: none"> - Lý thuyết về bài toán Topic Modeling trong Natural Language Processing. - Lý thuyết về các phương pháp biểu diễn văn bản: Bag of Words (BoW), TF-IDF, S-BERT và các thuật toán K-Means, KNN, Decision Tree. 	<ul style="list-style-type: none"> - Hiểu được khái niệm topic modeling và các phương pháp biểu diễn văn bản. - Có khả năng sử dụng Python để triển khai các thuật toán K-Means, KNN và Decision Tree cho bài toán topic modeling.
III.	<ul style="list-style-type: none"> - Ứng dụng K-Means clustering để phân cụm chủ đề và Decision Tree để gán nhãn chủ đề. - So sánh hiệu quả các phương pháp biểu diễn văn bản (BoW, TF-IDF, S-BERT) trong bài toán topic modeling. 	<ul style="list-style-type: none"> - Thực hiện phân cụm chủ đề và đánh giá mô hình bằng các chỉ số như Silhouette Score. - Đánh giá, so sánh và chọn phương pháp biểu diễn văn bản phù hợp cho dữ liệu thực tế.

Chương 20

Project 2: Phân loại khả năng mắc bệnh tim dựa vào các triệu chứng (dùng các giải thuật đã học)

20.1 Giới thiệu

Ở phần trước, chúng ta đã đặt những “viên gạch nền” cho bài toán dự đoán bệnh tim: hiểu cấu trúc dữ liệu Cleveland, chuẩn hóa quy trình tiền xử lý, áp dụng kỹ thuật tạo đặc trưng và xây dựng các mô hình Machine Learning (ML) cơ bản như Naive Bayes, K-Nearest Neighbors, Decision Tree, thậm chí thử nghiệm một biến thể ensemble cơ bản. Kết quả cho thấy các mô hình truyền thống, khi được huấn luyện bằng dữ liệu đã xử lý cẩn thận, có thể đạt hiệu năng đáng khích lệ và giúp ta đọc vị bài toán khá hiệu quả.



Hình 20.1: Pipeline Project Cleveland Heart Disease Diagnosis.

Bước sang phần 2 này, chúng ta tiếp tục nâng cấp cho hệ thống bằng những kỹ thuật ensemble nâng cao với những phương pháp vốn rất mạnh trên dữ liệu bảng:

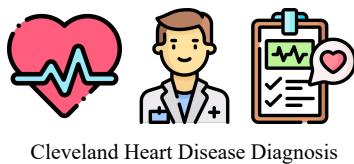
- Random Forest đại diện cho họ Bagging, kết hợp nhiều cây độc lập để giảm phương sai và tăng ổn định.

- AdaBoost và Gradient Boosting đại diện cho họ Boosting, nơi các mô hình liên tiếp “sửa sai” lân nhau.
- XGBoost hiện thực Boosting một cách tối ưu hơn với nhiều cải tiến về regularization và hiệu năng tính toán.

Trong toàn bộ dự án này, mọi điều kiện thực nghiệm được giữ nguyên, nhằm tái sử dụng các tập dữ liệu train/val/test và hai cấu hình dữ liệu gồm bản gốc và bản đã xử lý đặc trưng (feature engineering). Chúng ta cũng sử dụng bộ chỉ số quen thuộc như Accuracy, Precision, Recall, F1. Sự thay đổi chính sẽ tập trung vào việc xây dựng một bộ dữ liệu mới bằng mô hình Decision Tree; và huấn luyện các bộ dữ liệu này trên các mô hình Ensemble nâng cao. Chương tiếp theo sẽ trình bày cách triển khai thực nghiệm trong dự án này.

20.2 Cài đặt chương trình

20.2.1 Bộ dữ liệu



Cleveland Heart Disease Diagnosis

Hình 20.2: Bộ dữ liệu dự đoán bệnh tim Cleveland Heart Disease Diagnosis.

Để đảm bảo tính nhất quán và so sánh hiệu quả giữa các mô hình, dự án này tiếp tục sử dụng bộ dữ liệu **Cleveland Heart Disease** [?] đã được giới thiệu chi tiết ở dự án trước. Bộ dữ liệu bao gồm thông tin y tế của **303 bệnh nhân**, với 13 đặc trưng đầu vào (như tuổi, giới tính, cholesterol, v.v.) và một biến mục tiêu để chẩn đoán bệnh nhân có mắc bệnh tim hay không. Bảng dưới đây tóm tắt lại ý nghĩa của từng đặc trưng.

Bảng 20.1: Một số mẫu dữ liệu từ bộ Cleveland Heart Disease.

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0	0
67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0	1
67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0	1
37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0	0
41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0	0

Bảng mô tả đặc trưng bộ dữ liệu Cleveland Heart Disease

Đặc trưng	Mô tả và giá trị mã hóa
age	Tuổi của bệnh nhân (năm).
sex	Giới tính (1 = nam, 0 = nữ).
cp	Loại đau ngực. (1 = đau thắt ngực điển hình, 2 = không điển hình, 3 = đau không do tim, 4 = không triệu chứng)
trestbps	Huyết áp tâm thu lúc nghỉ (mmHg).
chol	Nồng độ cholesterol huyết thanh (mg/dL).
fbs	Đường huyết lúc đói > 120 mg/dL. (1 = đúng, 0 = sai)
restecg	Điện tâm đồ lúc nghỉ. (0 = bình thường, 1 = bất thường ST-T, 2 = phì đại thất trái)
thalach	Nhịp tim tối đa đạt được (lần/phút).
exang	Đau ngực khi gắng sức. (1 = có, 0 = không)
oldpeak	Mức độ trầm ST do gắng sức so với nghỉ.
slope	Độ dốc đoạn ST. (1 = dốc lên, 2 = bằng phẳng, 3 = dốc xuống)
ca	Số mạch máu chính được nhuộm màu (0–3).
thal	Thalassemia. (3 = bình thường, 6 = tổn thương cố định, 7 = tổn thương có thể đảo ngược)
num	Nhận mục tiêu. (0 = không bệnh, 1–4 = có bệnh)

Các giá trị rác được mã hóa trong ngoặc để phục vụ huấn luyện mô hình.

20.2.2 Cải thiện đặc trưng

Trong phần này, chúng ta thực hiện kỹ thuật xử lý đặc trưng (Feature Engineering - FE) tương tự như dự án trước để nhận được 2 bộ dữ liệu gồm: bộ gốc và bộ FE (cả 2 tập đều có 13 cột/đặc trưng). Sau đó, sử dụng mô hình Decision Tree (DT) để tìm các đặc trưng quan trọng và tạo bộ dữ liệu mới với $K = 10$ đặc trưng quan trọng nhất. Cuối cùng, ta sẽ có tổng cộng 4 bộ dữ liệu: bộ gốc, bộ FE, bộ gốc áp dụng DT, bộ FE áp dụng DT.

Công việc trước tiên là tải các thư viện và thiết lập giá trị ngẫu nhiên, tạo nền tảng vững chắc cho quá trình xử lý.

Tải các thư viện cần thiết

```
1 import os
2 import json
3 import random
4 import numpy as np
5 import pandas as pd
6 import matplotlib.pyplot as plt
7
8 from pathlib import Path
9 from sklearn.pipeline import Pipeline
10 from sklearn.impute import SimpleImputer
11 from sklearn.compose import ColumnTransformer
12 from sklearn.tree import DecisionTreeClassifier
13 from sklearn.model_selection import train_test_split
14 from sklearn.base import BaseEstimator, TransformerMixin
15 from sklearn.feature_selection import mutual_info_classif
16 from sklearn.preprocessing import OneHotEncoder, StandardScaler,
17                                         MinMaxScaler
18
19 os.environ['PYTHONHASHSEED'] = '42'
20 np.random.seed(42)
21 random.seed(42)
```

Sau đó, tải tập dữ liệu Cleveland được lưu trữ trên Google Drive.

Tải bộ dữ liệu gốc (chưa được xử lý)

```

1 # https://drive.google.com/file/d/16HPyuXWXPPtt5g3xvS_kR_wXAfjpR1Ju/
   view?usp=sharing
2 !gdown 16HPyuXWXPPtt5g3xvS_kR_wXAfjpR1Ju

```

Tiếp đến, tập dữ liệu được đọc và cấu trúc hóa với các cột được đặt tên, dữ liệu số được chuyển đổi, các giá trị thiếu được xác định và cột mục tiêu dự đoán được mã hóa nhị phân để sẵn sàng cho phân tích.

Đọc và cấu trúc hóa tập dữ liệu

```

1 DATA_PATH = 'cleveland.csv'
2 COLUMNS = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
3             'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target']
4
5 numeric_cols = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
6 categorical_cols = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca',
7                      'thal']
8
9 K_features = 10
10 raw = pd.read_csv(DATA_PATH, header=None)
11 raw.columns = COLUMNS
12
13 for c in ['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'ca', 'thal']:
14     raw[c] = pd.to_numeric(raw[c], errors='coerce')
15
16 raw['target'] = (raw['target'] > 0).astype(int)
17 print('Shape:', raw.shape)
18 display(raw.head())
19 display(raw.isna().sum())

```

Tiến hành chia dữ liệu thành các tập train, validation (val) và test.

Chia tập dữ liệu

```

1 TARGET = 'target'
2 raw_feature_cols = [c for c in raw.columns if c != TARGET]
3 X_all = raw[raw_feature_cols]
4 y_all = raw[TARGET]

```

```

5
6 X_train, X_temp, y_train, y_temp = train_test_split(
7     X_all, y_all, test_size=0.2, stratify=y_all, random_state=42
8 )
9 X_val, X_test, y_val, y_test = train_test_split(
10    X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=42
11 )

```

Một quy trình pipeline được xây dựng để tự động hóa việc tiền xử lý: xử lý các giá trị thiếu và chuẩn hóa dữ liệu, để đảm bảo tính nhất quán giữa các tập. Các tập dữ liệu đã được xử lý này sau đó được lưu thành các tệp CSV riêng biệt, tương ứng với train/val/test.

Xây dựng bộ dữ liệu gốc (đã được xử lý)

```

1 cat_proc = Pipeline(steps=[
2     ('imputer', SimpleImputer(strategy='most_frequent')),
3     ('scaler', MinMaxScaler())
4 ])
5 num_proc = Pipeline(steps=[
6     ('imputer', SimpleImputer(strategy='median')),
7     ('scaler', StandardScaler())
8 ])
9
10 preprocess = ColumnTransformer([
11     ('num', num_proc, numeric_cols),
12     ('cat', cat_proc, categorical_cols),
13 ])
14 raw_pipeline = Pipeline([
15     ('preprocess', preprocess),
16 ])
17
18 X_raw_train = raw_pipeline.fit_transform(X_train, y_train)
19 X_raw_val = raw_pipeline.transform(X_val)
20 X_raw_test = raw_pipeline.transform(X_test)
21
22 preprocessed_feature_names = []
23 for name, transformer, columns in preprocess.transformers_:
24     if hasattr(transformer, 'get_feature_names_out'):
25         preprocessed_feature_names.extend(transformer.
                                         get_feature_names_out(columns))

```

```

26     else:
27         preprocessed_feature_names.extend(columns)
28
29 X_raw_train_df = pd.DataFrame(
30     X_raw_train, columns=preprocessed_feature_names, index=X_train.
31                                         index)
32 X_raw_val_df = pd.DataFrame(
33     X_raw_val, columns=preprocessed_feature_names, index=X_val.index
34                                         )
35 X_raw_test_df = pd.DataFrame(
36     X_raw_test, columns=preprocessed_feature_names, index=X_test.
37                                         index)
38
39 out_dir = Path('splits'); out_dir.mkdir(parents=True, exist_ok=True)
40 pd.concat([X_raw_train_df, y_train.rename(TARGET)],
41           axis=1).to_csv(out_dir / 'raw_train.csv', index=False)
42 pd.concat([X_raw_val_df, y_val.rename(TARGET)],
43           axis=1).to_csv(out_dir / 'raw_val.csv', index=False)
44 pd.concat([X_raw_test_df, y_test.rename(TARGET)],
45           axis=1).to_csv(out_dir / 'raw_test.csv', index=False)
46
47 display(X_raw_train_df)

```

Sử dụng mô hình Decision Tree để chọn lọc các đặc trưng quan trọng nhất. Độ quan trọng của mỗi đặc trưng được tính toán và sắp xếp, từ đó chọn ra các đặc trưng hàng đầu để tạo ra một bộ dữ liệu mới, tối ưu hóa cho mô hình.

Tạo bộ dữ liệu mới với Decision Tree

```

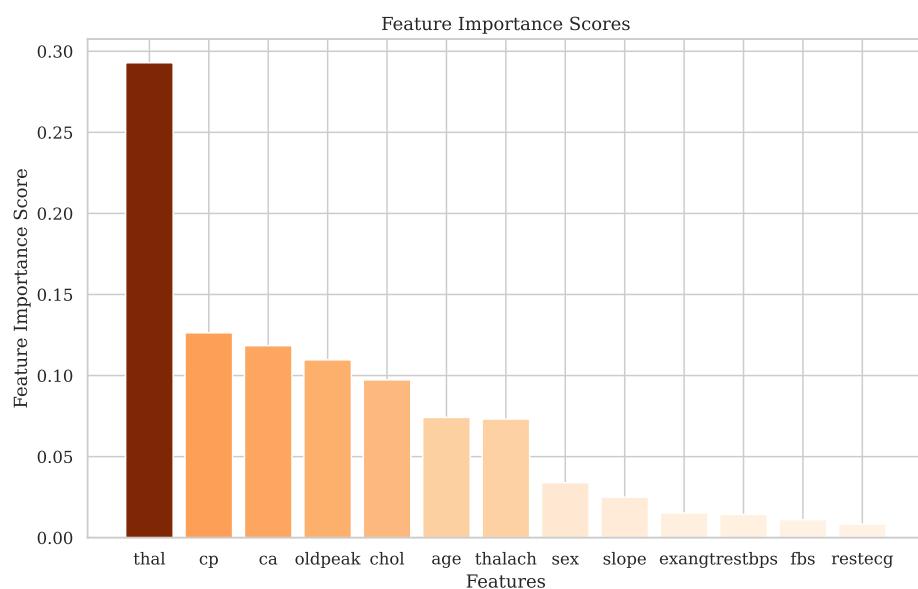
1 dt_feature_selection_pipeline = Pipeline([
2     ('preprocess', preprocess),
3     ('decision_tree', DecisionTreeClassifier(random_state=42))
4 ])
5
6 dt_feature_selection_pipeline.fit(X_train, y_train)
7 feature_importance_series = pd.Series(
8     dt_feature_selection_pipeline.named_steps['decision_tree'].
9         feature_importances_,
10        index=preprocessed_feature_names
11    )
12 sorted_feature_importances = feature_importance_series.sort_values(

```

```

12         ascending=False)
13 display(sorted_feature_importances)
14 selected_features = sorted_feature_importances.head(K_features).
15                                     index.tolist()
16 print(f'Top {K_features} selected features: {selected_features}')
17 X_dt_train = X_raw_train_df[selected_features]
18 X_dt_val = X_raw_val_df[selected_features]
19 X_dt_test = X_raw_test_df[selected_features]
20 display(X_dt_train.head())
21
22 pd.concat([X_dt_train, y_train.rename(TARGET)],
23            axis=1).to_csv(out_dir / 'dt_train.csv', index=False)
24 pd.concat([X_dt_val, y_val.rename(TARGET)],
25            axis=1).to_csv(out_dir / 'dt_val.csv', index=False)
26 pd.concat([X_dt_test, y_test.rename(TARGET)],
27            axis=1).to_csv(out_dir / 'dt_test.csv', index=False)

```



Hình 20.3: Mức độ ảnh hưởng của các đặc trưng dựa vào Decision Tree trên bộ dt_train.

Tiếp tục quá trình bằng cách tạo các đặc trưng mới (Feature Engineering), như tỷ lệ cholesterol theo tuổi, để làm giàu dữ liệu. Các đặc trưng này được tích hợp vào một pipeline riêng, để áp dụng lên bộ dữ liệu gốc (chưa được xử lý).

```
1 def add_new_features_func(df):
2     df = df.copy()
3     if {'chol', 'age'} <= set(df.columns):
4         df['chol_per_age'] = df['chol']/df['age']
5     if {'trestbps', 'age'} <= set(df.columns):
6         df['bps_per_age'] = df['trestbps']/df['age']
7     if {'thalach', 'age'} <= set(df.columns):
8         df['hr_ratio'] = df['thalach']/df['age']
9     if 'age' in df.columns:
10        df['age_bin'] = pd.cut(
11            df['age'], bins=5, labels=False
12            ).astype('category')
13    return df
14
15 class AddNewFeaturesTransformer(BaseEstimator, TransformerMixin):
16     def __init__(self):
17         pass
18
19     def fit(self, X, y=None):
20         self.columns_ = X.columns
21         self.new_features_ = []
22         if {'chol', 'age'} <= set(X.columns):
23             self.new_features_.append('chol_per_age')
24         if {'trestbps', 'age'} <= set(X.columns):
25             self.new_features_.append('bps_per_age')
26         if {'thalach', 'age'} <= set(X.columns):
27             self.new_features_.append('hr_ratio')
28         if 'age' in X.columns:
29             self.new_features_.append('age_bin')
30     return self
31
32     def transform(self, X):
33         return add_new_features_func(X)
34
35     def get_feature_names_out(self, input_features=None):
36         return list(self.columns_) + self.new_features_
37
38
```

```

39 gen_num = ['chol_per_age', 'bps_per_age', 'hr_ratio']
40 gen_cat = ['age_bin']
41 all_nums = [c for c in numeric_cols] + gen_num
42 all_cats = [c for c in categorical_cols] + gen_cat
43
44 num_proc = Pipeline([('imp', SimpleImputer(strategy='median')),
45                      ('sc', StandardScaler())])
46 cat_proc = Pipeline([('imp', SimpleImputer(strategy='most_frequent')
47                               ),
48                      ('ohe', OneHotEncoder(handle_unknown='ignore',
49                                   sparse_output=False))])
50
51 pre = ColumnTransformer([
52     ('num', num_proc, all_nums),
53     ('cat', cat_proc, all_cats),
54 ], verbose_feature_names_out=False).set_output(transform='pandas')
55 fe_pre = Pipeline([
56     ('add', AddNewFeaturesTransformer()),
57     ('pre', pre),
58 ]).set_output(transform='pandas')
59
60 Xt_tr = fe_pre.fit_transform(X_train, y_train)
61 Xt_va = fe_pre.transform(X_val)
62 Xt_te = fe_pre.transform(X_test)
63
64 nz_cols = Xt_tr.columns[Xt_tr.nunique(dropna=False) > 1]
65 Xt_tr = Xt_tr[nz_cols]
66 Xt_va = Xt_va[nz_cols]
67 Xt_te = Xt_te[nz_cols]

```

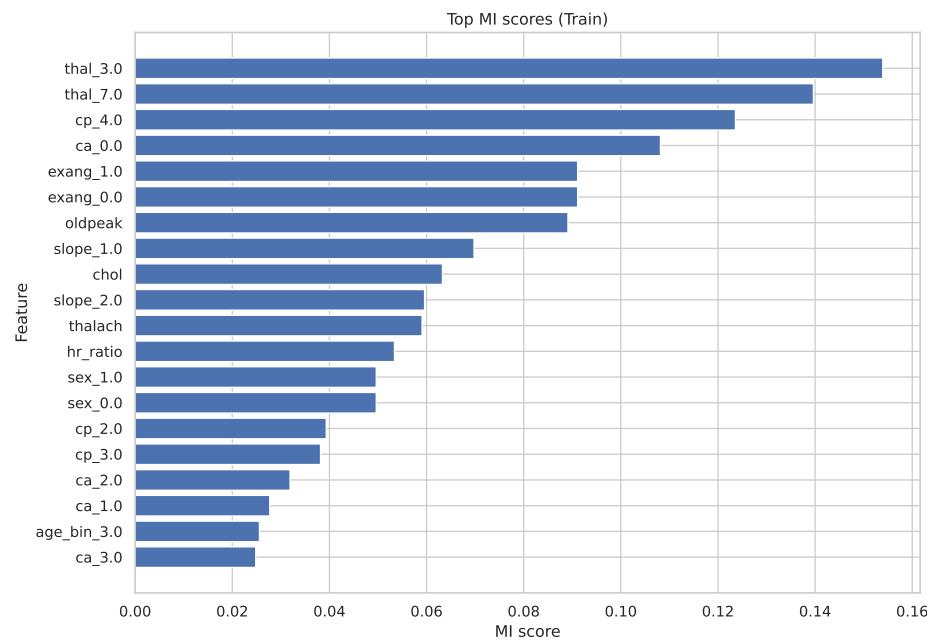
Áp dụng phương pháp Mutual Information (như dự án trước) để đánh giá và chọn lọc các đặc trưng có mối liên hệ cao nhất với biến mục tiêu. Sau đó, lưu lại bộ dữ liệu FE mới được tạo này.

```

1 ohe = fe_pre.named_steps['pre'].named_transformers_['cat'].named_steps['ohe']
2 cat_names = list(ohe.get_feature_names_out(all_cats))
3 is_discrete = np.array(
4     [c in cat_names for c in Xt_tr.columns],
5     dtype=bool
6 )
7 mi = mutual_info_classif(Xt_tr.values, y_train.values,

```

```
8             discrete_features=is_discrete,
9             random_state=42)
10            mi_series = pd.Series(
11                mi, index=Xt_tr.columns).sort_values(ascending=False)
12
13 N = min(20, len(mi_series))
14 topN = mi_series.head(N).iloc[::-1]
15 plt.figure(figsize=(10, max(6, 0.35*N)))
16 plt.barh(topN.index, topN.values)
17 plt.title('Top MI scores (Train)')
18 plt.xlabel('MI score')
19 plt.ylabel('Feature')
20 plt.tight_layout()
21 plt.savefig('top_mi_scores.pdf', bbox_inches='tight')
22 plt.show()
23
24 K = raw.columns.drop('target').shape[0]
25 topk_cols = list(mi_series.head(K).index)
26 fe_tr = Xt_tr[topk_cols].assign(target=y_train.values)
27 fe_va = Xt_va[topk_cols].assign(target=y_val.values)
28 fe_te = Xt_te[topk_cols].assign(target=y_test.values)
29
30 out = Path('splits'); out.mkdir(parents=True, exist_ok=True)
31 fe_tr.to_csv(out/'fe_train.csv', index=False)
32 fe_va.to_csv(out/'fe_val.csv', index=False)
33 fe_te.to_csv(out/'fe_test.csv', index=False)
34
35 display(pd.Series(
36     topk_cols, name='fe_topk_features'
37 ).reset_index(drop=True))
```



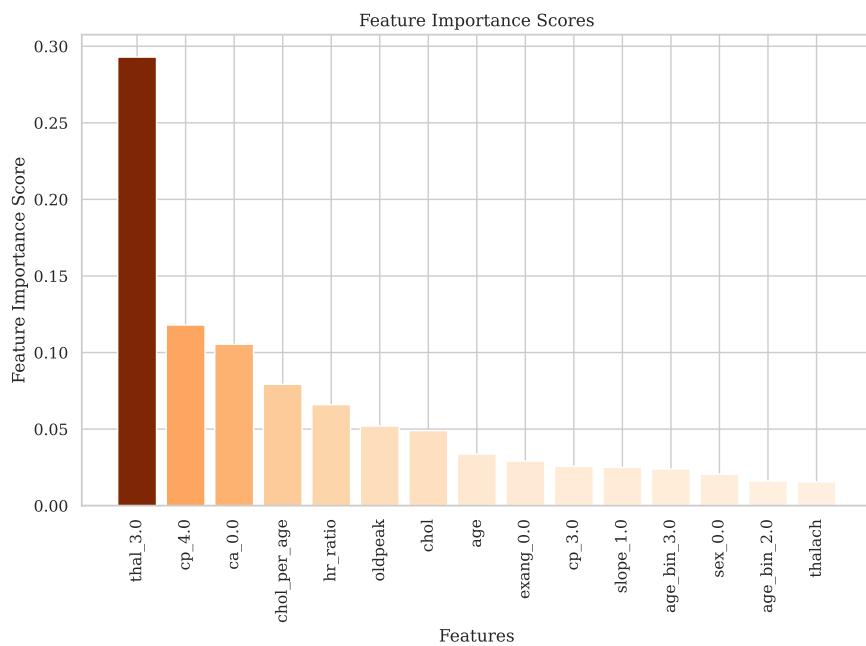
Hình 20.4: Mức độ ảnh hưởng của các đặc trưng theo phương pháp Mutual Information trên bộ **train** của tập gốc (đã được xử lý).

Cuối cùng, một lần nữa sử dụng mô hình Decision Tree để chọn lọc đặc trưng từ bộ dữ liệu đã qua kỹ thuật tạo đặc trưng (FE), xác định các đặc trưng có tác động lớn nhất. Từ đó, tạo ra các tập dữ liệu cuối cùng với các đặc trưng này.

Tạo tập dữ liệu mới trên tập FE với Decision Tree

```
1 dt_fe_feature_selection_pipeline = Pipeline([
2     ('preprocess', fe_pre),
3     ('decision_tree', DecisionTreeClassifier(random_state=42))
4 ])
5
6 dt_fe_feature_selection_pipeline.fit(X_train, y_train)
7 pipeline_feature_names = dt_fe_feature_selection_pipeline.
8                             named_steps['preprocess'].
9                             get_feature_names_out()
feature_importance_series = pd.Series(
    dt_fe_feature_selection_pipeline.named_steps['decision_tree'].
        feature importances ,
```

```
10     index=pipeline_feature_names
11 )
12 sorted_feature_importances = feature_importance_series.sort_values(
13                             ascending=False)
14 selected_features = sorted_feature_importances.head(K_features).
15                             index.tolist()
16 print(f'Top {K_features} selected features: {selected_features}')
17 X_fe_dt_train = Xt_tr[selected_features]
18 X_fe_dt_val = Xt_va[selected_features]
19 X_fe_dt_test = Xt_te[selected_features]
20 display(X_fe_dt_train.head())
21
22 pd.concat([X_fe_dt_train, y_train.rename(TARGET)],
23             axis=1).to_csv(out_dir / 'fe_dt_train.csv', index=False)
24 pd.concat([X_fe_dt_val, y_val.rename(TARGET)],
25             axis=1).to_csv(out_dir / 'fe_dt_val.csv', index=False)
26 pd.concat([X_fe_dt_test, y_test.rename(TARGET)],
27             axis=1).to_csv(out_dir / 'fe_dt_test.csv', index=False)
```



Hình 20.5: Mức độ ảnh hưởng của các đặc trưng dựa vào Decision Tree trên bộ **train** của tập FE.

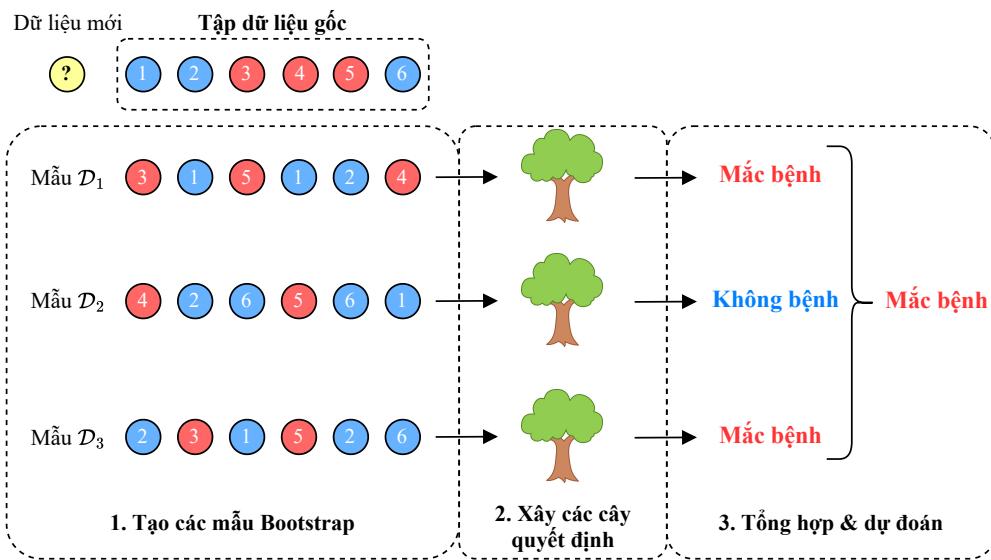
Sử dụng mã bên dưới để nén các tập dữ liệu thành `dataset.zip` để tải xuống từ Google Colab.

Nén các tập dữ liệu để tải xuống

```
1 !zip -r dataset.zip splits
```

20.2.3 Random Forest

Giới thiệu



Hình 20.6: Minh họa cách hoạt động của Random Forest.

Random Forest là một thuật toán học máy có giám sát thuộc nhóm ensemble, hoạt động bằng cách xây dựng một “rừng” gồm nhiều Decision Tree trong quá trình huấn luyện. Mục tiêu chính của thuật toán là cải thiện độ chính xác và khắc phục nhược điểm lớn của một cây quyết định đơn lẻ: xu hướng bị overfitting (quá khớp) và có phương sai cao.

Để đảm bảo các cây trong rừng là khác biệt và không bị tương quan cao với nhau, Random Forest áp dụng hai kỹ thuật ngẫu nhiên hóa cốt lõi:

- (i) **Bagging (Bootstrap Aggregating):** Mỗi cây được huấn luyện trên một tập dữ liệu con khác nhau, được lấy ngẫu nhiên có hoàn lại từ tập dữ liệu huấn luyện ban đầu.
- (ii) **Feature Randomness:** Tại mỗi bước phân nhánh của một cây, thay vì xem xét toàn bộ các đặc trưng, thuật toán chỉ chọn một tập con ngẫu nhiên của các đặc trưng để tìm ra điểm chia tối ưu.

Khi dự đoán trên dữ liệu mới, kết quả cuối cùng được tổng hợp từ tất cả các cây. Cụ thể, mô hình sẽ lấy lớp có nhiều phiếu bầu nhất cho bài toán **phân loại** (cơ chế *majority voting*), hoặc tính giá trị trung bình của các dự đoán cho bài toán **hồi quy**. Bằng cách tổng hợp từ nhiều cây đa dạng, Random Forest tạo ra một mô hình tổng thể vừa mạnh mẽ, chính xác, vừa có khả năng khai quát hóa tốt trên dữ liệu chưa từng thấy.

Triển khai

Lưu ý: Các khối code được đánh dấu “ * ” nghĩa là được sử dụng nhiều lần mà không cần sửa đổi giữa các mô hình trong dự án này.

Bắt đầu bằng cách nhập các thư viện cần thiết, thiết lập một số ngẫu nhiên chung để đảm bảo tính tái lập của kết quả.

*Tải các thư viện cần thiết

```
1 import os
2 import random
3 import warnings
4 import numpy as np
5 import pandas as pd
6 import seaborn as sns
7 import matplotlib.pyplot as plt
8
9 from xgboost import XGBClassifier
10 from sklearn.tree import DecisionTreeClassifier
11 from sklearn.ensemble import AdaBoostClassifier
12 from sklearn.ensemble import RandomForestClassifier
13 from sklearn.ensemble import GradientBoostingClassifier
14 from sklearn.metrics import accuracy_score, classification_report
15 from sklearn.model_selection import StratifiedKFold, cross_val_score
16
17 warnings.filterwarnings('ignore')
18
19 SEED = 42
20 os.environ['PYTHONHASHSEED'] = str(SEED)
21 np.random.seed(SEED)
22 random.seed(SEED)
23 print(f'Seed: {SEED}')
```

Sau đó, tải xuống bộ dữ liệu được lưu trữ trên Google Drive.

***Tải các bộ dữ liệu**

```

1 # https://drive.google.com/drive/folders/
      1cMoqIDEgGYDVzv8B7cKp3csxujQ40Fp7?
      usp=drive_link
2 !gdown --folder 1cMoqIDEgGYDVzv8B7cKp3csxujQ40Fp7

```

Tiếp theo, một hàm được tạo để đọc các tệp .csv, hiển thị năm dòng đầu tiên, đếm số lượng các nhãn mục tiêu và in kích thước của dữ liệu.

***Hàm đọc dữ liệu trong file .csv**

```

1 def read_csv(file_path):
2     df = pd.read_csv(file_path)
3     display(df.head())
4
5     X = df.drop('target', axis=1)
6     y = df['target']
7     display(y.value_counts())
8
9     print('Shape df: ', df.shape)
10    print('Shape X: ', X.shape)
11    print('Shape y: ', y.shape)
12
13    return X, y

```

Sử dụng hàm vừa định nghĩa, tiến hành đọc và tải các tập dữ liệu huấn luyện, bao gồm các phiên bản dữ liệu gốc, dữ liệu đã qua xử lý kỹ thuật xử lý đặc trưng (FE) và các tập dữ liệu được tạo bằng cách chọn lọc đặc trưng bằng Decision Tree (DT) trên 2 bộ trên.

***Đọc các tập dữ liệu**

```

1 # Original Dataset
2 X_train, y_train = read_csv('dataset_v3/raw_train.csv')
3 X_val, y_val = read_csv('dataset_v3/raw_val.csv')
4 X_test, y_test = read_csv('dataset_v3/raw_test.csv')
5
6 # FE Dataset
7 X_fe_train, y_fe_train = read_csv('dataset_v3/fe_train.csv')

```

```

8 X_fe_val, y_fe_val = read_csv('dataset_v3/fe_val.csv')
9 X_fe_test, y_fe_test = read_csv('dataset_v3/fe_test.csv')
10
11 # Original + DT Dataset
12 X_dt_train, y_dt_train = read_csv('dataset_v3/dt_train.csv')
13 X_dt_val, y_dt_val = read_csv('dataset_v3/dt_val.csv')
14 X_dt_test, y_dt_test = read_csv('dataset_v3/dt_test.csv')
15
16 # FE + DT Dataset
17 X_fe_dt_train, y_fe_dt_train = read_csv('dataset_v3/fe_dt_train.csv'
18                                     )
18 X_fe_dt_val, y_fe_dt_val = read_csv('dataset_v3/fe_dt_val.csv')
19 X_fe_dt_test, y_fe_dt_test = read_csv('dataset_v3/fe_dt_test.csv')

```

Sau đó, định nghĩa một hàm để tìm số lượng cây tối ưu cho mô hình Random Forest bằng phương pháp Stratified K-Fold Cross-Validation. Quá trình này giúp đánh giá hiệu suất mô hình trên các giá trị `n_estimators` (số cây con) khác nhau để chọn ra giá trị tốt nhất.

Hàm tìm số lượng cây con tối ưu

```

1 def find_optimal_rf(
2     X_train,y_train,n_estimators_range=range(50, 501, 50),cv_splits=
3             3,
4     max_depth=5,min_samples_split=2,min_samples_leaf=1,
5     max_features='sqrt',bootstrap=True, class_weight=None
6 ):
7     cv = StratifiedKFold(n_splits=cv_splits, shuffle=True,
8                           random_state=SEED)
9     scores = []
10    for n in n_estimators_range:
11        rf = RandomForestClassifier(
12            n_estimators=n,max_depth=max_depth,min_samples_split=
13                min_samples_split,
14                min_samples_leaf=min_samples_leaf,max_features=
15                    max_features,
16                    bootstrap=bootstrap,class_weight=class_weight,n_jobs=-1,
17                    random_state=SEED
18        )
19        cv_score = cross_val_score(rf,X_train,y_train,
20                                   cv=cv,scoring='accuracy',n_jobs=-1)

```

```

16     scores.append(cv_score.mean())
17     plt.figure(figsize=(10, 6))
18     plt.plot(list(n_estimators_range), scores, 'bo-')
19     plt.title(f'Chọn n_estimators tối ưu cho Random Forest (CV={cv_splits}-fold)')
20     plt.xlabel('n_estimators')
21     plt.ylabel('Cross-Validation Accuracy')
22     plt.grid(True)
23     plt.show()
24
25     best_n = list(n_estimators_range)[int(np.argmax(scores))]
26     print(f'n_estimators tối ưu (CV): {best_n}')
27
28     best_model = RandomForestClassifier(
29         n_estimators=best_n,max_depth=max_depth,min_samples_split=
30             min_samples_split,
31             min_samples_leaf=min_samples_leaf,max_features=max_features,
32             bootstrap=bootstrap,class_weight=class_weight,n_jobs=-1,
33             random_state=SEED
34     )
35     best_model.fit(X_train, y_train)
36     return best_model, best_n, max(scores)

```

Tiếp đến, một hàm khác được tạo để huấn luyện và đánh giá mô hình trên tập dữ liệu huấn luyện và thẩm định, sau đó dự đoán và đánh giá hiệu suất trên tập kiểm thử.

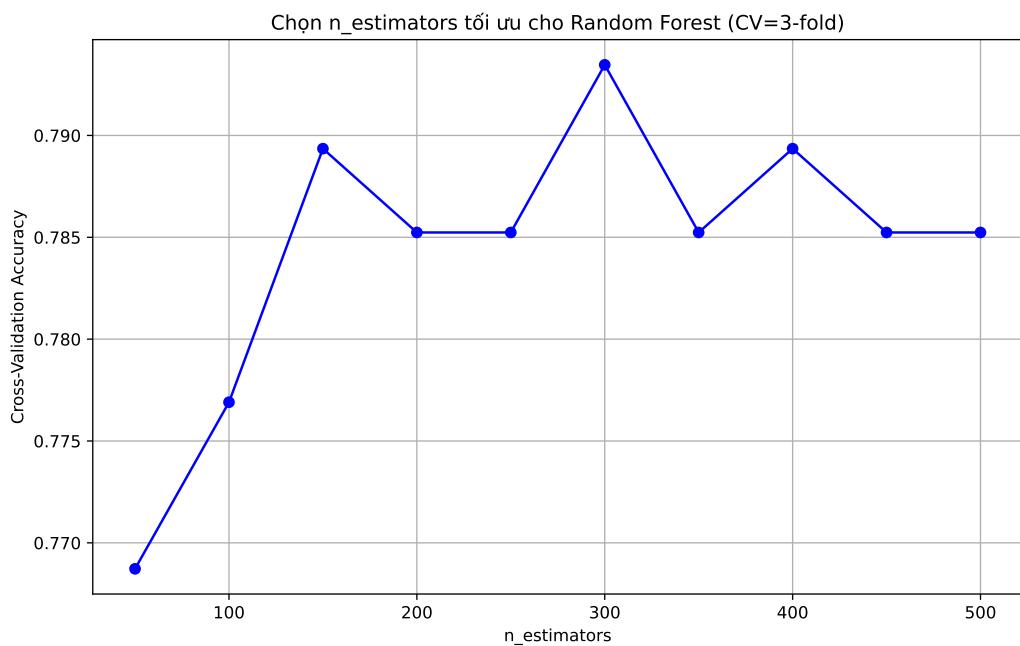
Hàm train/val và hàm test

```

1 def evaluate_val_rf(X_train, y_train, X_val, y_val,
2                     n_estimators_range=range(50, 501, 50),cv_splits=
3                         5,max_depth=5,
4                         min_samples_split=2,min_samples_leaf=1,
5                             max_features='sqrt',
6                             bootstrap=True,class_weight=None):
7     print('Tim n_estimators tối ưu cho Random Forest... ')
8     rf_model, best_n, cv_acc = find_optimal_rf(
9         X_train,y_train,n_estimators_range=n_estimators_range,
10            cv_splits=cv_splits,
11            max_depth=max_depth,min_samples_split=min_samples_split,
12            min_samples_leaf=min_samples_leaf,max_features=max_features,
13            bootstrap=bootstrap,class_weight=class_weight

```

```
11 )
12
13     val_pred = rf_model.predict(X_val)
14     val_acc = accuracy_score(y_val, val_pred)
15     print(f'\nĐộ chính xác Random Forest trên tập validation: {val_acc:.4f}')
16     print('Classification Report:')
17     print(classification_report(y_val, val_pred))
18     return rf_model, val_acc, {'n_estimators': best_n}
19
20 def evaluate_test_rf(rf_model, X_test, y_test):
21     test_pred = rf_model.predict(X_test)
22     test_acc = accuracy_score(y_test, test_pred)
23     print(f'\nĐộ chính xác Random Forest trên tập test: {test_acc:.4f}')
24     print('Classification Report:')
25     print(classification_report(y_test, test_pred))
26     return test_acc
```



Hình 20.7: Biểu đồ độ chính xác khi áp dụng Stratified K-Fold Cross-Validation cho mô hình Random Forest tại các giá trị `n_estimators` khác nhau.

Sử dụng các hàm trên, tiến hành huấn luyện và đánh giá mô hình Random Forest trên bốn bộ dữ liệu khác nhau: dữ liệu gốc, dữ liệu FE, dữ liệu gốc kết hợp với DT, và dữ liệu FE kết hợp với DT.

Huấn luyện và đánh giá trên các tập dữ liệu

```

1 # RF on Original Dataset
2 rf_model, val_acc, best_params = evaluate_val_rf(
3     X_train, y_train, X_val, y_val
4 )
5 test_acc = evaluate_test_rf(rf_model, X_test, y_test)
6
7 # RF on Feature Engineering Dataset
8 rf_model, val_fe_acc, best_params = evaluate_val_rf(
9     X_fe_train, y_fe_train, X_fe_val, y_fe_val
10 )
11

```

```

12 # RF on Original DT Dataset
13 rf_model, val_dt_acc, best_params = evaluate_val_rf(
14     X_dt_train, y_dt_train, X_dt_val, y_dt_val
15 )
16 test_dt_acc = evaluate_test_rf(rf_model, X_dt_test, y_dt_test)
17
18 # RF on Feature Engineering DT Dataset
19 rf_model, val_fe_dt_acc, best_params = evaluate_val_rf(
20     X_fe_dt_train, y_fe_dt_train, X_fe_dt_val, y_fe_dt_val,
21 )
22 test_fe_dt_acc = evaluate_test_rf(rf_model, X_fe_dt_test,
23                                     y_fe_dt_test)

```

Cuối cùng, một biểu đồ cột được vẽ để so sánh trực quan hiệu suất (độ chính xác) của mô hình trên các tập dữ liệu thẩm định và kiểm thử, giúp dễ dàng nhận biết bộ dữ liệu nào mang lại kết quả tốt nhất.

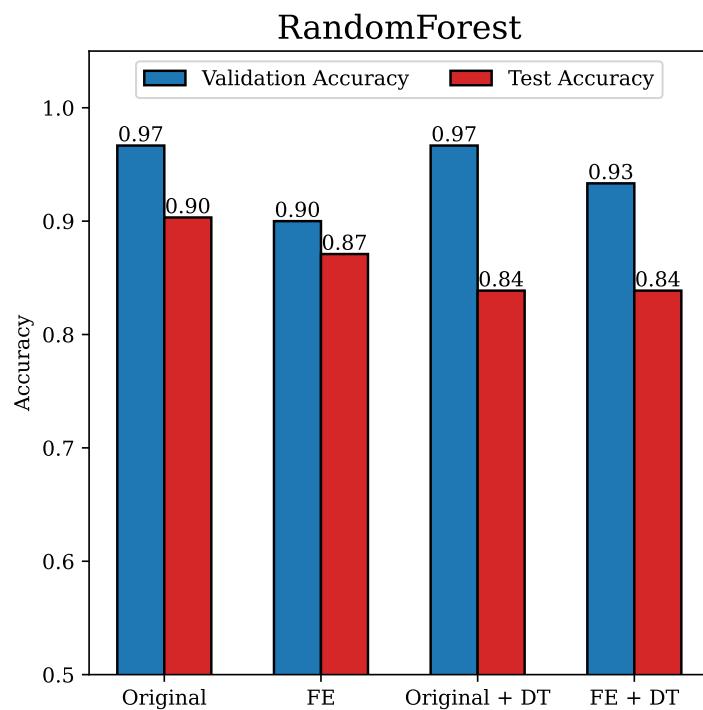
*Vẽ biểu đồ so sánh hiệu suất mô hình trên các tập dữ liệu

```

1 # Just change the title in line 22 to the corresponding model name
2 plt.rcParams['font.family'] = 'Serif'
3
4 labels = ['Original', 'FE', 'Original + DT', "FE + DT"]
5 val_accs = [val_acc, val_fe_acc, val_dt_acc, val_fe_dt_acc]
6 test_accs = [test_acc, test_fe_acc, test_dt_acc, test_fe_dt_acc]
7
8 x = np.arange(len(labels))
9 width = 0.3
10
11 fig, ax = plt.subplots(figsize=(5, 5))
12
13 rects1 = ax.bar(x - width/2, val_accs, width,
14                  label='Validation Accuracy',
15                  color='tab:blue', edgecolor='black', linewidth=1.2)
16 rects2 = ax.bar(x + width/2, test_accs, width,
17                  label='Test Accuracy',
18                  color='tab:red', edgecolor='black', linewidth=1.2)
19
20 ax.set_ylim(0.5, 1.05)
21 ax.set_ylabel('Accuracy')
22 ax.set_title('RandomForest', fontsize=16)
23 ax.set_xticks(x)

```

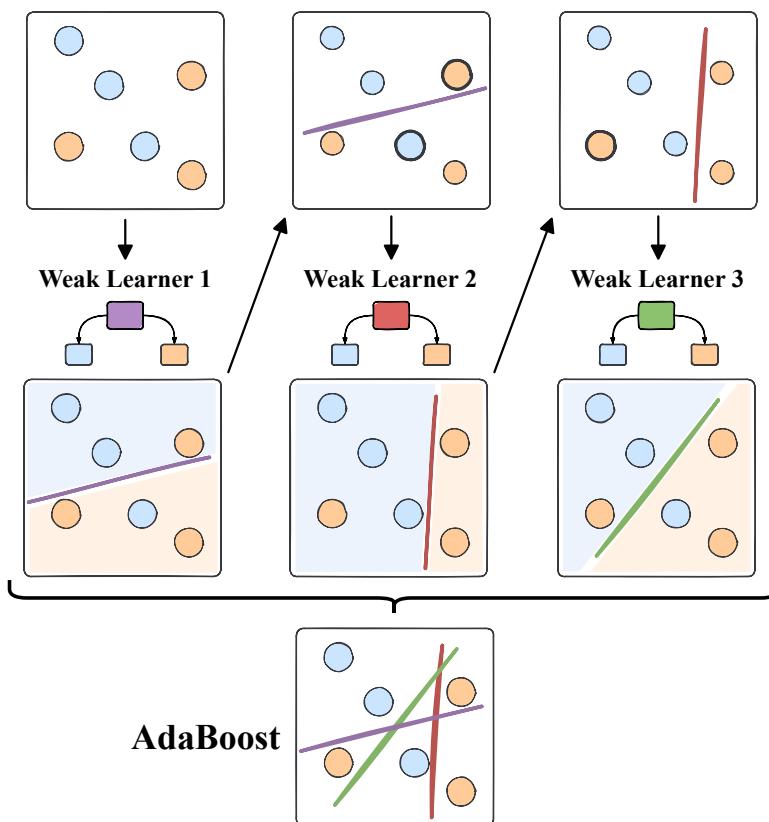
```
24 ax.set_xticklabels(labels)
25 ax.legend(ncol=2, loc="upper center")
26
27 def autolabel(rects):
28     for rect in rects:
29         h = rect.get_height()
30         ax.annotate(f'{h:.2f}', xy=(rect.get_x() + rect.get_width() / 2,
31                                     h),
32                     ha='center', va='bottom')
33
34 autolabel(rects1)
35 autolabel(rects2)
36
37 fig.tight_layout()
38 plt.show()
```



Hình 20.8: Hiệu suất của mô hình Random Forest trên 4 bộ dữ liệu.

20.2.4 AdaBoost

Giới thiệu



Hình 20.9: Hình minh họa thuật toán AdaBoost cho bài toán phân loại.

AdaBoost [?] hoạt động theo một quy trình lặp lại, chú trọng vào các điểm dữ liệu mà các mô hình trước đó đã phân loại sai. Cụ thể như sau:

- Ban đầu, mỗi điểm dữ liệu trong tập huấn luyện đều có một trọng số như nhau.
- Một mô hình yếu (Weak Learner 1), chẳng hạn như một cây quyết định đơn giản (stump), được huấn luyện trên dữ liệu.
- Sau khi huấn luyện mô hình yếu đầu tiên, trọng số của các điểm dữ liệu sẽ được điều chỉnh. Các điểm dữ liệu mà mô hình phân loại sai sẽ

được tăng trọng số, trong khi các điểm phân loại đúng sẽ giảm trọng số. Điều này buộc mô hình yếu tiếp theo phải tập trung hơn vào các điểm dữ liệu khó.

- Quá trình này lặp lại nhiều lần với các mô hình yếu mới (Weak Learner 2, Weak Learner 3). Mỗi mô hình mới sẽ được huấn luyện trên tập dữ liệu đã được điều chỉnh trọng số, do đó tập trung vào những lỗi của mô hình trước đó.
- Cuối cùng, AdaBoost sẽ kết hợp tất cả các mô hình yếu lại với nhau thành một mô hình tổng thể. Mỗi mô hình yếu sẽ được gán một trọng số dựa trên hiệu suất của nó - những mô hình có độ chính xác cao sẽ có trọng số lớn hơn trong quyết định cuối cùng.

Kết quả là một mô hình dự đoán mạnh mẽ và có độ chính xác cao, được xây dựng từ sự đóng góp của nhiều mô hình yếu, mỗi mô hình đều tập trung vào việc khắc phục những điểm yếu của mô hình trước đó.

Triển khai

Xây dựng một hàm để tìm số lượng cây con tối ưu cho mô hình AdaBoost bằng cách sử dụng Stratified K-Fold Cross-Validation để đánh giá hiệu suất qua các giá trị `n_estimators` khác nhau.

Hàm tìm số lượng cây con tối ưu

```

1 def find_optimal_ada(
2     X_train, y_train,
3     n_estimators_range=range(50, 501, 50),
4     cv_splits=3,
5     learning_rate=0.1,
6     base_max_depth=1,
7     algorithm='SAMME'
8 ):
9     cv = StratifiedKFold(n_splits=cv_splits, shuffle=True,
10                           random_state=SEED)
11     scores = []
12     for n in n_estimators_range:
13         ada = AdaBoostClassifier(

```

```

14         estimator=DecisionTreeClassifier(max_depth=
15                         base_max_depth, random_state=SEED)
16                         ,
17                         n_estimators=n, learning_rate=learning_rate,
18                         algorithm=algorithm, random_state=SEED
19         )
20         cv_score = cross_val_score(
21             ada, X_train, y_train, cv=cv, scoring='accuracy', n_jobs
22                         ==-1
23         )
24         scores.append(cv_score.mean())
25
26         plt.figure(figsize=(10, 6))
27         plt.plot(list(n_estimators_range), scores, 'bo-')
28         plt.title(f'Chọn n_estimators tối ưu cho AdaBoost (CV={cv_splits
29                         }-fold)')
30         plt.xlabel('n_estimators')
31         plt.ylabel('Cross-Validation Accuracy')
32         plt.grid(True)
33         plt.show()
34
35         best_n = list(n_estimators_range)[int(np.argmax(scores))]
36         print(f'n_estimators tối ưu (CV): {best_n}')
37
38         best_model = AdaBoostClassifier(
39             estimator=DecisionTreeClassifier(max_depth=base_max_depth,
40                         random_state=SEED),
41                         n_estimators=best_n, learning_rate=learning_rate,
42                         algorithm=algorithm, random_state=SEED
43         )
44         best_model.fit(X_train, y_train)
45         return best_model, best_n, max(scores)

```

Tiếp theo, tạo các hàm để huấn luyện và đánh giá mô hình AdaBoost. Một hàm sẽ huấn luyện mô hình trên tập train và đánh giá trên tập val, sau đó hàm còn lại sẽ đánh giá hiệu suất trên tập test.

Hàm train/val và hàm test

```

1 def evaluate_val_ada(X_train, y_train, X_val, y_val,
2                         n_estimators_range=range(50, 501, 50),
3                         cv_splits=3,

```

```

4             learning_rate=0.1,
5             base_max_depth=1,
6             algorithm='SAMME'):
7     print('Tìm n_estimators tối ưu cho AdaBoost... ')
8     ada_model, best_n, cv_acc = find_optimal_ada(
9         X_train, y_train,
10        n_estimators_range=n_estimators_range,
11        cv_splits=cv_splits,
12        learning_rate=learning_rate,
13        base_max_depth=base_max_depth,
14        algorithm=algorithm
15    )
16
17    val_pred = ada_model.predict(X_val)
18    val_acc = accuracy_score(y_val, val_pred)
19    print(f'\nĐộ chính xác AdaBoost trên tập validation: {val_acc:.4f}')
20    print('Classification Report:')
21    print(classification_report(y_val, val_pred))
22    return ada_model, val_acc, {'n_estimators': best_n}
23
24 def evaluate_test_ada(ada_model, X_test, y_test):
25     test_pred = ada_model.predict(X_test)
26     test_acc = accuracy_score(y_test, test_pred)
27     print(f'\nĐộ chính xác AdaBoost trên tập test: {test_acc:.4f}')
28     print('Classification Report:')
29     print(classification_report(y_test, test_pred))
30     return test_acc

```

Cuối cùng, sử dụng các hàm đã định nghĩa để huấn luyện và đánh giá mô hình trên bốn bộ dữ liệu khác nhau, bao gồm dữ liệu gốc, dữ liệu đã qua kỹ thuật đặc trưng (FE), dữ liệu kết hợp với Decision Tree (DT), và dữ liệu FE kết hợp với DT.

Huấn luyện và đánh giá trên các tập dữ liệu

```

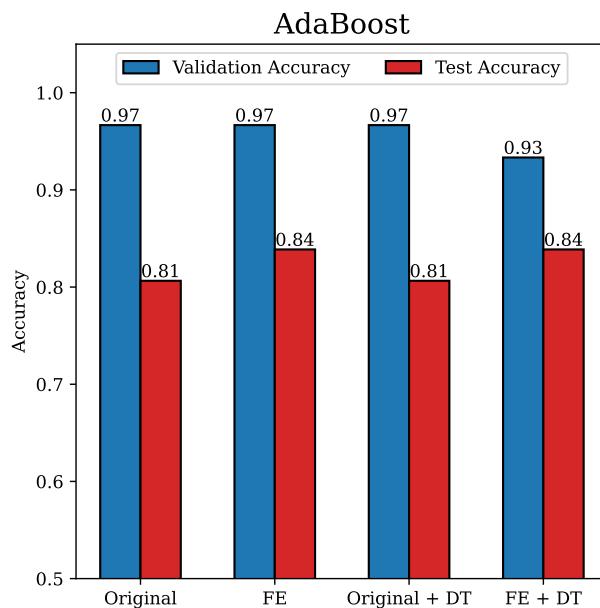
1 # AB on Original Dataset
2 ada_model, val_acc, best_params = evaluate_val_ada(
3     X_train, y_train, X_val, y_val
4 )
5 test_acc = evaluate_test_ada(ada_model, X_test, y_test)

```

```

6
7 # AB on FE Dataset
8 ada_model, val_fe_acc, best_params = evaluate_val_ada(
9     X_fe_train, y_fe_train, X_fe_val, y_fe_val
10)
11 test_fe_acc = evaluate_test_ada(ada_model, X_fe_test, y_fe_test)
12
13 # AB on Original DT Dataset
14 ada_model, val_dt_acc, best_params = evaluate_val_ada(
15     X_dt_train, y_dt_train, X_dt_val, y_dt_val
16)
17 test_dt_acc = evaluate_test_ada(ada_model, X_dt_test, y_dt_test)
18
19 # AB on FE + DT Dataset
20 ada_model, val_fe_dt_acc, best_params = evaluate_val_ada(
21     X_fe_dt_train, y_fe_dt_train, X_fe_dt_val, y_fe_dt_val,
22)
23 test_fe_dt_acc = evaluate_test_ada(ada_model, X_fe_dt_test,
y_fe_dt_test)

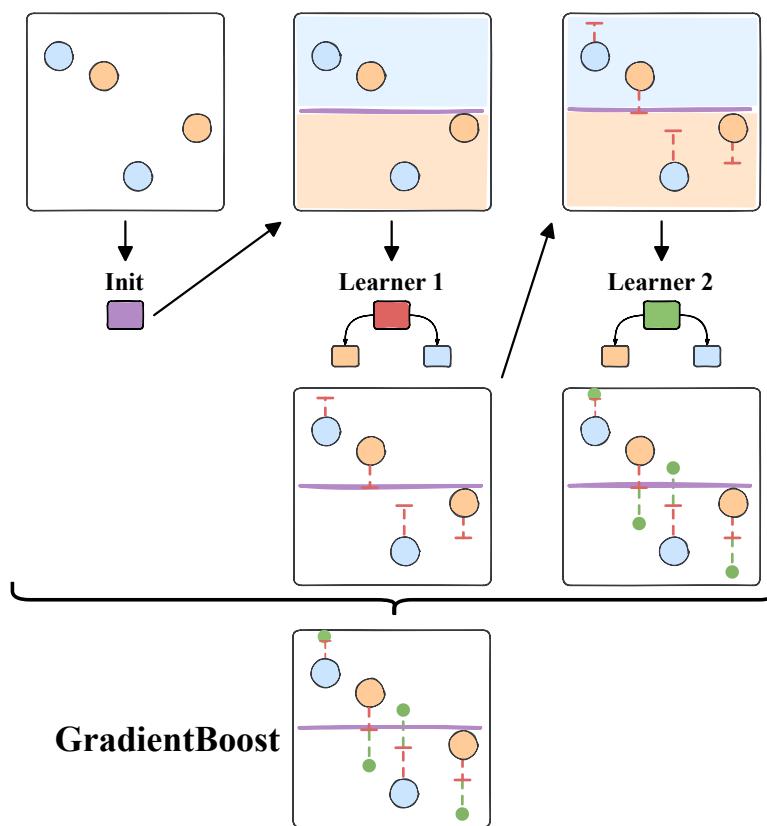
```



Hình 20.10: Hiệu suất của mô hình AdaBoost trên 4 bộ dữ liệu.

20.2.5 Gradient Boosting

Giới thiệu



Hình 20.11: Hình minh họa thuật toán Gradient Boosting cho bài toán phân loại.

Gradient Boosting bắt đầu với một mô hình ban đầu rất đơn giản (Init), thường là một giá trị trung bình hoặc một hằng số. Sau đó lặp lại quá trình sau:

- Tại mỗi bước, thuật toán sẽ tính toán phần dư, tức là sự khác biệt giữa giá trị thực tế và giá trị dự đoán của mô hình hiện tại. Phần dư này chính là "lỗi" mà mô hình cần phải học cách sửa.

- Một mô hình mới, thường là một cây quyết định (Learner 1, Learner 2, ...), được huấn luyện để dự đoán chính xác những phần dư đó. Thay vì dự đoán kết quả cuối cùng, mô hình này tập trung vào việc sửa lỗi của mô hình trước.
- Mô hình mới này sẽ được thêm vào mô hình tổng thể, cùng với một hệ số học (learning rate) nhỏ để kiểm soát tốc độ học, giúp tránh việc mô hình trở nên quá phức tạp (overfitting) một cách nhanh chóng.

Quá trình này lặp lại cho đến khi mô hình tổng thể đạt được độ chính xác mong muốn hoặc đã xây dựng đủ số lượng cây. Kết quả cuối cùng là một mô hình tổng hợp, cực kỳ hiệu quả, được tạo thành từ tổng hợp của tất cả các mô hình nhỏ đã được huấn luyện tuần tự.

Triển khai

Tương tự thuật toán trước, tạo một hàm để tìm số lượng cây con tối ưu cho mô hình Gradient Boosting bằng cách sử dụng Stratified K-Fold Cross-Validation nhằm xác định hiệu suất qua các giá trị `n_estimators`.

Hàm tìm số lượng cây con tối ưu

```

1 def find_optimal_gb(
2     X_train, y_train,
3     n_estimators_range=range(50, 501, 50),
4     cv_splits=3
5 ):
6     cv = StratifiedKFold(n_splits=cv_splits, shuffle=True,
7                           random_state=SEED)
8     scores = []
9
10    for n in n_estimators_range:
11        gb = GradientBoostingClassifier(
12            n_estimators=n, learning_rate=0.1,
13            max_depth=5, subsample=1.0, random_state=SEED
14        )
15        cv_score = cross_val_score(
16            gb, X_train, y_train,
17            cv=cv, scoring='accuracy', n_jobs=-1
18        )
19        scores.append(cv_score.mean())

```

```

19     plt.figure(figsize=(10, 6))
20     plt.plot(list(n_estimators_range), scores, 'bo-')
21     plt.title(f'Chọn n_estimators tối ưu cho Gradient Boosting (CV={cv_splits}-fold)')
22
23     plt.xlabel('n_estimators')
24     plt.ylabel('Cross-Validation Accuracy')
25     plt.grid(True)
26     plt.show()
27
28     best_n = list(n_estimators_range)[int(np.argmax(scores))]
29     print(f'n_estimators tối ưu (CV): {best_n}')
30
31     best_model = GradientBoostingClassifier(
32         n_estimators=best_n,
33         learning_rate=0.1,
34         max_depth=5,
35         subsample=1.0,
36         random_state=SEED
37     )
38     best_model.fit(X_train, y_train)
39     return best_model, best_n, max(scores)

```

Tiếp theo, xây dựng các hàm để huấn luyện trên tập train và đánh giá mô hình Gradient Boosting trên tập val và tập test.

Hàm train/val và hàm test

```

1 def evaluate_val_gb(X_train, y_train, X_val, y_val,
2                     n_estimators_range=range(50, 501, 50),
3                     cv_splits=3):
4     print('Tìm n_estimators tối ưu cho Gradient Boosting... ')
5     gb_model, best_n, cv_acc = find_optimal_gb(
6         X_train, y_train,
7         n_estimators_range=n_estimators_range,
8         cv_splits=cv_splits
9     )
10
11     val_pred = gb_model.predict(X_val)
12     val_acc = accuracy_score(y_val, val_pred)
13     print(f'\nĐộ chính xác GB trên tập validation: {val_acc:.4f}')
14     print('Classification Report: ')

```

```

15     print(classification_report(y_val, val_pred))
16     return gb_model, val_acc, {'n_estimators': best_n}
17
18 def evaluate_test_gb(gb_model, X_test, y_test):
19     test_pred = gb_model.predict(X_test)
20     test_acc = accuracy_score(y_test, test_pred)
21     print(f'\nĐộ chính xác GB trên tập test: {test_acc:.4f}')
22     print('Classification Report:')
23     print(classification_report(y_test, test_pred))
24     return test_acc

```

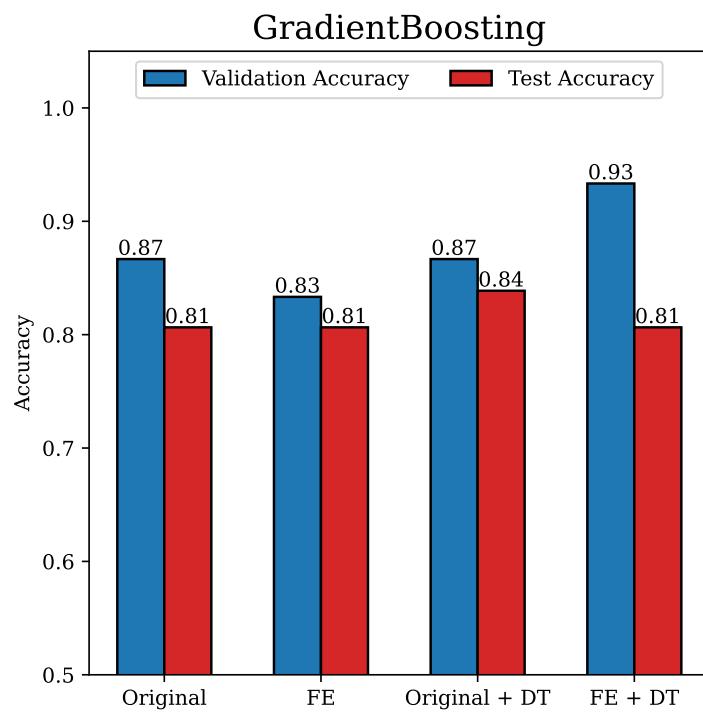
Sau đó, tiến hành huấn luyện và đánh giá mô hình trên bốn bộ dữ liệu khác nhau.

Huấn luyện và đánh giá trên các tập dữ liệu

```

1 # GB on Original Dataset
2 gb_model, val_acc, best_params = evaluate_val_gb(
3     X_train, y_train, X_val, y_val
4 )
5 test_acc = evaluate_test_gb(gb_model, X_test, y_test)
6
7 # GB on Feature Engineering Dataset
8 gb_model, val_fe_acc, best_params = evaluate_val_gb(
9     X_fe_train, y_fe_train, X_fe_val, y_fe_val
10 )
11 test_fe_acc = evaluate_test_gb(gb_model, X_fe_test, y_fe_test)
12
13 # GB on Original DT Dataset
14 gb_model, val_dt_acc, best_params = evaluate_val_gb(
15     X_dt_train, y_dt_train, X_dt_val, y_dt_val
16 )
17 test_dt_acc = evaluate_test_gb(gb_model, X_dt_test, y_dt_test)
18
19 # GB on Feature Engineering DT Dataset
20 gb_model, val_fe_dt_acc, best_params = evaluate_val_gb(
21     X_fe_dt_train, y_fe_dt_train, X_fe_dt_val, y_fe_dt_val,
22 )
23 test_fe_dt_acc = evaluate_test_gb(gb_model, X_fe_dt_test,
24                                     y_fe_dt_test)

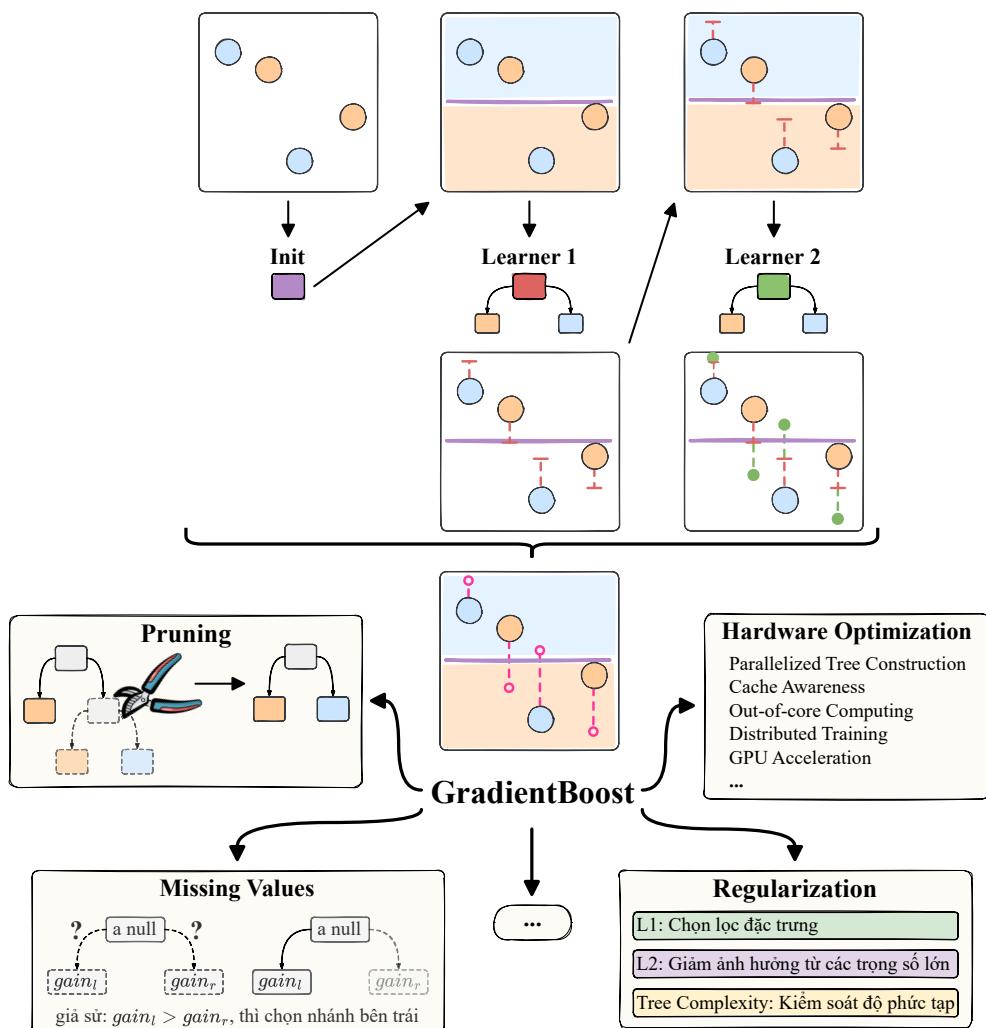
```



Hình 20.12: Hiệu suất của mô hình Gradient Boosting trên 4 bộ dữ liệu.

20.2.6 XGBoost

Giới thiệu



Hình 20.13: Hình minh họa thuật toán XGBoost cho bài toán phân loại.

XGBoost [?] cải tiến từ Gradient Boosting bằng cách kết hợp nhiều kỹ thuật để tối ưu hóa hiệu suất và giảm thiểu các vấn đề như quá khớp (overfitting). Cụ thể, các đặc điểm nổi bật của nó bao gồm:

- Sử dụng kỹ thuật như Parallelized Tree Construction (xây dựng cây song song) và GPU Acceleration (tăng tốc bằng GPU) để tăng tốc độ tính toán, đặc biệt với các bộ dữ liệu lớn.
- Tự động xử lý các Missing Values (các giá trị bị thiếu) bằng cách học cách phân nhánh tốt nhất ngay cả khi dữ liệu bị thiếu.
- Áp dụng các kỹ thuật Regularization (điều chỉnh) như L1 và L2 để kiểm soát độ phức tạp của mô hình và tránh việc mô hình học quá sát với dữ liệu huấn luyện. Pruning (cắt tia cây) cũng được sử dụng để loại bỏ các nhánh không cần thiết.

Về cơ bản, XGBoost hoạt động bằng cách xây dựng một loạt các cây quyết định một cách tuần tự (như trong sơ đồ là Learner 1 và Learner 2). Mỗi cây mới được thêm vào sẽ cố gắng sửa lỗi của các cây trước đó, dần dần cải thiện độ chính xác tổng thể của mô hình.

Triển khai

Bắt đầu tương tự như các thuật toán trên với một hàm được tìm số lượng cây con tối ưu cho mô hình XGBoost bằng cách sử dụng Stratified K-Fold Cross-Validation với các giá trị `n_estimators` khác nhau.

Hàm tìm số lượng cây con tối ưu

```

1 def find_optimal_xgb(
2     X_train, y_train,
3     n_estimators_range=range(50, 501, 50),
4     cv_splits=3,
5     learning_rate=0.1,
6     max_depth=5,
7     subsample=1.0,
8     use_gpu=False
9 ):
10    cv = StratifiedKFold(n_splits=cv_splits, shuffle=True,
11                          random_state=SEED)
12    scores = []
13
14    n_classes = len(np.unique(y_train))
        objective = 'binary:logistic' if n_classes == 2 else 'multi:
                                softprob'

```

```
15     eval_metric = 'logloss' if n_classes == 2 else 'mlogloss'
16
17     for n in n_estimators_range:
18         xgb = XGBClassifier(
19             n_estimators=n, learning_rate=learning_rate, max_depth=
20                             max_depth, subsample=subsample,
21                             objective=objective, eval_metric=eval_metric,
22                             random_state=SEED, n_jobs=-1,
23                             tree_method='gpu_hist' if use_gpu else 'hist', verbosity
24                             =0
25         )
26         cv_score = cross_val_score(
27             xgb, X_train, y_train,
28             cv=cv, scoring='accuracy', n_jobs=-1
29         )
30         scores.append(cv_score.mean())
31
32     plt.figure(figsize=(10, 6))
33     plt.plot(list(n_estimators_range), scores, 'bo-')
34     plt.title(f'Chọn n_estimators tối ưu cho XGBoost (CV=[cv_splits]
35 -fold)')
36     plt.xlabel('n_estimators')
37     plt.ylabel('Cross-Validation Accuracy')
38     plt.grid(True)
39     plt.show()
40
41     best_n = list(n_estimators_range)[int(np.argmax(scores))]
42     print(f'n_estimators tối ưu (CV): {best_n}')
43
44     best_model = XGBClassifier(
45         n_estimators=best_n, learning_rate=learning_rate, max_depth=
46                         max_depth, subsample=subsample,
47                         objective=objective, eval_metric=eval_metric, random_state=
48                             SEED, n_jobs=-1,
49                         tree_method='gpu_hist' if use_gpu else 'hist', verbosity=0
50     )
51     best_model.fit(X_train, y_train)
52     return best_model, best_n, max(scores)
```

Dựa trên hàm tìm kiếm tham số tối ưu, các hàm tiếp theo được tạo để huấn luyện và đánh giá hiệu suất của mô hình.

Hàm train/val và hàm test

```

1 def evaluate_val_xgb(X_train, y_train, X_val, y_val,
2                         n_estimators_range=range(50, 501, 50),
3                         cv_splits=3,
4                         learning_rate=0.1, max_depth=5, subsample=1.0,
5                         colsample_bytree=1.0, use_gpu=False):
6     print('Tìm n_estimators tối ưu cho XGBoost...')
7     xgb_model, best_n, cv_acc = find_optimal_xgb(
8         X_train, y_train, n_estimators_range=n_estimators_range,
9         cv_splits=cv_splits, learning_rate=learning_rate,
10        max_depth=max_depth, subsample=subsample, use_gpu=use_gpu
11    )
12
13     val_pred = xgb_model.predict(X_val)
14     val_acc = accuracy_score(y_val, val_pred)
15     print(f'\nĐộ chính xác XGBoost trên tập validation: {val_acc:.4f}')
16
17     print('Classification Report:')
18     print(classification_report(y_val, val_pred))
19     return xgb_model, val_acc, {'n_estimators': best_n}
20
21
22 def evaluate_test_xgb(xgb_model, X_test, y_test):
23     test_pred = xgb_model.predict(X_test)
24     test_acc = accuracy_score(y_test, test_pred)
25     print(f'\nĐộ chính xác XGBoost trên tập test: {test_acc:.4f}')
26     print('Classification Report:')
27     print(classification_report(y_test, test_pred))
28     return test_acc

```

Sau khi đã có các hàm cần thiết, tiến hành huấn luyện và đánh giá mô hình trên bốn bộ dữ liệu khác nhau.

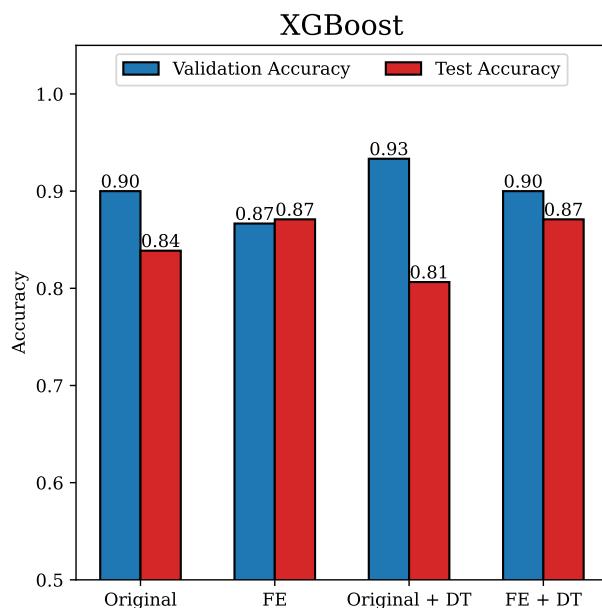
Huấn luyện và đánh giá trên các tập dữ liệu

```

1 # XGB on Original Dataset
2 xgb_model, val_acc, best_params = evaluate_val_xgb(
3     X_train, y_train, X_val, y_val
4 )
5 test_acc = evaluate_test_xgb(xgb_model, X_test, y_test)
6
7 # XGB on Feature Engineering Dataset

```

```
8 xgb_model, val_fe_acc, best_params = evaluate_val_xgb(
9     X_fe_train, y_fe_train, X_fe_val, y_fe_val
10)
11 test_fe_acc = evaluate_test_xgb(xgb_model, X_fe_test, y_fe_test)
12
13 # XGB on Original DT Dataset
14 xgb_model, val_dt_acc, best_params = evaluate_val_xgb(
15     X_dt_train, y_dt_train, X_dt_val, y_dt_val
16)
17 test_dt_acc = evaluate_test_xgb(xgb_model, X_dt_test, y_dt_test)
18
19 # XGB on Feature Engineering DT Dataset
20 xgb_model, val_fe_dt_acc, best_params = evaluate_val_xgb(
21     X_fe_dt_train, y_fe_dt_train, X_fe_dt_val, y_fe_dt_val,
22)
23 test_fe_dt_acc = evaluate_test_xgb(xgb_model, X_fe_dt_test,
24                                     y_fe_dt_test)
```



Hình 20.14: Hiệu suất của mô hình XGBoost trên 4 bộ dữ liệu.

20.2.7 Tổng hợp kết quả

Trong phần này, ta lập một bảng ghi nhận độ chính xác của các phương pháp trên tập val và test của từng bộ dữ liệu. Đây là một cách làm thường thấy trong các bài báo khoa học, cung cấp cho người đọc một góc nhìn toàn diện về khả năng của tất cả các phương pháp học máy được xem xét, trong trường hợp này là thuộc xuyên suốt 2 phần của dự án chẩn đoán bệnh tim của chúng ta. Theo đó, các phương pháp được đặt lần lượt theo thứ tự trình bày trong project. Có 4 tập dữ liệu bao gồm bộ gốc (Origin) và các bộ với các kiểu phối hợp tiền xử lý khác nhau như Feature Engineering (FE) và chọn lọc đặc trưng với Decision Tree (DT). Hiệu suất của các mô hình được đo bằng Accuracy (độ chính xác).

Bảng 20.2: Bảng tổng hợp các kết quả thực nghiệm trên cả 2 dự án (Part 1 và 2). Trong đó, FE là viết tắt của “Feature Engineering”; DT là viết tắt của “Decision Tree”.

Model	Val				Test			
	Origin	FE	Origin+DT	FE+DT	Origin	FE	Origin+DT	FE+DT
Naive Bayes	.90	.90	.93	.93	.84	.84	.84	.84
KNN	.90	.90	.97	.87	.84	.84	.87	.84
KMeans	.70	.80	.83	.63	.87	.87	.84	.77
Decision Tree	.93	.93	.93	.93	.81	.81	.81	.81
Ensemble (KNN,DT,NB)	.87	.93	.90	.87	.84	.90	.84	.84
Random Forest	.97	.90	.97	.93	.90	.87	.84	.84
AdaBoost	.97	.97	.97	.93	.81	.84	.81	.84
Gradient Boosting	.87	.83	.87	.93	.81	.81	.84	.81
XGBoost	.90	.87	.93	.90	.84	.87	.81	.87

Dựa vào Bảng 20.2, ta nhận thấy độ chính xác cao nhất có thể đạt được trong bài là 97%. Các phương pháp thuộc nhóm Ensemble Learning, cụ thể với Random Forest và AdaBoost, cho kết quả khả quan nhất trên cả 4 kiểu dataset. Bên cạnh đó, việc áp dụng tiền xử lý dữ liệu nhìn chung có thể giúp các phương pháp học máy cải thiện độ chính xác một cách tương đối. Điều này phản ánh cho chúng ta tầm quan trọng của việc áp dụng tiền xử lý dữ liệu cũng như độ hiệu quả của các phương pháp có ứng dụng Ensemble Learning.

20.2.8 Cài đặt UI và Deploy

Để nâng cao tính hoàn thiện và tính trực quan của dự án, việc xây dựng một bản demo với giao diện đơn giản là cần thiết. Bản demo đóng vai trò minh họa trực tiếp cho hiệu quả của mô hình, đồng thời hỗ trợ quá trình kiểm thử. Qua đó, kết quả nghiên cứu không chỉ dừng lại ở mức lý thuyết, mà còn được cụ thể hóa thành một sản phẩm có thể sử dụng được.

Google Colab không được thiết kế để lưu trữ ứng dụng web, nên để chạy và truy cập giao diện Streamlit từ Colab ta cần tạo một đường hầm (tunnel) an toàn. Quy trình gồm: (1) viết mã ứng dụng vào app.py, (2) cài các phụ thuộc và khởi chạy Streamlit trên Colab, (3) dùng Cloudflare để mở tunnel từ máy chủ Colab ra ngoài, qua đó nhận một URL công khai để truy cập giao diện từ trình duyệt.

Bước 1: Tạo file ứng dụng app.py

Đầu tiên, chúng ta cần viết toàn bộ code của ứng dụng Streamlit vào một file Python. Trong môi trường Colab, ta sử dụng “magic command” `%%writefile` để thực hiện việc này. Cell đầu tiên này sẽ tạo ra file `app.py` và chứa toàn bộ logic của ứng dụng, từ tải dữ liệu, huấn luyện mô hình cho đến hiển thị giao diện.

Khai báo thư viện và Cấu hình

app.py - Phần 1

```
1 %%writefile app.py
2 import streamlit as st
3 import pandas as pd
4 from sklearn.model_selection import GridSearchCV
5 from sklearn.preprocessing import OneHotEncoder
6 from sklearn.compose import ColumnTransformer
7 from sklearn.pipeline import Pipeline
8 from sklearn.impute import SimpleImputer
9 from sklearn.tree import DecisionTreeClassifier, plot_tree
10 import matplotlib.pyplot as plt
11
```

```

12 st.set_page_config(page_title='Heart Disease Prediction', layout='wide')
13 URL = 'https://archive.ics.uci.edu/ml/machine-learning-databases/
           heart-disease/processed.cleveland.
           data'
14 COLUMNS = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
15           'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target']
16 NUMERIC = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
17 CATEGORICAL = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal
                  '']

```

- Giải thích:** Import các thư viện cần thiết, bao gồm GridSearchCV để tự động tối ưu tham số. Chúng ta cũng cấu hình tiêu đề, icon cho trang web và định nghĩa các hằng số chứa URL dữ liệu và danh sách các cột.

Tải dữ liệu, Tối ưu hóa và Huấn luyện Mô hình

app.py - Phần 2

```

1 @st.cache_resource
2 def find_best_model():
3     df = pd.read_csv(URL, header=None, names=COLUMNS, na_values='?')
4     df = df.dropna()
5     df['target'] = (df['target'] > 0).astype(int)
6
7     X = df[NUMERIC + CATEGORICAL]
8     y = df['target']
9
10    num_pipe = Pipeline([('imputer', SimpleImputer(strategy='median'))])
11    cat_pipe = Pipeline([
12        ('imputer', SimpleImputer(strategy='most_frequent')),
13        ('ohe', OneHotEncoder(handle_unknown='ignore', sparse_output
14                           =False))
15    ])
16
17    preprocessor = ColumnTransformer([
18        ('num', num_pipe, NUMERIC),
19        ('cat', cat_pipe, CATEGORICAL)
20    ])

```

```
20     pipeline = Pipeline([
21         ('preprocessor', preprocessor),
22         ('classifier', DecisionTreeClassifier(random_state=42))
23     ])
24
25     param_grid = {'classifier__max_depth': range(3, 11)}
26
27     grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='
28                                 accuracy', n_jobs=-1)
29     grid_search.fit(X, y)
30
31     return grid_search
32
33 model = find_best_model()
```

- **Giải thích:** Đây là phần logic cốt lõi. Hàm `find_best_model` thực hiện toàn bộ quy trình:
 - Tải dữ liệu từ URL và làm sạch cơ bản.
 - Xây dựng một `Pipeline` hoàn chỉnh để xử lý dữ liệu số, dữ liệu phân loại và đưa vào mô hình Cây quyết định.
 - Định nghĩa một không gian tham số (`param_grid`) để tìm kiếm độ sâu (`max_depth`) tốt nhất cho cây.
 - Sử dụng `GridSearchCV` để thực hiện kiểm định chéo (cross-validation) 5-fold, tự động tìm ra tham số tối ưu và huấn luyện lại mô hình tốt nhất trên toàn bộ dữ liệu.
 - Decorator `@st.cache_resource` đảm bảo toàn bộ quá trình tốn nhiều thời gian này chỉ chạy một lần duy nhất khi ứng dụng khởi động. Kết quả (mô hình đã được tối ưu) sẽ được lưu vào cache và tái sử dụng, giúp ứng dụng phản hồi ngay lập tức.

Xây dựng Giao diện Người dùng (UI)

app.py - Phần 3

```
1 st.title('Heart Disease Prediction App')
2 st.write('Enter patient data in the sidebar. The model will predict
           the likelihood of heart disease.')
3 st.sidebar.info(f'Optimal model depth found: {model.best_params_['
                  classifier__max_depth']}')
4
5 with st.sidebar.form('input_form'):
6     st.header('Patient Parameters')
7     age = st.slider('Age', 20, 80, 50)
8     sex = st.selectbox('Gender', [('Male', 1), ('Female', 0)],
9                         format_func=lambda x: x[0])[1]
10    cp = st.selectbox('Chest Pain Type', [('Typical Angina', 1), ('Atypical Angina', 2), ('Non-anginal Pain', 3), ('Asymptomatic', 4)], format_func=lambda x: x[0])[1]
11    trestbps = st.slider('Resting Blood Pressure (mmHg)', 90, 200, 120)
12    chol = st.slider('Serum Cholesterol (mg/dl)', 120, 570, 240)
13    fbs = st.selectbox('Fasting Blood Sugar > 120 mg/dl', [('False', 0), ('True', 1)], format_func=lambda x: x[0])[1]
14    restecg = st.selectbox('Resting ECG', [('Normal', 0), ('ST-T Abnormality', 1), ('LV Hypertrophy', 2)], format_func=lambda x: x[0])[1]
15    thalach = st.slider('Max Heart Rate Achieved', 70, 210, 150)
16    exang = st.selectbox('Exercise-induced Angina', [('No', 0), ('Yes', 1)], format_func=lambda x: x[0])[1]
17    oldpeak = st.slider('ST Depression', 0.0, 6.2, 1.0, step=0.1)
18    slope = st.selectbox('Slope of Peak ST', [('Upsloping', 1), ('Flat', 2), ('Downsloping', 3)], format_func=lambda x: x[0])[1]
19    ca = st.slider('Major Vessels colored by Fluoroscopy', 0, 3, 0)
20    thal = st.selectbox('Thalassemia', [('Normal', 3), ('Fixed Defect', 6), ('Reversible Defect', 7)], format_func=lambda x: x[0])[1]
21    submitted = st.form_submit_button('Predict')
```



- **Giải thích:** Đoạn code này tạo tiêu đề và các thành phần tương tác. `st.sidebar.form` tạo một form ở thanh bên, gom tất cả các widget nhập liệu lại. Một điểm mới là `st.sidebar.info`, nó sẽ hiển thị độ sâu tối ưu mà `GridSearchCV` đã tìm được, giúp người dùng biết mô hình đang chạy với cấu hình nào.

Xử lý, Dự đoán và Trực quan hóa

app.py - Phần 4

```

1 if submitted:
2     input_data = pd.DataFrame([{'age': age, 'sex': sex, 'cp': cp,
3                                 'trestbps': trestbps, 'chol': chol,
4                                 'fbs': fbs, 'restecg': restecg,
5                                 'thalach': thalach, 'exang': exang,
6                                 'oldpeak': oldpeak, 'slope':
7                                 slope, 'ca': ca, 'thal': thal}])
8
9     proba = model.predict_proba(input_data)[0]
10    prob_disease = proba[1]
11
12    st.subheader('Prediction Result')
13    if prob_disease > 0.5:
14        st.error(f'**Disease Likely** - Probability: {prob_disease:.2\%}')
15    else:
16        st.success(f'**Disease Unlikely** - Probability of No
17                    Disease: {(1-prob_disease):.2\%}')
18
19    st.subheader('Optimal Model Decision Tree Visualization')
20    st.write('This chart shows the decision rules of the best model
21            found via Cross-Validation.')
22
23    best_pipeline = model.best_estimator_
24    preprocessor = best_pipeline.named_steps['preprocessor']
25    classifier = best_pipeline.named_steps['classifier']
26
27    try:
28        feature_names = preprocessor.get_feature_names_out()
29        fig, ax = plt.subplots(figsize=(25, 12))
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
307
308
309
309
310
311
312
313
313
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
140
```

```

22     plot_tree(classifier, feature_names=feature_names, class_names=[ 
23         'No Disease', 'Disease'], filled= 
24         True, rounded=True, ax=ax, 
25         fontsize=10)
26     st.pyplot(fig)
27 except Exception as e:
28     st.warning(f'Could not display the tree visualization. Error: {e}')
29 
```

- Giải thích:** Khi người dùng nhấn “Predict”, code sẽ thu thập dữ liệu, tạo DataFrame và đưa vào mô hình để dự đoán xác suất. Kết quả được hiển thị trực quan. Phần cuối cùng trực quan hóa cây quyết định. Lưu ý rằng model lúc này là một đối tượng GridSearchCV, vì vậy chúng ta cần truy cập model.best_estimator_ để lấy ra pipeline tốt nhất. Từ đó, ta trích xuất bộ phân loại (classifier) và vẽ cây quyết định tương ứng với mô hình đã được tối ưu.

Bước 2: Cài đặt môi trường

Chúng ta cần cài đặt Streamlit và công cụ **Cloudflared** để tạo đường hầm.

Cài đặt Streamlit

```
1 !pip -q install streamlit
```

- Giải thích:** Lệnh này sử dụng pip để cài đặt thư viện streamlit.

Tải và cấp quyền cho Cloudflared

```

1 !wget -q https://github.com.cloudflare/cloudflared/releases/latest/
        download/cloudflared-linux-amd64 - 
        O cloudflared
2 !chmod +x cloudflared

```

- Giải thích:** Tải về file thực thi của Cloudflared và cấp quyền để có thể chạy nó.

Bước 3: Khởi chạy ứng dụng

Đây là bước cuối cùng để khởi động server và tạo link truy cập công khai.

Chạy Streamlit Server dưới nền

```
1 !streamlit run app.py \
2   --server.port 8501 \
3   --server.address 0.0.0.0 \
4   --server.headless true \
5   --server.enableCORS false \
6   --server.enableXsrfProtection false &>/content/logs.txt &
```

- **Giải thích:** Lệnh này khởi chạy ứng dụng app.py trên cổng 8501 và chạy ngầm dưới nền, cho phép chúng ta thực hiện lệnh tiếp theo.

Tạo đường hầm với Cloudflare

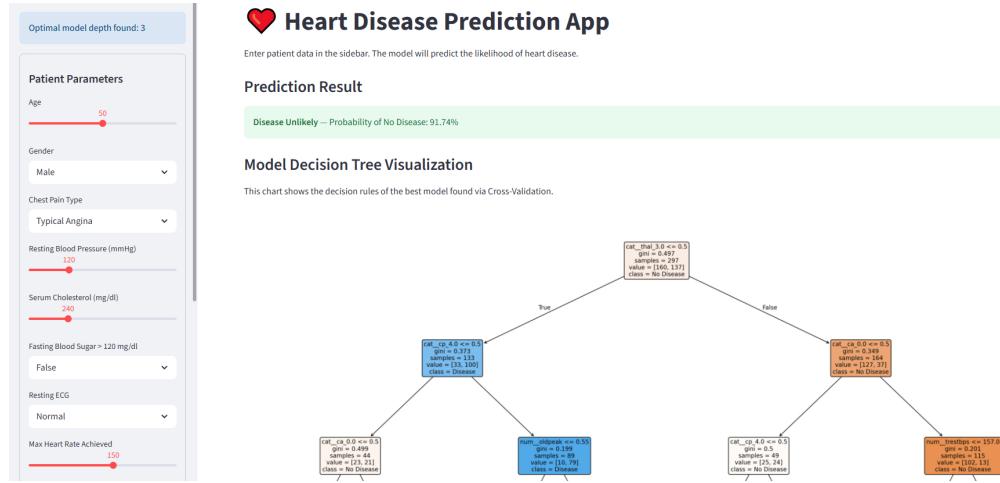
```
1 ./cloudflared tunnel --url http://localhost:8501 --no-autoupdate
```

- **Giải thích:** Lệnh này tạo một đường hầm công khai đến ứng dụng đang chạy cục bộ trên cổng 8501 của server Colab.

🔗 Kết quả và Cách truy cập

Sau khi chạy cell cuối cùng, bạn sẽ thấy một loạt các dòng output. Hãy tìm một đường link có dạng <https://...trycloudflare.com>. Đây chính là địa chỉ công khai của ứng dụng Streamlit của bạn.

Hãy nhập vào đường link để mở ứng dụng trong một tab mới!



Hình 20.15: Giao diện demo.

20.3 Câu hỏi trắc nghiệm

1. Xét đoạn code chia dữ liệu sau:

```

1 X_train, X_temp, y_train, y_temp = train_test_split(
2     X_all, y_all, test_size=0.2, stratify=y_all, random_state=
        42)

```

Tham số `stratify=y_all` có vai trò quan trọng nhất là gì trong bài toán này?

- (a) Đảm bảo các đặc trưng trong tập huấn luyện và kiểm thử có cùng phân phối.
 - (b) Giúp quá trình chia dữ liệu nhanh hơn bằng cách lấy mẫu có thứ tự.
 - (c) Đảm bảo tỷ lệ người có bệnh và không có bệnh được giữ nguyên trong các tập dữ liệu sau khi chia, tránh thiên vị do mất cân bằng nhãn.
 - (d) Xáo trộn dữ liệu ngẫu nhiên để loại bỏ sự phụ thuộc về thứ tự.
2. Cho các pipeline xử lý dữ liệu sau:

```

1 cat_proc = Pipeline(steps=[
2     ('imputer', SimpleImputer(strategy='most_frequent')),
3     ('scaler', MinMaxScaler())
4 ])
5 num_proc = Pipeline(steps=[
6     ('imputer', SimpleImputer(strategy='median')),
7     ('scaler', StandardScaler())
8 ])
9

```

Sau khi áp dụng `fit_transform`, một cột phân loại (ví dụ: `sex`) và một cột số (ví dụ: `age`) sẽ có đặc điểm gì?

- (a) Cột `sex` có giá trị trong khoảng [0, 1] và cột `age` có trung bình 0, độ lệch chuẩn 1.

- (b) Cả hai cột đều có trung bình 0 và độ lệch chuẩn 1.
- (c) Cột `sex` được điền bằng giá trị mode, cột `age` được điền bằng giá trị median, không có scaling.
- (d) Cả hai cột đều có giá trị trong khoảng [0, 1].
3. So sánh Hình 20.4 (Mutual Information) và Hình 20.5 (Decision Tree), sự khác biệt chính trong cách hai phương pháp đánh giá đặc trưng là gì?
- (a) Cả hai đều xếp hạng `thal`, `cp`, `ca` cao nhất, cho thấy chúng hoạt động theo cùng một nguyên lý.
- (b) Mutual Information đánh giá đặc trưng đã mã hóa one-hot (ví dụ: `thal_3.0`) dựa trên sự phụ thuộc thống kê với nhãn, trong khi Decision Tree đánh giá đặc trưng gốc (ví dụ: `thal`) dựa trên khả năng giảm impurity tại các điểm chia.
- (c) Decision Tree ưu tiên các đặc trưng số như `oldpeak` và `chol` hơn Mutual Information.
- (d) Mutual Information chỉ áp dụng được trên dữ liệu FE, còn Decision Tree dùng được cho cả dữ liệu gốc.
4. Hàm `find_optimal_rf` cố định `max_depth=5`. Việc giới hạn độ sâu cây trong khi tìm kiếm `n_estimators` có thể gây ra hệ quả tiềm tàng nào?

```

1 def find_optimal_rf(
2     # ...
3     max_depth=5, min_samples_split=2, min_samples_leaf=1,
4     max_features='sqrt', bootstrap=True, class_weight=None
5 ): # ...
6

```

- (a) Mô hình chắc chắn sẽ bị quá khớp (overfitting).
- (b) Giúp mô hình hội tụ nhanh và luôn tìm được cấu hình tốt nhất toàn cục.
- (c) Có thể khiến mô hình bị chưa khớp (underfitting) vì mỗi cây không đủ phức tạp để nắm bắt các mối quan hệ phức tạp, ngay cả khi có nhiều cây.

- (d) Tham số này không ảnh hưởng đến hiệu suất, chỉ có `n_estimators` là quan trọng.
5. Trong đoạn code XGBoost sau, tham số `tree_method` đang áp dụng kỹ thuật tối ưu hóa hiệu năng nào?

```

1 xgb = XGBClassifier(
2     # ...
3     tree_method='gpu_hist' if use_gpu else 'hist', verbosity=0
4 )
5

```

- (a) Kỹ thuật điều chỉnh L2 để kiểm soát trọng số của mô hình.
- (b) Thuật toán xây dựng cây dựa trên histogram giúp tăng tốc độ tìm điểm chia, và có tùy chọn tăng tốc bằng GPU.
- (c) Cơ chế tự động xử lý giá trị thiếu mà không cần imputation.
- (d) Kỹ thuật cắt tỉa cây (pruning) sau khi xây dựng hoàn chỉnh.
6. Trong code Streamlit, biến `model` được dùng theo hai cách. Tại sao cần truy cập `model.best_estimator_` để vẽ cây, trong khi có thể dùng `model` trực tiếp để dự đoán?

```

1 # Cách 1: Dự đoán
2 proba = model.predict_proba(input_data)[0]
3
4 # Cách 2: Lấy thông tin để vẽ cây
5 best_pipeline = model.best_estimator_
6 classifier = best_pipeline.named_steps['classifier']
7 plot_tree(classifier, ...)
8

```

- (a) Đây là quy ước đặt tên, `model` và `model.best_estimator_` thực chất là một.
- (b) Lệnh `plot_tree` yêu cầu một pipeline, còn `predict_proba` chỉ cần một mô hình phân loại.

- (c) Biến `model` là đối tượng `GridSearchCV`. Các hàm như `predict_proba` được ủy quyền cho mô hình tốt nhất, nhưng để truy cập các thành phần nội tại của pipeline (như `classifier`), ta phải tường minh lấy ra pipeline tốt nhất qua thuộc tính `.best_estimator_`.
- (d) Chỉ cần truy cập `.best_estimator_` trong lần chạy đầu tiên.
7. Trong Gradient Boosting, cơ chế “sửa sai” hoạt động bằng cách nào, và “sai số” được mô hình kế tiếp học dựa trên đại lượng gì?
- (a) Mô hình kế tiếp học trên các mẫu bị phân loại sai, với trọng số của các mẫu này được tăng lên.
 - (b) “Sai số” là phần chênh lệch giữa nhãn thực tế và dự đoán hiện tại, mô hình kế tiếp được huấn luyện để dự đoán chính các phần dư này.
 - (c) “Sai số” được tính bằng gradient bậc hai của hàm mất mát và dùng để cập nhật các nút lá.
 - (d) Mỗi mô hình mới được huấn luyện trên một mẫu bootstrap, và “sai số” được tổng hợp bằng cách lấy trung bình.
8. Đâu là sự khác biệt lý thuyết cốt lõi trong cơ chế “sửa sai” tuần tự giữa AdaBoost và Gradient Boosting được mô tả trong tài liệu?
- (a) Cả hai thuật toán đều tính toán phần dư (residual), nhưng AdaBoost chỉ áp dụng cho bài toán phân loại nhị phân.
 - (b) AdaBoost điều chỉnh trọng số của các mẫu huấn luyện, buộc mô hình sau tập trung vào các mẫu bị phân loại sai; trong khi Gradient Boosting huấn luyện mô hình sau để trực tiếp dự đoán phần dư của mô hình tổng hợp trước đó.
 - (c) AdaBoost luôn sử dụng các cây quyết định rất nông (stumps) làm mô hình yếu, trong khi Gradient Boosting phải sử dụng cây sâu hơn để tính toán phần dư.
 - (d) AdaBoost sử dụng learning rate để kết hợp các mô hình yếu, trong khi Gradient Boosting thì không.
9. Việc dùng `fit_transform` cho tập train nhưng chỉ `transform` cho tập val/test trong đoạn code sau nhằm tránh vấn đề cốt lõi nào?

```

1 X_raw_train = raw_pipeline.fit_transform(X_train, y_train)
2 X_raw_val = raw_pipeline.transform(X_val)
3 X_raw_test = raw_pipeline.transform(X_test)
4

```

- (a) Lỗi tương thích phiên bản của Scikit-learn.
- (b) Giảm thời gian tính toán vì `transform` nhanh hơn.
- (c) Rò rỉ dữ liệu, bằng cách đảm bảo các tham số (ví dụ: trung vị, tham số scaling) chỉ được học từ tập huấn luyện.
- (d) Ngăn ngừa overfitting bằng cách giảm số đặc trưng của tập val/test.
10. Lý do chính đằng sau việc tạo ra đặc trưng tương tác như `chol/age` là gì?

```

1 if {'chol', 'age'} <= set(df.columns):
2     df['chol_per_age'] = df['chol']/df['age']
3

```

- (a) Để chuẩn hóa giá trị cholesterol về cùng thang đo với tuổi.
- (b) Giả định rằng tác động của cholesterol lên nguy cơ bệnh tim có thể thay đổi theo độ tuổi (ví dụ, cholesterol cao ở người trẻ sẽ nguy hiểm hơn).
- (c) Đây là kỹ thuật bắt buộc để xử lý giá trị ngoại lai trong cột `chol` và `age`.
- (d) Để giảm số chiều dữ liệu bằng cách kết hợp hai đặc trưng.

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
 2. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
 3. **Demo:** Chương trình demo cho nội dung trong bài có thể được truy cập tại [đây](#).
 4. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#).
5. **Rubric:**

Mục	Kiến Thức	Đánh Giá
I.	<ul style="list-style-type: none"> - Hiểu bối cảnh bệnh tim mạch và tầm quan trọng của chẩn đoán sớm. - Nắm được thông tin về bộ dữ liệu Cleveland Heart Disease (303 mẫu, 13 đặc trưng). - Biết các đặc trưng y tế chính (age, sex, cp, trestbps, chol, thalach, oldpeak, slope, ca, thal, ...). 	<ul style="list-style-type: none"> - Trình bày được lý do chọn bộ dữ liệu Cleveland và cách nhị phân hóa nhãn mục tiêu. - Nhận biết và giải thích ý nghĩa các đặc trưng trong dữ liệu. - Hiểu mối liên hệ giữa dữ liệu y tế và mô hình học máy; nếu được cảnh báo phi y khoa.
II.	<ul style="list-style-type: none"> - Kiến thức về chọn đặc trưng bằng Decision Tree importance và Mutual Information. - Hiểu cấu trúc 4 biến thể dữ liệu: Original, FE, Original+DT(Top-K), FE+DT(Top-K). 	<ul style="list-style-type: none"> - Giải thích vì sao chọn K đặc trưng và tác động đến bias/variance. - Phân biệt cách xử lý biến phân loại (giữ nguyên, OHE, scaling). - Trình bày đúng pipeline fit/transform cho train/val/test, tránh data leakage.
III.	<ul style="list-style-type: none"> - Kiến thức về các mô hình ensemble nâng cao: Random Forest, AdaBoost, Gradient Boosting, XGBoost. - Kiến thức về baseline Deep Learning (MLP) trên dữ liệu bảng nhỏ. - Hiểu sự khác biệt Bagging vs Boosting và cơ chế học sửa sai. 	<ul style="list-style-type: none"> - Giải thích cơ chế của RF (giảm phương sai) và Boosting (giảm bias). - Nêu nguyên lý trọng số lỗi trong AdaBoost, residual trong Gradient Boosting. - Trình bày ưu/nhược điểm của XGBoost; khi nào nên dùng GPU/hist. - Nhận diện khi nào MLP hữu ích hoặc dễ overfit trên bộ nhỏ.

IV.	<ul style="list-style-type: none"> - Kiến thức triển khai bằng scikit-learn/xgboost: pipeline, GridSearchCV, cross-validation. - Biết sử dụng StratifiedKFold và chia train/val/test cố định để so sánh công bằng. - Biết đánh giá bằng Accuracy, Precision, Recall, F1 (ROC-AUC nếu có), và trực quan hóa. 	<ul style="list-style-type: none"> - Viết và hiểu các hàm <code>evaluate_val_*</code>, <code>evaluate_test_*</code>. - Giải thích mục đích của StratifiedKFold trong tìm tham số tối ưu. - Đọc và phân tích được chênh lệch val/test (overfit/underfit). - So sánh hiệu quả giữa các biến thể dữ liệu và mô hình khác nhau.
V.	<ul style="list-style-type: none"> - Kiến thức về tái lập thí nghiệm: cố định seed, quản lý dữ liệu nhất quán. - Nhận diện và tránh data leakage. - Biết tổ chức code: module hóa, ghi log, tách rõ phần build dữ liệu và train mô hình. 	<ul style="list-style-type: none"> - Cho thấy chạy lại với cùng seed cho kết quả tương đương. - Chỉ ra các điểm dễ gây rò rỉ dữ liệu và cách khắc phục. - Code rõ ràng, có chú thích; quản lý đường dẫn/thư mục đúng chuẩn.
VI.	<ul style="list-style-type: none"> - Kiến thức về xây dựng giao diện với Streamlit. - Biết cách deploy trên Colab thông qua cloudflare. - Biết đóng gói pipeline (preprocessing + model) để dự đoán trực tiếp từ input. 	<ul style="list-style-type: none"> - Ứng dụng chạy được: nhập tham số, trả dự đoán và xác suất. - Vẽ cây quyết định từ best estimator; hiển thị kết quả dễ hiểu. - Xử lý đúng vấn đề version (sklearn, OHE sparse_output).

6. **Đề xuất phương án cải tiến project:** Trong quá trình triển khai, project đã xây dựng được pipeline hoàn chỉnh từ xử lý dữ liệu, huấn luyện mô hình cho đến ứng dụng demo. Tuy nhiên, vẫn có nhiều hướng mở rộng và tinh chỉnh để nâng cao chất lượng kết quả cũng như khả năng ứng dụng. Các gợi ý dưới đây tập trung vào bốn khía cạnh chính: dữ liệu, mô hình, quản lý thí nghiệm và sản phẩm demo.

- **Cải thiện dữ liệu:**

- Thử nghiệm nhiều phương pháp chuẩn hóa khác nhau như: StandardScaler, MinMaxScaler, RobustScaler..., và so sánh ảnh hưởng đến hiệu suất.
- Sử dụng VarianceThreshold để loại bỏ đặc trưng ít biến thiên; áp dụng ma trận tương quan để loại bỏ đặc trưng trùng lặp.

– Tạo thêm đặc trưng mới từ dữ liệu sẵn có (feature engineering) tương tự như một cách làm ví dụ được trình bày ở nội dung bài viết phần 1.

– Úng dụng SelectKBest hoặc PolynomialFeatures để khai thác đặc trưng bậc cao.

- **Đa dạng hóa mô hình:**

– Bổ sung Logistic Regression để so sánh với các mô hình đã triển khai.

– Thử nhiều giá trị k trong KNN hoặc sử dụng weighted KNN để đánh giá độ ổn định.

– Áp dụng Support Vector Machine (SVM) với kernel tuyến tính hoặc RBF để kiểm chứng khả năng phân tách dữ liệu.

- **Quản lý thí nghiệm:**

– Đặt seed cố định để đảm bảo tính tái lập kết quả.

– Ghi chú và lưu lại cấu hình, tham số, kết quả nhằm thuận tiện cho việc so sánh.

– Sử dụng Optuna hoặc GridSearchCV mở rộng để tự động hóa quá trình tối ưu tham số.

- **Cải thiện sản phẩm demo:**

– Thiết kế giao diện nhập liệu trực quan hơn: dùng `st.slider` cho biến số (tuổi, cholesterol) và `st.selectbox` cho biến phân loại (giới tính, loại đau ngực).

– Hiển thị xác suất dự đoán bên cạnh nhãn phân loại để làm rõ mức độ rủi ro.

– Bổ sung phần giải thích đặc trưng quan trọng (feature importance) từ mô hình.

– Lưu lại lịch sử dự đoán trong ứng dụng hoặc triển khai bằng Docker/Streamlit Cloud để tăng tính khả chuyen.

7. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại [đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 tiếng.

AIO_QAs-Verified

🔒 Private group · 1.4K members



Hình 20.16: Hình ảnh group facebook AIO Q&A.

Phần V

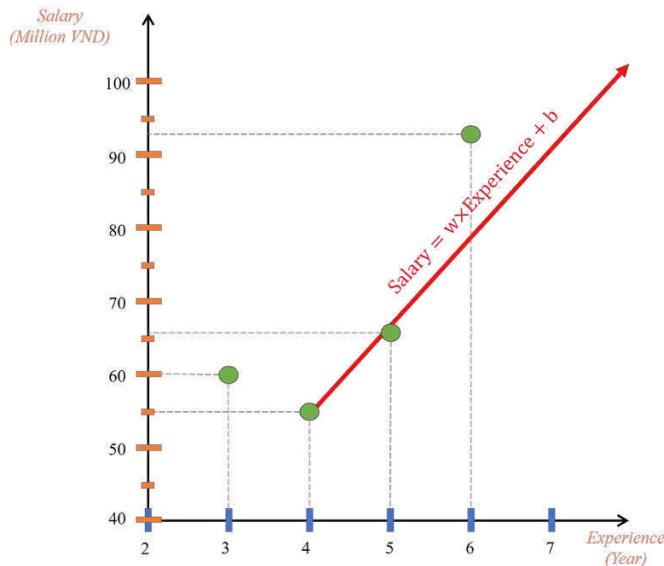
Module 5: Nền tảng cho deep learning (1)

Chương 21

Linear Regression (trình bày theo cách cơ bản)

21.1 Giới thiệu

Linear Regression (hồi quy tuyến tính) là phương pháp cổ điển bắt nguồn từ phương pháp “bình phương tối thiểu” do *Legendre* (1805) và *Gauss* (1809) phát triển; thuật ngữ *regression* được *Francis Galton* (1886) đặt khi mô tả hiện tượng “hồi quy về trung bình” trong nghiên cứu di truyền.



Hình 21.1: Mô hình hồi quy tuyến tính (Linear Regression) dự đoán mức lương dựa vào kinh nghiệm

Trong phần thực hành hôm nay, chúng ta sẽ tiếp cận Linear Regression thông qua một chuỗi bài tập code, đi từ những bước cơ bản nhất đến hoàn chỉnh mô hình. Cụ thể, ta sẽ bắt đầu bằng việc đọc và tiền xử lý dữ liệu, sau đó lần lượt xây dựng các thành phần của mô hình: hàm dự đoán, hàm mất mát, gradient, và quy tắc cập nhật tham số. Tiếp theo, mô hình sẽ được huấn luyện với nhiều tốc độ học khác nhau, trực quan hóa đường cong loss, và

cuối cùng đánh giá kết quả bằng các chỉ số quen thuộc như MSE, RMSE và R^2 .

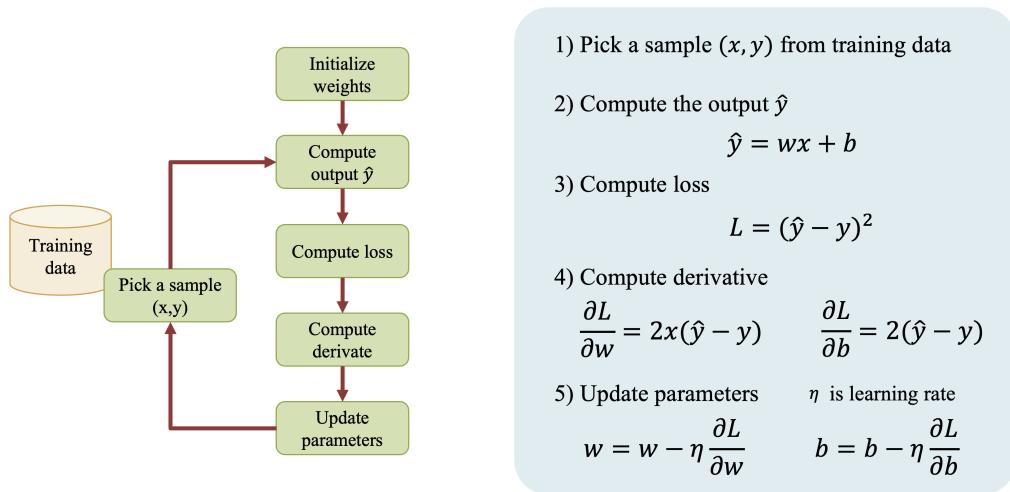
Dữ liệu sử dụng trong bài tập là bộ advertising.csv gồm 200 mẫu (rows), với bốn thông tin: chi phí quảng cáo trên TV, Radio, Newspaper, và Sales (doanh số bán hàng).

index	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9
5	8.7	48.9	75.0	7.2
6	57.5	32.8	23.5	11.8
7	120.2	19.6	11.6	13.2
8	8.6	2.1	1.0	4.8

Hình 21.2: Một vài sample data từ dữ liệu quảng cáo advertising.csv

21.2 Các bước trong Linear Regression cho bài toán hồi quy

Các bước huấn luyện dưới đây được trình bày minh họa cho trường hợp dữ liệu có 3 thuộc tính đầu vào (TV, Radio, Newspaper) và Linear Regression theo từng mẫu (Stochastic Gradient Descent)



Hình 21.3: Quá trình huấn luyện mô hình hồi quy tuyến tính

Step 0: Khởi tạo trọng số

Để huấn luyện mô hình chúng ta khởi tạo các trọng số: w và b .

Step 1: Lấy một sample

Chọn một mẫu (x_1, x_2, x_3, y) từ tập huấn luyện.

Step 2: Tính output dự đoán

$$\hat{y} = w_1x_1 + w_2x_2 + w_3x_3 + b.$$

Step 3: Tính hàm mất mát

$$L = (\hat{y} - y)^2.$$

Step 4: Tính đạo hàm (gradient)

$$\frac{\partial L}{\partial w_1} = 2x_1(\hat{y} - y), \quad \frac{\partial L}{\partial w_2} = 2x_2(\hat{y} - y), \quad \frac{\partial L}{\partial w_3} = 2x_3(\hat{y} - y), \quad \frac{\partial L}{\partial b} = 2(\hat{y} - y).$$

Step 5: Cập nhật tham số

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}, \quad w_2 \leftarrow w_2 - \eta \frac{\partial L}{\partial w_2}, \quad w_3 \leftarrow w_3 - \eta \frac{\partial L}{\partial w_3}, \quad b \leftarrow b - \eta \frac{\partial L}{\partial b}.$$

Sau khi cập nhật, quay lại **Step 1** và tiếp tục lặp qua các mẫu cho đến khi hội tụ hoặc đạt số vòng lặp tối đa.

21.3 Chuẩn bị dữ liệu

Bài tập về kỹ thuật đọc và xử lý dữ liệu từ file.csv: biết file dữ liệu advertising.csv đã được tải ở code bên dưới, hãy hoàn thành function **prepare_data(file_name_dataset)** để trả về dữ liệu đã được tổ chức (X cho input và y cho output).

```

1 !gdown 1Z3bs8lnk6os1n5RE7x_p75JR-1yk1sf1
2 !unzip advertising.zip
3
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import random
7
8 def get_column(data, index):
9
10     ****your code here***
11
12     return result
13
14 def prepare_data(file_name_dataset):

```

```

15 data = np.genfromtxt(file_name_dataset, delimiter=',', skip_header
16 =1).tolist()
17 N = len(data)
18 # get tv (index=0)
19 tv_data = get_column(data, 0)
20
21 # get radio (index=1)
22 radio_data = get_column(data, 1)
23
24 # get newspaper (index=2)
25 newspaper_data = get_column(data, 2)
26
27 # get sales (index=3)
28 sales_data = get_column(data, 3)
29
30 # building X input and y output for training
31 X = [tv_data, radio_data, newspaper_data]
32 y = sales_data
33 return X,y

```

☰ Yêu cầu bài tập

```

1 X,y = prepare_data('..../content/advertising.csv')
2 list = [sum(X[0][:5]), sum(X[1][:5]), sum(X[2][:5]), sum(y[:5])
3 print(list)

```

Chạy code trên, phân tích kết quả thu được và lựa chọn đáp án đúng trong các phương án sau:

- a) [624.1, 175.1, 300.5, 78.9]
- b) [625, 175.1, 75.0, 7.2]
- c) [626, 175.1, 75.0, 7.2]
- d) [627.8, 175.1, 75.0, 7.2]

21.4 Huấn luyện mô hình

Bài tập về kỹ thuật huấn luyện data dùng one sample – linear regression: sử dụng kết quả dữ liệu đầu vào X , và dữ liệu đầu ra y từ Bài tập 1, để phát triển chương trình dự đoán thông tin **sales** (y) từ X bằng cách dùng giải thuật linear regression với one-sample training.

Loss được tính bằng công thức Mean Squared Error:

$$L = (\hat{y} - y)^2$$

Vì để thống nhất việc đánh giá kết quả cho toàn bài tập trong module này, các bạn được yêu cầu khởi tạo cỗ định wi, cũng như b như sau (trong thực tế bạn nên sử dụng hàm random):

```

1 def initialize_params():
2     # w1 = random.gauss(mu=0.0, sigma=0.01)
3     # w2 = random.gauss(mu=0.0, sigma=0.01)
4     # w3 = random.gauss(mu=0.0, sigma=0.01)
5     # b = 0
6
7     w1, w2, w3, b = (0.016992259082509283, 0.0070783670518262355, -0
8                               .002307860847821344, 0)
9
10    return w1, w2, w3, b

```

Hoàn thành function **predict(x1, x2, x3, w1, w2, w3, b)** để trả về kết quả dự đoán y tương ứng

```

1 def predict(x1, x2, x3, w1, w2, w3, b):
2
3     ****your code here****
4
5     return result

```

☰ Yêu cầu bài tập

```
1 y = predict(x1=1, x2=1, x3=1, w1=0, w2=0.5, w3=0, b=0.5)
2 print(y)
```

Chạy đoạn code trên, phân tích kết quả thu được và lựa chọn đáp án đúng trong các phương án sau:

- a) 1.0
- b) 2.0
- c) 3.0
- d) 4.0

Hoàn thành function **compute_loss(y_hat, y)** để tính loss giữa kết quả dự đoán **y_hat** và giá trị thực **y**, sử dụng Mean Squared Error.

```
1 def compute_loss(y_hat, y):
2
3     ****your code here****
4
5     return result
```

☰ Yêu cầu bài tập

```

1 l = compute_loss(y_hat=1, y=0.5)
2 print(l)

```

Chạy đoạn code trên, phân tích kết quả thu được và lựa chọn đáp án đúng trong các phương án sau:

- a) 0.25
- b) 0.26
- c) 0.27
- d) 0.28

Hoàn thành function **compute_gradient_wi(xi, y, y_hat)** để tính đạo hàm của hàm loss

$$L = (\hat{y} - y)^2$$

theo w_i và function **compute_gradient_b(y, y_hat)** để tính đạo hàm của hàm loss L theo b .

```

1 def compute_gradient_wi(xi, y, y_hat):
2
3     ***your code here***
4
5     return dl_dwi
6
7 def compute_gradient_b(y, y_hat):
8
9     ***your code here***
10
11    return dl_db

```

☰ Yêu cầu bài tập

```
1 g_wi = compute_gradient_wi(xi=1.0, y=1.0, y_hat=0.5)
2 print(g_wi)
```

Chạy đoạn code trên, phân tích kết quả thu được và lựa chọn đáp án đúng trong các phương án sau:

- a) -1.0
- b) -2.0
- c) 1.0
- d) 2.0

☰ Yêu cầu bài tập

```
1 g_b = compute_gradient_b(y=2.0, y_hat=0.5)
2 print(g_b)
```

Chạy đoạn code trên, phân tích kết quả thu được và lựa chọn đáp án đúng trong các phương án sau:

- a) -1.0
- b) -3.0
- c) 1.0
- d) 2.0

Hoàn thành function **update_weight_wi(wi, dl_dwi, lr)** để cập nhật w_i sau khi tính đạo hàm loss L theo w_i , và function **update_weight_b(b, dl_db, lr)** để update bias (b) sau khi tính đạo hàm loss L theo b .

```
1 def update_weight_wi(wi, dl_dwi, lr):
2
3     ****your code here***
4
5     return wi
6
7 def update_weight_b(b, dl_db, lr):
8
9     ****your code here***
10
11    return b
```

☰ Yêu cầu bài tập

```
1 after_wi = update_weight_wi(wi=1.0, dl_dwi=-0.5, lr=1e-5)
2 print(after_wi)
```

Chạy đoạn code trên, phân tích kết quả thu được và lựa chọn đáp án đúng trong các phương án sau:

- a) 1.000005
- b) -3.0
- c) 1.0
- d) 2.0

☰ Yêu cầu bài tập

```

1 after_b = update_weight_b(b=0.5, dl_db=-1.0, lr=1e-5)
2 print(after_b)

```

Chạy đoạn code trên, phân tích kết quả thu được và lựa chọn đáp án đúng trong các phương án sau:

- a) 0.50001
- b) -3.0
- c) 1.0
- d) 2.0

Sau khi hoàn thành các function trên ta sẽ tạo được function **implement_linear_regression(X_data, y_data, epoch_max, lr)** và trả về 4 tham số w_1, w_2, w_3, b cùng lịch sử tính loss như bên dưới:

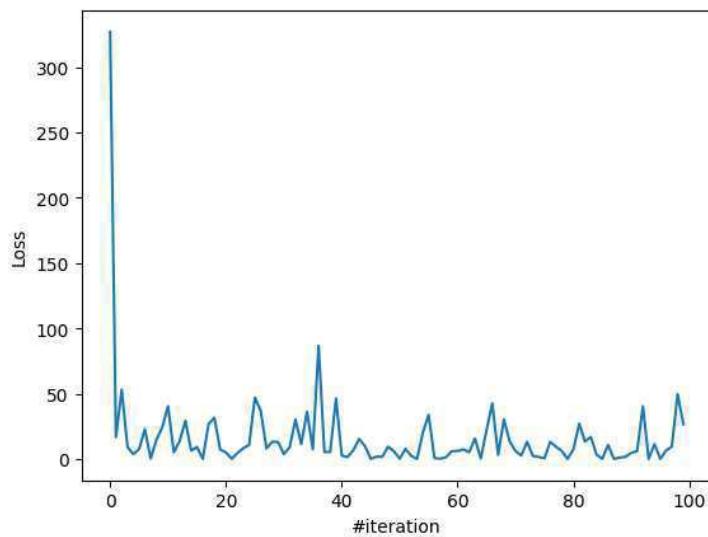
```

1 def implement_linear_regression(X_data, y_data, epoch_max = 50, lr =
1e-5):
2     losses = []
3
4     w1, w2, w3, b = initialize_params()
5
6     N = len(y_data)
7     for epoch in range(epoch_max):
8         for i in range(N):
9             # get a sample
10            x1 = X_data[0][i]
11            x2 = X_data[1][i]
12            x3 = X_data[2][i]
13
14            y = y_data[i]
15
16            # print(y)
17            # compute output
18            y_hat = predict(x1, x2, x3, w1, w2, w3, b)

```

```
19
20     # compute loss
21     loss = compute_loss(y, y_hat)
22
23     # compute gradient w1, w2, w3, b
24     dl_dw1 = compute_gradient_wi(x1, y, y_hat)
25     dl_dw2 = compute_gradient_wi(x2, y, y_hat)
26     dl_dw3 = compute_gradient_wi(x3, y, y_hat)
27     dl_db = compute_gradient_b(y, y_hat)
28
29     # update parameters
30     w1 = update_weight_wi(w1, dl_dw1, lr)
31     w2 = update_weight_wi(w2, dl_dw2, lr)
32     w3 = update_weight_wi(w3, dl_dw3, lr)
33     b = update_weight_b(b, dl_db, lr)
34
35     # logging
36     losses.append(loss)
37 return (w1,w2,w3,b, losses)
```

```
1 (w1,w2,w3,b, losses) = implement_linear_regression(X,y)
2 print(losses[0:100])
3 plt.plot(losses[0:100])
4 plt.xlabel("#iteration")
5 plt.ylabel("Loss")
6 plt.show()
```



Hình 21.4: Kết quả loss sau 100 iteration cập nhật.

☰ Yêu cầu bài tập

```
1 X,y = prepare_data('..../content/advertising.csv')
2
3 (w1, w2, w3, b, losses) = implement_linear_regression(X, y)
4 print(w1, w2, w3)
```

Chạy đoạn code trên, phân tích kết quả thu được và lựa chọn đáp án đúng trong các phương án sau:

- a) $w1 = 0.074, w2 = 0.15, w3 = 0.17$
- b) $w1 = 1.074, w2 = 1.15, w3 = 1.17$
- c) $w1 = 2.074, w2 = 0.15, w3 = 2.17$
- d) $w1 = 3.074, w2 = 0.15, w3 = 3.17$

☰ Yêu cầu bài tập

```
1 # given new data
2 tv = 19.2
3 radio = 35.9
4 newspaper = 51.3
5
6 X,y = prepare_data('..../content/advertising.csv')
7 (w1, w2, w3, b, losses) = implement_linear_regression(X, y,
               epoch_max=50, lr=1e-5)
8 sales = predict(tv, radio, newspaper, w1, w2, w3, b)
9 print(f'predicted sales is {sales}')
```

Chạy đoạn code trên, phân tích kết quả thu được và lựa chọn đáp án đúng trong các phương án sau:

- a) predicted sales is 6.18
- b) predicted sales is 8.18
- c) predicted sales is 7.18
- d) predicted sales is 9.18

21.5 Chuẩn hoá đặc trưng

Trong phần này, chúng ta sẽ được tìm hiểu thêm về hai phương pháp chuẩn hoá dữ liệu phổ biến: **Min-Max Scaling** và **Z-score Scaling (Standardization)**. Ngoài ra, chúng ta cũng sẽ giới thiệu cách thực hiện **inverse transform** để đưa dữ liệu đã chuẩn hoá trở về giá trị ban đầu. Các ví dụ minh họa sẽ giúp làm rõ cơ chế hoạt động của từng phương pháp.

Min-Max Scaling

Phương pháp này đưa dữ liệu về khoảng $[0, 1]$ theo công thức:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

trong đó x_{\min} và x_{\max} lần lượt là giá trị nhỏ nhất và lớn nhất của thuộc tính.

Inverse Min-Max Scaling

Để khôi phục dữ liệu gốc từ dữ liệu đã chuẩn hoá, ta sử dụng:

$$x = x' \times (x_{\max} - x_{\min}) + x_{\min}$$

Min-Max Scaling & Inverse Min-Max Scaling

```

1 # -----
2 # 1. Min-Max Scaling
3 #
4 def min_max_scaling(X):
5     mins = []
6     maxs = []
7     X_scaled = []
8
9     # Tìm min và max cho từng feature
10    for feature in X:
11        min_val = min(feature)
12        max_val = max(feature)
13        mins.append(min_val)
14        maxs.append(max_val)
15
16        # Scale từng phần tử

```

```

17     scaled_feature = []
18     for x in feature:
19         scaled_x = (x - min_val) / (max_val - min_val)
20         scaled_feature.append(scaled_x)
21
22     X_scaled.append(scaled_feature)
23
24 return X_scaled, mins, maxs
25
26
27 # -----
28 # 2. Inverse Min-Max
29 # -----
30 def inverse_min_max_scaling(X_scaled, mins, maxs):
31     X_recovered = []
32
33     for i in range(len(X_scaled)):
34         feature = X_scaled[i]
35         min_val = mins[i]
36         max_val = maxs[i]
37
38         recovered_feature = []
39         for x in feature:
40             original_x = x * (max_val - min_val) + min_val
41             recovered_feature.append(original_x)
42
43     X_recovered.append(recovered_feature)
44
45 return X_recovered

```

Z-score Scaling (Standardization)

Phương pháp này chuẩn hoá dữ liệu về phân phối có trung bình bằng 0 và độ lệch chuẩn bằng 1:

$$x' = \frac{x - \mu}{\sigma}$$

trong đó μ là giá trị trung bình và σ là độ lệch chuẩn.

Inverse Z-score Scaling

Để đưa dữ liệu đã chuẩn hoá về ban đầu, ta áp dụng:

$$x = x' \times \sigma + \mu$$

Z-score Scaling (Standardization) & Inverse Z-score Scaling

```

1 # -----
2 # 3. Z-Score Scaling
3 # -----
4 def z_score_scaling(X):
5     means = []
6     stds = []
7     X_scaled = []
8
9     for feature in X:
10         mean_val = sum(feature) / len(feature)
11         means.append(mean_val)
12
13         std_val = ((sum((x - mean_val) ** 2 for x in feature)) / (len(
14             feature) - 1)) ** 0.5
15         stds.append(std_val)
16
17         scaled_feature = []
18         for x in feature:
19             scaled_x = (x - mean_val) / std_val
20             scaled_feature.append(scaled_x)
21
22         X_scaled.append(scaled_feature)
23
24
25
26 # -----
27 # 4. Inverse Z-Score
28 # -----
29 def inverse_z_score_scaling(X_scaled, means, stds):
30     X_recovered = []
31
32     for i in range(len(X_scaled)):
33         feature = X_scaled[i]
34         mean_val = means[i]
35         std_val = stds[i]

```

```
36     recovered_feature = []
37     for x in feature:
38         original_x = x * std_val + mean_val
39         recovered_feature.append(original_x)
40
41     X_recovered.append(recovered_feature)
42
43
44 return X_recovered
```

Ví dụ áp dụng

```
1 data1 = [1, 2, 3]
2 data2 = [4, 5, 6]
3 data3 = [7, 8, 9]
4 X = [data1, data2, data3]
5
6 # Min-Max
7 X_minmax, mins, maxs = min_max_scaling(X)
8 print("Min-Max Scaled:", X_minmax)
9 print("Recovered Min-Max:", inverse_min_max_scaling(X_minmax, mins,
10                                         maxs))
11
12 # Z-Score
13 X_zscore, means, stds = z_score_scaling(X)
14 print("\nZ-Score Scaled:", X_zscore)
15 print("Recovered Z-Score:", inverse_z_score_scaling(X_zscore, means,
16                                         stds))
```

21.6 Câu hỏi trắc nghiệm

1. Với $w = 2.5$, $b = 0.5$, $x = 1.2$, giá trị \hat{y} là?

Công thức: $\hat{y} = wx + b$

- (a) 3.2
- (b) 3.5
- (c) 3.0
- (d) 2.8

2. Với $w = 1.5$, $b = 0$, $x = 2$, $y = 4$, giá trị hàm mất mát L là?

Công thức: $L = (\hat{y} - y)^2$, $\hat{y} = wx + b$

- (a) 0.5
- (b) 1
- (c) 2
- (d) 4

3. Cho $x = 3$, $w = 0.4$, $b = 0.2$, $y = 1$. Gradient theo w là?

Công thức: $\frac{\partial L}{\partial w} = 2x(\hat{y} - y)$, $\hat{y} = wx + b$

- (a) 2.4
- (b) 1.2
- (c) 0.8
- (d) 0.4

4. Cập nhật w với $\alpha = 0.05$ cho mẫu ở câu trên. Giá trị w mới là?

Công thức: $w \leftarrow w - \alpha \frac{\partial L}{\partial w}$ với $\frac{\partial L}{\partial w} = 2x(\hat{y} - y)$

- (a) 0.28
- (b) 0.52
- (c) 0.36
- (d) 0.32

5. Với $w_1 = 0.5$, $w_2 = -1.0$, $b = 2.0$, $x_1 = 4$, $x_2 = 1.5$, giá trị \hat{y} là?

Công thức: $\hat{y} = w_1x_1 + w_2x_2 + b$

- (a) 2.5
- (b) 3.0
- (c) 1.5
- (d) 0.5

6. Với $x_1 = 2$, $x_2 = -1$, $w_1 = 1$, $w_2 = 0.5$, $b = 0$, $y = 1$, gradient theo w_1 là?

Công thức: $\frac{\partial L}{\partial w_1} = 2x_1(\hat{y} - y)$, $\hat{y} = w_1x_1 + w_2x_2 + b$

- (a) 1.0
- (b) 2.0
- (c) 0.5
- (d) -1.0

7. Với $x_1 = 2$, $x_2 = 3$, $w_1 = 0.5$, $w_2 = -0.5$, $b = 1$, $y = 0$. Gradient theo w_2 là?

Công thức: $\frac{\partial L}{\partial w_2} = 2x_2(\hat{y} - y)$, $\hat{y} = w_1x_1 + w_2x_2 + b$

- (a) 3
- (b) 1.5
- (c) -3
- (d) 0.5

8. Tiếp theo câu trên, cập nhật b với $\alpha = 0.05$. Giá trị b mới là?

Công thức: $b \leftarrow b - \alpha \frac{\partial L}{\partial b}$, $\frac{\partial L}{\partial b} = 2(\hat{y} - y)$

- (a) 0.95
- (b) 0.975
- (c) 1.05
- (d) 1.10

9. Chuẩn hoá z-score với $x = 150$, $\mu = 120$, $\sigma = 15$. Giá trị sau khi scale là?

Công thức:
$$z = \frac{x - \mu}{\sigma}$$

- (a) 1.5
 - (b) 2.0
 - (c) 0.5
 - (d) -2.0
10. Min–max scaling (làm tròn 2 chữ số thập phân) với $x = 40$, $x_{\min} = 20$, $x_{\max} = 80$. Giá trị sau khi scale là?

Công thức:
$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

- (a) 0.25
- (b) 0.33
- (c) 0.50
- (d) 0.66

1. **Hint:** Các file code gợi ý và dữ liệu nếu có được lưu trong thư mục có thể được tải [tại đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại [đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 giờ.

AIO_QAs-Verified

• Nhóm Riêng tư · 1,4K thành viên



Hình 21.5: Hình ảnh group facebook AIO Q&A

4. Rubric:

Module 4 - Exercise - Linear Regression - Rubric		
Câu	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Khái niệm linear regression và các biến thể của thuật toán Gradient Descent (Stochastic Gradient Descent, Mini Batch Gradient Descent, và Batch Gradient Descent) - Cách thức thực hiện 3 biến thể trên của Gradient Descent theo kiểu Vectorization (với sự hỗ trợ của thư viện numpy) - Cách đọc và xử lý data cơ bản với numpy 	<ul style="list-style-type: none"> - Hiểu và nắm được ý chính cách hoạt động của 3 biến thể của thuật toán Gradient Descent (Stochastic Gradient Descent, Mini Batch Gradient Descent, và Batch Gradient Descent) với Linear Regression - Vận dụng và thao tác với thư viện numpy để thực hiện 3 biến thể trên theo kiểu Vectorization - Biết cách load data kiểu csv và normalize các feature trước khi đi train linear regression model
2	<ul style="list-style-type: none"> - Hiểu về cách áp dụng của thuật toán Linear Regression cho dữ liệu Time Series - Phân tích một số đặc trưng của dữ liệu Time Series 	<ul style="list-style-type: none"> - Áp dụng Linear Regression cho bài toán dự đoán giá Bitcoin - Đánh giá hiệu quả trên những cài đặt khác nhau

Chương 22

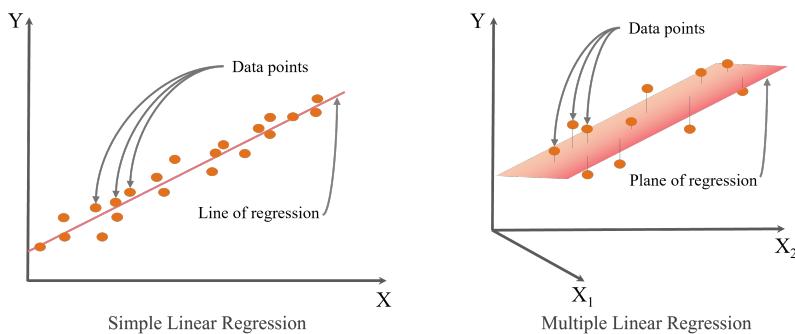
Linear Regression (ứng dụng Numpy và Đại số tuyến tính)

22.1 Giới thiệu

Ban mở một quán cà phê nhỏ và nhận ra: những ngày nắng nóng, số ly cold brew bán ra tăng vọt; trời mát thì khách chuộng latte nóng và doanh thu giảm nhẹ. Bạn bắt đầu ghi lại dữ liệu mỗi ngày: nhiệt độ trung bình, số ly bán ra, doanh thu. Sau vài tuần, bạn tự hỏi: "Nếu mai nhiệt độ 34°C , mình sẽ bán khoảng bao nhiêu ly?"

Đó chính là lúc các thuật toán bước vào đời sống. Khi một đại lượng (doanh số) thay đổi theo một đại lượng khác (nhiệt độ), ta tìm một đường thẳng 'phù hợp nhất' đi qua các điểm dữ liệu để mô tả mối quan hệ ấy. Với *linear regression* (hồi quy tuyến tính), đường thẳng $y = wx + b$ cho ta hai ý nghĩa:

- **w:** mỗi khi nhiệt độ tăng 1°C , doanh số ước tính tăng/giảm bao nhiêu.
- **b:** doanh số kỳ vọng khi nhiệt độ bằng 0.



Hình 22.1: Hình ảnh minh họa về Linear Regression.

Từ đường thẳng này, ta có thể dự đoán tương lai (forecast), giải thích tác động (diễn giải hệ số), và ra quyết định (nhập hàng, bố trí nhân sự). Tất nhiên, không phải lúc nào mọi thứ cũng "tuyến tính": có nhiều, ngoại lệ, và

nhiều yếu tố khác (thứ trong tuần, khuyến mãi). Nhưng bắt đầu bằng hồi quy tuyến tính giúp ta có một mô hình đủ đơn giản, minh bạch để giải quyết các vấn đề thường gặp.

Trong phần tiếp theo chúng ta sẽ tìm hiểu về các phương pháp tối ưu cho huấn luyện các mô hình hồi quy tuyến tính.

Để việc đọc thuận tiện hơn, bảng sau liệt kê và chuẩn hoá các ký hiệu sẽ xuất hiện xuyên suốt nội dung bài.

Bảng Ký hiệu Toán học	
Ký hiệu	Ý nghĩa
x_i	Đặc trưng (đầu vào) của mẫu thứ i (trường hợp 1 biến).
y_i	Nhân (giá trị mục tiêu) của mẫu i .
\hat{y}_i	Giá trị dự đoán từ x_i .
w, b	Trọng số và bias của mô hình.
ε_i	Sai số ngẫu nhiên (nhiều) giữa y_i và \hat{y}_i .
d	Số đặc trưng.
m	Kích thước batch (số mẫu trong một batch).
$\mathbb{R}^{a \times b}$	Không gian các ma trận số thực kích thước $a \times b$ (a hàng, b cột).
$\mathbf{x} \in \mathbb{R}^{(d+1)}$	Vector đặc trưng đã gộp bias cho <i>một</i> mẫu.
$\boldsymbol{\theta} \in \mathbb{R}^{(d+1)}$	Vector tham số (gồm các w_j và b).
$\mathbf{X} \in \mathbb{R}^{m \times (d+1)}$	Ma trận đặc trưng của một batch m mẫu (đã gộp bias).
$\mathbf{y} \in \mathbb{R}^m$	Vector mục tiêu của batch.
L	Hàm mất mát: thước đo độ sai lệch giữa dự đoán và thực tế.
$\frac{\partial L_i}{\partial w}, \frac{\partial L_i}{\partial b}$	Thành phần gradient theo từng tham số tại mẫu i .
$\nabla_{\boldsymbol{\theta}} L$	Gradient: xác định hướng và tốc độ dốc nhất để giảm lỗi.
η	Tốc độ học (learning rate).

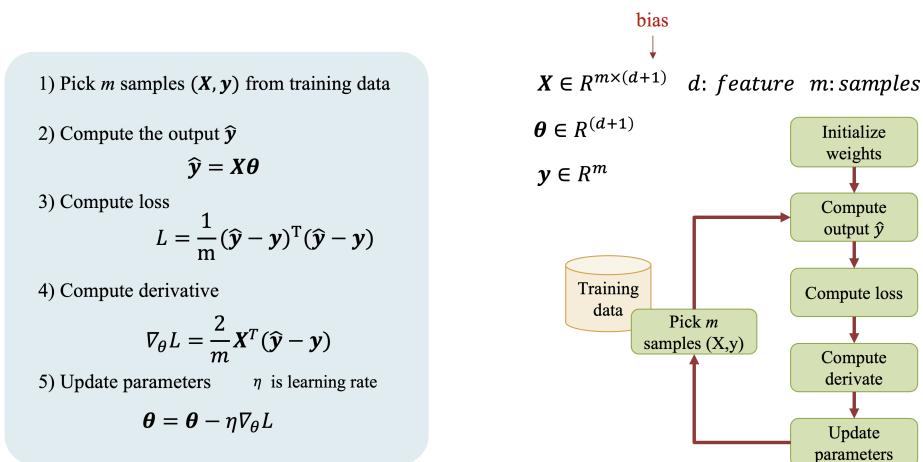
22.2 Mô hình Linear Regression

22.2.1 Tổng quan

Linear Regression (hồi quy tuyến tính) là phương pháp cổ điển bắt nguồn từ phương pháp “bình phương tối thiểu” do *Legendre* (1805) và *Gauss* (1809) phát triển; thuật ngữ *regression* được *Francis Galton* (1886) đặt khi mô tả hiện tượng “hồi quy về trung bình” trong nghiên cứu di truyền.

Mục tiêu của Linear Regression là mô hình hoá quan hệ gần tuyến tính giữa đầu vào và đầu ra để dự đoán hoặc giải thích ảnh hưởng của từng biến đầu vào lên kết quả.

Trong bài toán có d đặc trưng x và nhãn y áp dụng với từng mẫu dữ liệu, mô hình:



Hình 22.2: Hình ảnh minh họa về quá trình huấn luyện Linear Regression với m samples.

22.2.2 Linear Regression áp dụng từng mẫu dữ liệu

Step 1: Khởi tạo

Khởi tạo tham số đăt gộp bias:

$$\boldsymbol{\theta} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \\ b \end{bmatrix} \in \mathbb{R}^{(d+1)}, \quad \boldsymbol{\theta} \sim \mathcal{N}(0, 0.01^2 I_{d+1}).$$

Chú ý: Chúng ta hoàn toàn có thể đặt biến b đứng trước w_1 trong vector $\boldsymbol{\theta}$, thay vì đứng sau w_d như trên. Lúc này, vector x ở bước sau cũng cần thay đổi vị trí số 1 tương ứng.

Step 2: Lấy một mẫu

Chọn một cặp (x, y) với

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \\ 1 \end{bmatrix} \in \mathbb{R}^{(d+1)}, \quad y \in \mathbb{R}.$$

Step 3: Dự đoán (vector hoá)

$$\hat{y} = \mathbf{x}^T \boldsymbol{\theta} = \boldsymbol{\theta}^T \mathbf{x} \in \mathbb{R}.$$

Step 4: Mất mát từng mẫu (MSE)

$$L = (\hat{y} - y)^2.$$

Step 5: Gradient theo vector tham số

$$\nabla_{\boldsymbol{\theta}} L = 2\mathbf{x}(\hat{y} - y) \in \mathbb{R}^{(d+1)}.$$

Step 6: Cập nhật SGD (dùng 1 mẫu mỗi bước, learning rate α)

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L$$

Lặp lại từ **Step 2** qua các mẫu cho đến khi hội tụ hoặc đạt số vòng lặp tối đa.

22.2.3 Linear Regression cho batch (mini-batch hoặc full batch)

Step 1: Khởi tạo

Khởi tạo tham số đăt gộp bias:

$$\boldsymbol{\theta} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \\ b \end{bmatrix} \in \mathbb{R}^{(d+1)}, \quad \boldsymbol{\theta} \sim \mathcal{N}(0, 0.01^2 I_{d+1}).$$

Step 2: Lấy một batch

Chọn batch m mẫu $(X^{(i)}, y^{(i)})$ với $i = 1, \dots, m$ và ghép thành:

$$\mathbf{X} = \begin{bmatrix} X^{(1)} \\ \vdots \\ X^{(m)} \end{bmatrix} \in \mathbb{R}^{m \times (d+1)}, \quad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m,$$

trong đó mỗi $X^{(i)} \in \mathbb{R}^{1 \times (d+1)}$ là một vector hàng.

Step 3: Dự đoán cho batch

$$\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\theta} \in \mathbb{R}^m.$$

Step 4: Mát mát cho batch

$$L = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2.$$

Step 5: Gradient cho batch

$$\nabla_{\theta} L = \frac{2}{m} \mathbf{X}^{\top} (\hat{\mathbf{y}} - \mathbf{y}) \in \mathbb{R}^{(d+1)}.$$

Step 6: Cập nhật tham số

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\theta} L.$$

Lặp lại từ **Step 2** đến khi hội tụ hoặc đạt số epoch tối đa.

22.3 Xây dựng mô hình Linear Regression cho bài toán advertising

Ở bài toán này chúng ta sẽ thực hành các bài tập code về giải thuật linear regression cho bài toán advertising theo cách vectorization dùng stochastic gradient descent, m samples (mini-batch gradient descent), and N samples (batch gradient descent).

Thực hiện train linear regresion model trên tập data advertising.csv theo các yêu cầu sau. Các bạn sẽ dựa trên 3 thông tin đầu vào là TV, Radio, Newspaper để dự đoán Sale.

Giới thiệu về tập data: Data có 200 samples (rows), gồm 4 cột thông tin Tv, Radio, Newspaper, và Sales. Đề bài yêu cầu dùng thông tin ở 3 cột đầu tiên (Tv, Radio, Newspaper) để dự đoán được cột cuối cùng (Sale) dùng linear regression model. Code tải dataset được cung cấp ở bên dưới như sau:

```
1 !gdown 1x30X304P0-bonyooK4FKcRhyrTmT4kif
2 !unzip advertising.zip
```

Để chuẩn hoá data đầu vào, bạn được cung cấp trước function đọc dữ liệu và chuẩn hoá **mean_normalization(X)** như bên dưới:

mean_normalization(X)

```

1 import numpy as np
2 from numpy import genfromtxt
3 import matplotlib.pyplot as plt
4
5 # dataset
6 data = genfromtxt("../content/advertising.csv", delimiter=",",
7                   skip_header=1)
8 N = data.shape[0]
9 X = data[:, :3]
10 y = data[:, 3:]
11
12 # Normalize input data by using mean normalization
13 def mean_normalization(X):
14     N = len(X)
15     maxi = np.max(X)
16     mini = np.min(X)
17     avg = np.mean(X)
18     X = (X - avg) / (maxi - mini)
19     X_b = np.c_[np.ones((N, 1)), X]
20
21 X_b, maxi, mini, avg = mean_normalization(X)

```

Yêu cầu của bài tập này là các bạn lần lượt hiện thực lại giải thuật linear regression để dự đoán Sales dựa vào các yêu cầu sau:

Hoàn thành function **stochastic_gradient_descent()** để huấn luyện data sử dụng Stochastic Gradient Descent. Lưu ý các bạn cần tận dụng tối đa vectorization để hoàn thiện bài tập này.

- **input:** (4 inputs) X_b, y, n_epochs, learning_rate
- **output:** thetas_path, losses

stochastic_gradient_descent()

```

1 def stochastic_gradient_descent(X_b, y, n_epochs=50, learning_rate=0
2                                 .01):
3     N, d_plus1 = X_b.shape

```

```

4   # Step 1: Init parameters
5   thetas = np.asarray([[1.16270837], [-0.81960489],
6   [1.39501033], [0.29763545]])
7   thetas_path = [thetas.copy()]
8   losses = []
9   # Step 2-6: SGD loop
10  for epoch in range(n_epochs):
11      for i in range(N):
12          random_index = i
13          xi = X_b[random_index:random_index+1]
14          yi = y[random_index:random_index+1]
15
16          # Step 3: prediction
17          ***Your code here***
18
19          # Step 4: MSE loss for 1 sample
20          ***Your code here***
21
22          # Step 5: gradient
23          ***Your code here***
24
25          # Step 6: update
26          ***Your code here***
27
28          # log
29          thetas_path.append(thetas.copy())
30          losses.append(li[0][0])
31      return thetas_path, losses

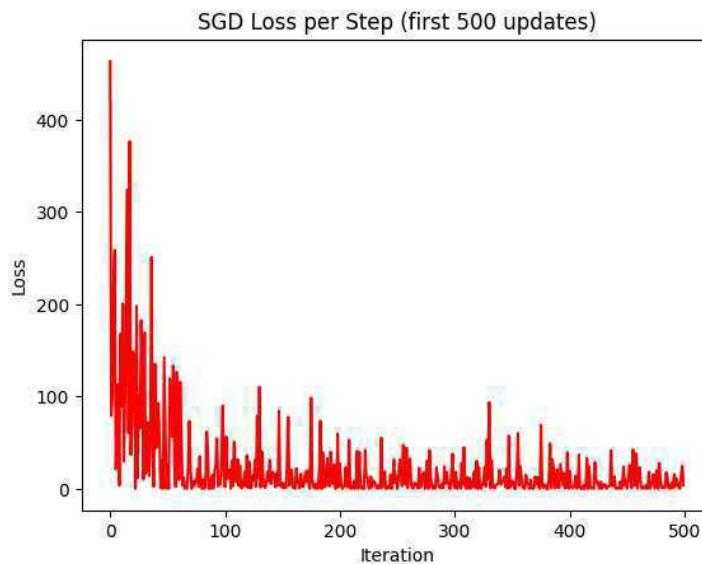
```

Hình 2 là kết quả sau khi thực thi đoạn code sau:

```

1 sgd_theta, losses = stochastic_gradient_descent(X_b, y, n_epochs=50,
2                                                 learning_rate=0.01)
3
4 # In loss cho 500 bước đầu
5 x_axis = list(range(500))
6 plt.plot(x_axis, losses[:500], color="r")
7 plt.xlabel("Iteration")
8 plt.ylabel("Loss")
9 plt.title("SGD Loss per Step (first 500 updates)")
10 plt.show()

```



Hình 22.3: Kết quả loss values sử dụng Stochastic Gradient Descent

Hoàn thành function **mini_batch_gradient_descent()** để huấn luyện data sử dụng Mini-batch Gradient Descent. Lưu ý các bạn cần tận dụng tối đa vectorization để hoàn thiện bài tập này.

- **input:** (5 inputs) X_b , y , n_{epochs} , minibatch_size , learning_rate
- **output:** thetas_path , losses

```
mini_batch_gradient_descent()
```

```

1 def mini_batch_gradient_descent(X_b, y, n_epochs=50, minibatch_size=
2                                     20, learning_rate=0.01):
3     N, d_plus1 = X_b.shape
4     # Step 1: Init parameters
5     thetas = np.asarray([[1.16270837], [-0.81960489],
6                          [1.39501033], [0.29763545]])
7     thetas_path = [thetas.copy()]
8     losses = []
9     for epoch in range(n_epochs):
10         # shuffle data
11         shuffled_indices = np.random.permutation(N)
12         X_b_shuffled = X_b[shuffled_indices]

```

```

12     y_shuffled = y[shuffled_indices]
13
14     for i in range(0, N, minibatch_size):
15         xi = X_b_shuffled[i:i+minibatch_size]      # (m, d+1)
16         yi = y_shuffled[i:i+minibatch_size]        # (m, 1)
17
18         # Step 3: prediction
19         ***Your code here***
20
21         # Step 4: MSE loss for minibatch
22         ***Your code here***
23
24         # Step 5: gradient for minibatch
25         ***Your code here***
26
27         # Step 6: update
28         ***Your code here***
29
30         # log
31         thetas_path.append(thetas.copy())
32         losses.append(loss)
33     return thetas_path, losses

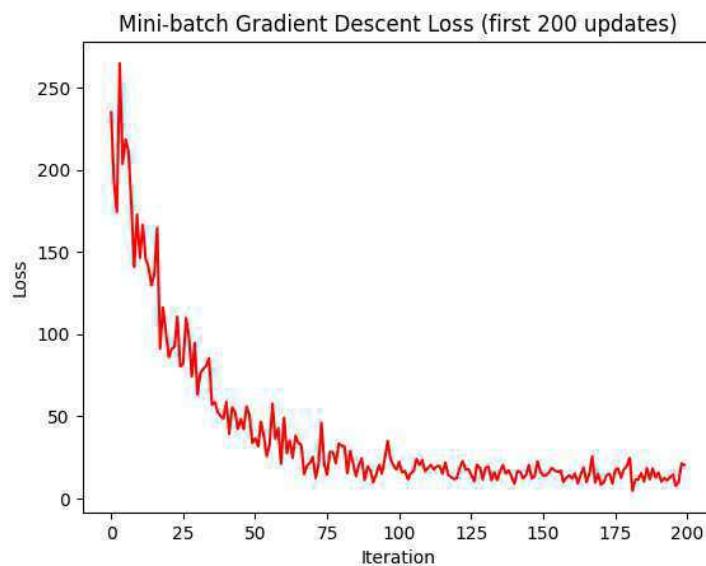
```

Hình 3 là kết quả sau khi thực thi đoạn code sau:

```

1 mbgd_thetas, losses = mini_batch_gradient_descent(X_b, y, n_epochs=
50, minibatch_size=20,
learning_rate=0.01)
2
3 # visualize
4 x_axis = list(range(200))
5 plt.plot(x_axis, losses[:200], color="r")
6 plt.xlabel("Iteration")
7 plt.ylabel("Loss")
8 plt.title("Mini-batch Gradient Descent Loss (first 200 updates)")
9 plt.show()

```



Hình 22.4: Kết quả loss values sử dụng Mini batch Gradient Descent

Hoàn thành function **batch_gradient_descent()** để huấn luyện data sử dụng batch Gradient Descent. Lưu ý các bạn cần tận dụng tối đa vectorization để hoàn thiện bài tập này.

- **input:** (4 inputs) X_b , y , n_{epochs} , learning_rate
- **output:** thetas_path , losses

batch_gradient_descent()

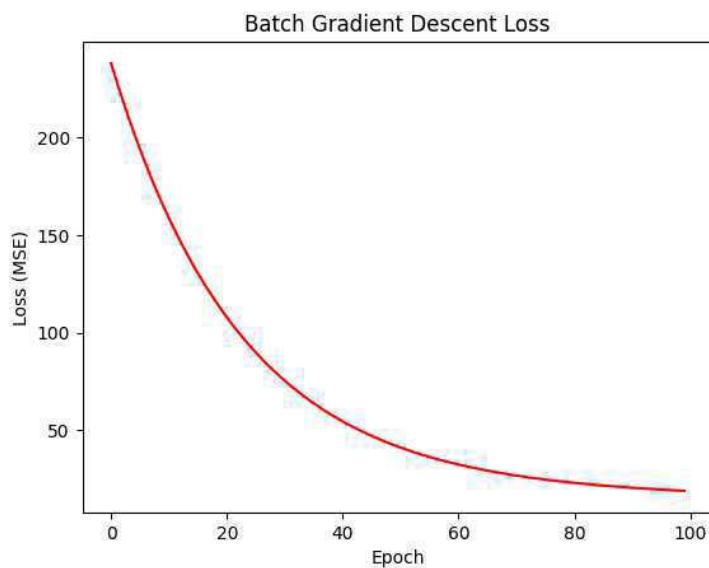
```

1 def batch_gradient_descent(X_b, y, n_epochs=100, learning_rate=0.01):
2     :
3     N, d_plus1 = X_b.shape
4     # Step 1: Init parameters
5     thetas = np.asarray([[1.16270837], [-0.81960489], [1.39501033],
6                         [0.29763545]])
7     thetas_path = [thetas.copy()]
8     losses = []
9
10    for epoch in range(n_epochs):
11        # Step 3: prediction
12        ***Your code here***
```

```
11      # Step 4: MSE loss
12      ***Your code here***
13
14
15      # Step 5: gradient
16      ***Your code here***
17
18      # Step 6: update parameters
19      ***Your code here***
20
21      # log
22      thetas_path.append(thetas.copy())
23      losses.append(loss)
24
25      return thetas_path, losses
```

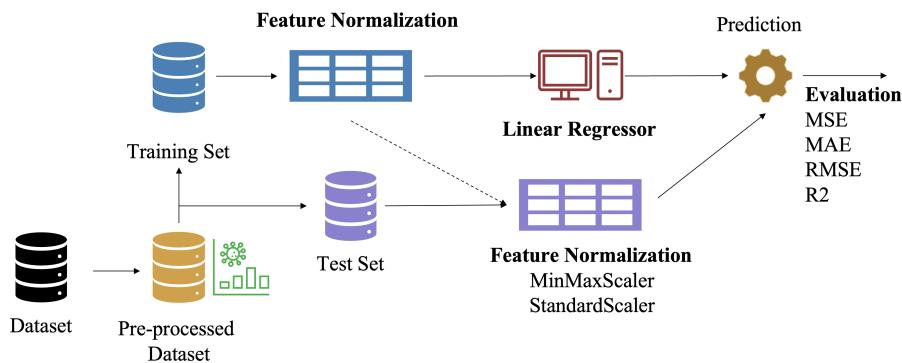
Hình 4 là kết quả sau khi thực thi đoạn code sau:

```
1 bgd_thetas, losses = batch_gradient_descent(X_b, y, n_epochs=100,
                                                learning_rate=0.01)
2
3 # visualize
4 x_axis = list(range(100))
5 plt.plot(x_axis, losses[:100], color="r")
6 plt.xlabel("Epoch")
7 plt.ylabel("Loss (MSE)")
8 plt.title("Batch Gradient Descent Loss")
9 plt.show()
```



Hình 22.5: Kết quả loss values sử dụng Batch Gradient Descent

22.4 Xây dựng mô hình Linear Regression cho bài toán dự đoán Bitcoin



Hình 22.6: Các bước thực hiện xây dựng mô hình hồi quy tuyến tính cho bộ dữ liệu bitcoin.

Bước 1: Tải và đọc dữ liệu

Đầu tiên, ta tiến hành tải bộ dữ liệu Bitcoin forecasting bằng code sau:

```

1 !gdown 1mc89F1dkQ6XRQnwQTqqryjaXt5ZWXTT0
2 !unzip BTC-Daily.zip

```

Import các thư viện cần thiết và tiến hành đọc file.csv như sau:

```

1 import pandas as pd
2 import numpy as np
3 import datetime as dt
4 import matplotlib.pyplot as plt
5
6 df = pd.read_csv("../content/BTC-Daily.csv")
7 df = df.drop_duplicates()
8 df.head()

```

Ở bước này ta sử dụng pandas để đọc dữ liệu lịch sử giá Bitcoin từ file BTC-Daily.csv. Lệnh drop_duplicates() giúp loại bỏ các dòng trùng lặp, đảm bảo dữ liệu sạch. Cuối cùng, df.head() hiển thị 5 dòng đầu tiên của bảng dữ liệu, cho phép ta kiểm tra nhanh cấu trúc và các cột trong dataset.

	unix	date	symbol	open	high	low	close	Volume BTC	Volume USD
0	1646092800	2022-03-01 00:00:00	BTC/USD	43221.71	43626.49	43185.48	43185.48	49.006289	2.116360e+06
1	1646006400	2022-02-28 00:00:00	BTC/USD	37717.10	44256.08	37468.99	43178.98	3160.618070	1.364723e+08
2	1645920000	2022-02-27 00:00:00	BTC/USD	39146.66	39886.92	37015.74	37712.68	1701.817043	6.418008e+07
3	1645833600	2022-02-26 00:00:00	BTC/USD	39242.64	40330.99	38600.00	39146.66	912.724087	3.573010e+07
4	1645747200	2022-02-25 00:00:00	BTC/USD	38360.93	39727.97	38027.61	39231.64	2202.851827	8.642149e+07

Hình 22.7: Vài mẫu dữ liệu của dataset Bitcoin forecasting

Bước 2: Khám phá dữ liệu

```

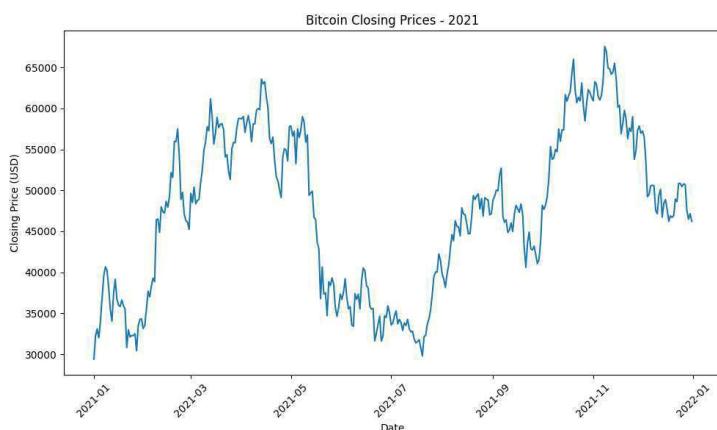
1 # Range of dates covered
2 df["date"] = pd.to_datetime(df["date"])
3 date_range = str(df["date"].dt.date.min()) + " to " + str(df["date"].
4                                         dt.date.max())
5 print(date_range)
6
7 df["year"] = df["date"].dt.year
8 df["month"] = df["date"].dt.month
9 df["day"] = df["date"].dt.day
10
11 unique_years = df["year"].unique()
12 for year in unique_years:
13
14     dates = pd.date_range(start=f"{year}-01-01", end=f"{year}-12-31"
15                           , freq="D")
16     year_month_day = pd.DataFrame({"date": dates})
17     year_month_day["year"] = year_month_day["date"].dt.year
18     year_month_day["month"] = year_month_day["date"].dt.month
19     year_month_day["day"] = year_month_day["date"].dt.day
20
21     merged_data = pd.merge(year_month_day, df, on=["year", "month",
22                             "day"], how="left")

```

```
22
23     # Plot
24     plt.figure(figsize=(10, 6))
25     plt.plot(merged_data["date_x"], merged_data["close"])
26     plt.title(f"Bitcoin Closing Prices - {year}")
27     plt.xlabel("Date")
28     plt.ylabel("Closing Price (USD)")
29     plt.xticks(rotation=45)
30     plt.tight_layout()
31     plt.show()
```

Trong bước này, dữ liệu được xử lý và trực quan hóa theo các bước:

- Chuyển đổi dữ liệu về dạng ngày tháng để xác định khoảng thời gian mà tập dữ liệu bao phủ.
- Tách thêm các cột `year`, `month`, `day` nhằm dễ dàng thao tác theo từng năm.
- Với mỗi năm, dữ liệu được ghép với dãy ngày đầy đủ để tránh bị thiếu ngày (nếu thị trường nghỉ giao dịch hoặc dữ liệu không đầy).
- Vẽ biểu đồ giá đóng cửa (`close`) theo thời gian cho từng năm, giúp quan sát trực quan biến động giá Bitcoin và phát hiện các xu hướng chính trong từng giai đoạn.



Hình 22.8: Biểu đồ giá kết thúc phiên trong năm 2021

Bước 3: Tạo biểu đồ nền giá giao dịch từ năm 2019 - 2022

Cài đặt thư viện mplfinance để vẽ biểu đồ nền giao dịch

```
1 !pip install mplfinance
```

Tiếp theo, ta thực thi code bên dưới để tạo biểu đồ giao dịch từ năm 2019 - 2022

```
1 import matplotlib.pyplot as plt
2 import matplotlib.dates as mdates
3 from mplfinance.original_flavor import candlestick_ohlc
4 import datetime
5
6 # Filter data for 2019-2022
7 df_filtered = df[(df["date"] >= "2019-01-01") & (df["date"] <= "2022-12-31")]
8
9 # Convert date to matplotlib format
10 df_filtered = df[(df["date"] >= "2019-01-01") & (df["date"] <= "2022-12-31")].copy()
11 df_filtered["date"] = df_filtered["date"].map(mdates.date2num)
12
13 # Create the candlestick chart
14 fig, ax = plt.subplots(figsize=(20, 6))
15
16 candlestick_ohlc(ax, df_filtered[["date", "open", "high", "low", "close"]].values, width=0.6,
17                   colorup="g", colordown="r")
18
19 ax.xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m-%d"))
20 fig.autofmt_xdate()
21
22 plt.title("Bitcoin Candlestick Chart (2019-2022)")
23 plt.xlabel("Date")
24 plt.ylabel("Price (USD)")
25 plt.grid(True)
26
```

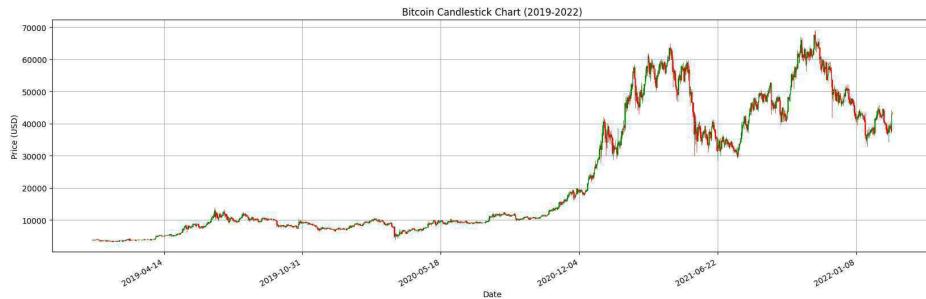
```

27 # Save the plot as a PDF
28 plt.savefig("bitcoin_candlestick_2019_2022.pdf")
29
30 plt.show()

```

Trong bước này, dữ liệu được lọc cho giai đoạn từ năm 2019 đến 2022 và biểu diễn dưới dạng biểu đồ nến (candlestick).

- Mỗi nến thể hiện biến động giá trong một ngày giao dịch, gồm 4 giá trị: `open`, `high`, `low`, `close`.
- Nến màu xanh (green) biểu thị giá đóng cửa cao hơn giá mở cửa, trong khi nến màu đỏ (red) thể hiện chiều ngược lại.
- Biểu đồ nến cho phép quan sát trực quan xu hướng tăng/giảm trong ngắn hạn và các biến động mạnh của giá Bitcoin trong từng giai đoạn.
- Kết quả cuối cùng được lưu ra file PDF để thuận tiện cho việc sử dụng trong báo cáo.



Hình 22.9: Biểu đồ giao dịch từ năm 2019 - 2022

Bước 4: Tạo mô hình Linear Regression

Để huấn luyện mô hình Linear Regression, ta cần xây dựng các hàm cơ bản như `predict`, `gradient` và `update_weight`.

```

1 def predict(X, w, b):
2     ***Your code here***
3
4 def gradient(y_hat, y, x):
5     ***Your code here***
6     return (dw, db, cost)
7
8 def update_weight(w, b, lr, dw, db):
9     ***Your code here***
10    return (w_new, b_new)

```

Bước 5: Chia tập train và test

```

1 from sklearn.preprocessing import StandardScaler
2 from sklearn.model_selection import train_test_split
3
4 X = df[["open", "low", "high"]].values
5 y = df["close"].values
6
7 X_train, X_test, y_train, y_test = train_test_split(
8     X, y, test_size=0.3, random_state=42, shuffle=True
9 )
10
11 scalar = StandardScaler()
12
13 X_train_scaled = scalar.fit_transform(X_train)
14 X_test_scaled = scalar.transform(X_test)

```

Tách dữ liệu:

- X : các đặc trưng open, low, high.
- y : nhãn close.

Chia dữ liệu bằng `train_test_split` (70% train, 30% test, `random_state=42`).

Chuẩn hóa dữ liệu với `StandardScaler`:

- `fit_transform` trên tập train.

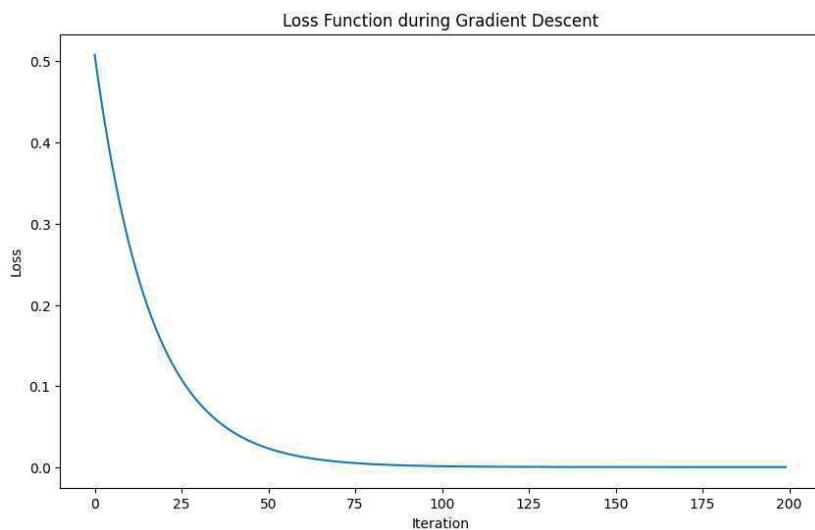
- transform trên tập test.

Bước 6: Huấn luyện và đánh giá mô hình tự xây dựng

```
1 def linear_regression_vectorized(X, y, learning_rate=0.01,
2                                 num_iterations=200):
3     n_samples, n_features = X.shape
4     w = np.zeros(n_features) # Initialize weights
5     b = 0 # Initialize bias
6     losses = []
7
8     ***Your code here***
9
10    return w, b, losses
```

Huấn luyện mô hình bằng Batch Gradient Descent và trực quan hoá quá trình tối ưu bằng cách vẽ biểu đồ hàm mất mát theo số vòng lặp để quan sát mức độ hội tụ của mô hình.

```
1 w, b, losses = linear_regression_vectorized(X_train.values, y_train.
2                                              values, learning_rate=0.01 ,
3                                              num_iterations=200)
4
5 # Plot the loss function
6 plt.figure(figsize=(10, 6))
7 plt.plot(losses)
8 plt.xlabel("Iteration")
9 plt.ylabel("Loss")
10 plt.title("Loss Function during Gradient Descent")
11 plt.show()
```



Hình 22.10: Biểu đồ hàm loss trong quá trình huấn luyện

Sau khi mô hình được huấn luyện, ta tiến hành đánh giá hiệu quả dự đoán bằng cách tính các chỉ số sai số phổ biến như **RMSE**, **MAE**, **MAPE**, và hệ số xác định R^2 cho cả tập huấn luyện và tập kiểm tra. Các chỉ số này giúp kiểm chứng độ chính xác và khả năng khái quát hoá của mô hình.

```

1 from sklearn.metrics import r2_score
2
3 # Make predictions on the test set
4 y_pred = predict(X_test, w, b)
5
6 # Calculate RMSE
7 rmse = np.sqrt(np.mean((y_pred - y_test) ** 2))
8
9 # Calculate MAE
10 mae = np.mean(np.abs(y_pred - y_test))
11
12 # Calculate MAPE
13 mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
14
15
16 # Calculate R-squared on training data
17 y_train_pred = predict(X_train, w, b)

```

```

18 train_accuracy = r2_score(y_train, y_train_pred)
19
20 # Calculate R-squared on testing data
21 test_accuracy = r2_score(y_test, y_pred)
22
23 print("Root Mean Square Error (RMSE):", round(rmse, 4))
24 print("Mean Absolute Error (MAE):", round(mae, 4))
25 print("Training Accuracy (R-squared):", round(train_accuracy, 4))
26 print("Testing Accuracy (R-squared):", round(test_accuracy, 4))

```

Để minh họa kết quả dự đoán, ta thử nghiệm trên dữ liệu quý 1 năm 2019 (từ 01/01 đến 31/03). Biểu đồ dưới đây thể hiện sự so sánh giữa **giá thực tế** và **giá dự đoán**, giúp quan sát trực quan mức độ sai lệch và xu hướng giữa hai đường giá.

```

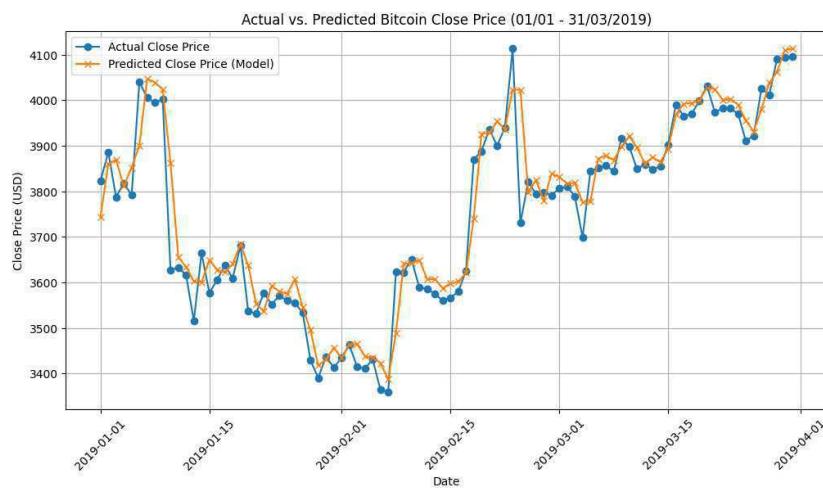
1 # Lọc dữ liệu 2019 Q1
2 df_2019_Q1 = df[(df["date"] >= "2019-01-01") & (df["date"] <= "2019-
3
4 # Chọn đúng các cột feature đã train
5 X_2019 = df_2019_Q1[["open", "low", "high"]].values
6 X_2019_scaled = scalar.transform(X_2019)    # dùng scalar đã fit trướ
7                                     c đó
8 y_2019 = df_2019_Q1["close"].values
9
10 # Dự đoán bằng mô hình
11 y_pred_2019 = predict(X_2019_scaled, w, b)
12
13 # Gắn vào dataframe để dễ vẽ
14 df_2019_Q1["predicted_close"] = y_pred_2019
15
16 # Vẽ biểu đồ
17 plt.figure(figsize=(12,6))
18 plt.plot(df_2019_Q1["date"], df_2019_Q1["close"], label="Actual
19                                     Close Price", marker="o")
20 plt.plot(df_2019_Q1["date"], df_2019_Q1["predicted_close"], label="Predicted Close Price (Model)",
21                                     marker="x")
22 plt.title("Actual vs. Predicted Bitcoin Close Price (01/01 - 31/03/
23                                     2019)")
24 plt.xlabel("Date")

```

```

21 plt.ylabel("Close Price (USD)")
22 plt.xticks(rotation=45)
23 plt.legend()
24 plt.grid(True)
25 plt.show()

```



Hình 22.11: Biểu đồ so sánh giá kết thúc phiên thực tế và dự đoán của Bitcoin (01/01/2019 - 31/03/2019).

Bước 7: Huấn luyện và đánh giá mô hình từ thư viện scikit-learn

```

1 from sklearn.linear_model import LinearRegression
2
3 linear_regressor = LinearRegression()
4 linear_regressor.fit(X_train_scaled, y_train)

```

Mô hình được xây dựng bằng `LinearRegression` của `scikit-learn`, được huấn luyện trên dữ liệu đã chuẩn hóa để tự động tìm ra các hệ số w và bias b tối ưu.

```

1 # Make predictions on the test set
2 y_pred = linear_regressor.predict(X_test_scaled)
3
4 # Evaluation
5 mse = mean_squared_error(y_test, y_pred)
6 mae = mean_absolute_error(y_test, y_pred)
7 r2 = r2_score(y_test, y_pred)
8
9 print("Mean Square Error (MSE):", round(mse, 4))
10 print("Mean Absolute Error (MAE):", round(mae, 4))
11 print("R-squared:", round(r2, 4))

```

Mô hình sau khi huấn luyện được dùng để dự đoán trên tập kiểm tra, các chỉ số đánh giá gồm MSE (Mean Squared Error), MAE (Mean Absolute Error) và R^2 (R-squared) được tính toán để đo lường chất lượng dự báo.

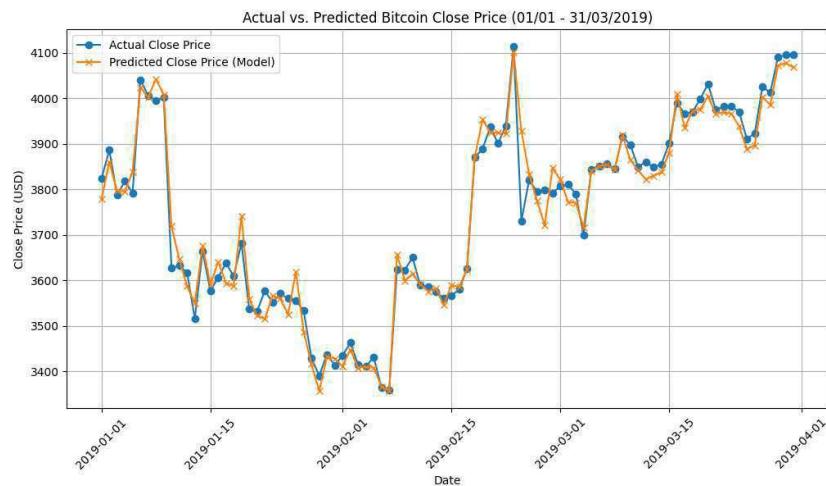
```

1 # Lọc dữ liệu 2019 Q1
2 df_2019_Q1 = df[(df["date"] >= "2019-01-01") & (df["date"] <= "2019-
3 03-31")].copy()
4
5 # Chọn đúng các cột feature đã train
6 X_2019 = df_2019_Q1[["open", "low", "high"]].values
7 X_2019_scaled = scalar.transform(X_2019)
8 y_2019 = df_2019_Q1["close"].values
9
10 # Dự đoán bằng model sklearn
11 y_pred_2019 = linear_regressor.predict(X_2019_scaled)
12
13 # Gắn vào dataframe để tiện vẽ
14 df_2019_Q1["predicted_close"] = y_pred_2019
15
16 # Vẽ
17 plt.figure(figsize=(12,6))
18 plt.plot(df_2019_Q1["date"], df_2019_Q1["close"], label="Actual
19           Close Price", marker="o")
20 plt.plot(df_2019_Q1["date"], df_2019_Q1["predicted_close"], label="Predicted Close Price (Model)",
21           marker="x")
22 plt.title("Actual vs. Predicted Bitcoin Close Price (01/01 - 31/03/
23           2019)")
24 plt.xlabel("Date")

```

```
21 plt.ylabel("Close Price (USD)")  
22 plt.xticks(rotation=45)  
23 plt.legend()  
24 plt.grid(True)  
25 plt.show()
```

Dữ liệu được lọc cho giai đoạn Q1/2019, sau đó chuẩn hoá bằng scaler đã huấn luyện. Mô hình LinearRegression được sử dụng để dự đoán giá đóng cửa, và kết quả được so sánh trực quan với giá thực tế thông qua biểu đồ đường.



Hình 22.12: Biểu đồ so sánh giá kết thúc phiên thực tế và dự đoán của Bitcoin (dùng sklearn) (01/01/2019 - 31/03/2019).

22.5 Câu hỏi trắc nghiệm

1. (Lý thuyết) Trong Linear Regression, tại sao cần thêm bias b vào vector tham số θ ?
 - (a) Để làm mô hình phức tạp hơn
 - (b) Để dịch chuyển đường hồi quy, tránh ràng buộc đi qua gốc tọa độ
 - (c) Để giảm số chiều của dữ liệu
 - (d) Để tăng tốc độ hội tụ của gradient descent
2. (Lý thuyết) So sánh giữa Stochastic Gradient Descent (SGD) và Batch Gradient Descent (BGD), phát biểu nào đúng?
 - (a) SGD dùng toàn bộ dữ liệu trong mỗi bước cập nhật, còn BGD dùng một mẫu duy nhất
 - (b) SGD hội tụ nhanh hơn nhưng dao động nhiều hơn, BGD ổn định hơn nhưng chậm
 - (c) SGD và BGD cho cùng tốc độ hội tụ
 - (d) SGD luôn chính xác hơn BGD
3. Với 1 sample ($x = 2, y = 5$), tham số $w = 1, b = 0$. Giá trị dự đoán \hat{y} là gì? Công thức: $\hat{y} = wx + b$
 - (a) 2
 - (b) 3
 - (c) 4
 - (d) 5
4. Cho $\hat{y} = 7, y = 4$, hàm mất mát MSE theo 1 mẫu là gì? Công thức: $L = (\hat{y} - y)^2$
 - (a) 3
 - (b) 6
 - (c) 9
 - (d) 12

5. Với $\mathbf{x} = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1 \end{bmatrix}$, $\boldsymbol{\theta} = \begin{bmatrix} 1 \\ 0 \\ 2 \\ 1 \end{bmatrix}$, giá trị dự đoán \hat{y} là gì? Công thức: $\hat{y} = \mathbf{x}^T \boldsymbol{\theta} = \boldsymbol{\theta}^T \mathbf{x}$

- (a) 6
- (b) 8
- (c) 10
- (d) 12

6. Cho batch gồm $m = 2$ mẫu: $\mathbf{X} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 0 & 1 \end{bmatrix}$, $\boldsymbol{\theta} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$ Tính vector dự đoán $\hat{\mathbf{y}}$. Công thức: $\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\theta}$

- (a) $[4, 3]^\top$
- (b) $[5, 4]^\top$
- (c) $[6, 5]^\top$
- (d) $[3, 2]^\top$

7. Với batch trên, tính hàm mất mát trung bình. Công thức: $L = \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2$

- (a) 0.5
- (b) 1.0
- (c) 1.5
- (d) 2.0

8. Với sample ($\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$, $y = 5$), $\boldsymbol{\theta} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$, tính gradient $\nabla_{\boldsymbol{\theta}} L$. Công thức: $\nabla_{\boldsymbol{\theta}} L = 2\mathbf{x}(\hat{y} - y)$

- (a) $[-2, -4, -2]^\top$
- (b) $[2, 4, 2]^\top$
- (c) $[-4, -8, -4]^\top$

- (d) $[4, 8, 4]^\top$
9. Cho batch $\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$, $\boldsymbol{\theta} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$, tính gradient $\nabla_{\theta} L$. Công thức: $\nabla_{\theta} L = \frac{2}{m} \mathbf{X}^\top (\hat{\mathbf{y}} - \mathbf{y})$
- (a) $[0, 0]^\top$
(b) $[1, 1.5]^\top$
(c) $[-1, -1.5]^\top$
(d) $[-2, -3]^\top$
10. Sau 1 bước cập nhật với $\boldsymbol{\theta} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $\nabla_{\theta} L = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$, learning rate $\alpha = 0.1$, tham số mới θ là gì? Công thức: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\theta} L$
- (a) $[0.8, 0.6]^\top$
(b) $[1.2, 1.4]^\top$
(c) $[0.9, 0.7]^\top$
(d) $[0.7, 0.3]^\top$

1. **Hint:** Các file code gợi ý và dữ liệu nếu có được lưu trong thư mục có thể được tải [tại đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại [đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 giờ.

AIO_QAs-Verified

• Nhóm Riêng tư · 1,4K thành viên



Hình 22.13: Hình ảnh group facebook AIO Q&A

4. Rubric:

Module 4 - Exercise - Linear Regression - Rubric		
Câu	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Khái niệm linear regression và các biến thể của thuật toán Gradient Descent (Stochastic Gradient Descent, Mini Batch Gradient Descent, và Batch Gradient Descent) - Cách thức thực hiện 3 biến thể trên của Gradient Descent theo kiểu Vectorization (với sự hỗ trợ của thư viện numpy) - Cách đọc và xử lý data cơ bản với numpy 	<ul style="list-style-type: none"> - Hiểu và nắm được ý chính cách hoạt động của 3 biến thể của thuật toán Gradient Descent (Stochastic Gradient Descent, Mini Batch Gradient Descent, và Batch Gradient Descent) với Linear Regression - Vận dụng và thao tác với thư viện numpy để thực hiện 3 biến thể trên theo kiểu Vectorization - Biết cách load data kiểu csv và normalize các feature trước khi đi train linear regression model
2	<ul style="list-style-type: none"> - Hiểu về cách áp dụng của thuật toán Linear Regression cho dữ liệu Time Series - Phân tích một số đặc trưng của dữ liệu Time Series 	<ul style="list-style-type: none"> - Áp dụng Linear Regression cho bài toán dự đoán giá Bitcoin - Đánh giá hiệu quả trên những cài đặt khác nhau

Chương 23

Tính ngẫu nhiên và Giải thuật di truyền

23.1 Giới thiệu

Trong bài này, chúng ta sẽ thực hành lập trình về nội dung **Genetic Algorithm (GA)**. GA là phương pháp tối ưu hoá theo quần thể lời giải, trong đó mỗi thế hệ sẽ thực hiện các bước tính toán gồm: *fitness* (tạm dịch: đánh giá), *selection* (tạm dịch: chọn lọc), *crossover* (tạm dịch: lai ghép) và *mutation* (tạm dịch: đột biến). Qua loạt bài tập, chúng ta củng cố lại các khái niệm cốt lõi trong GA đã học và kết nối tất cả thành một pipeline cài đặt code hoàn chỉnh để giải một số bài toán học máy cơ bản.

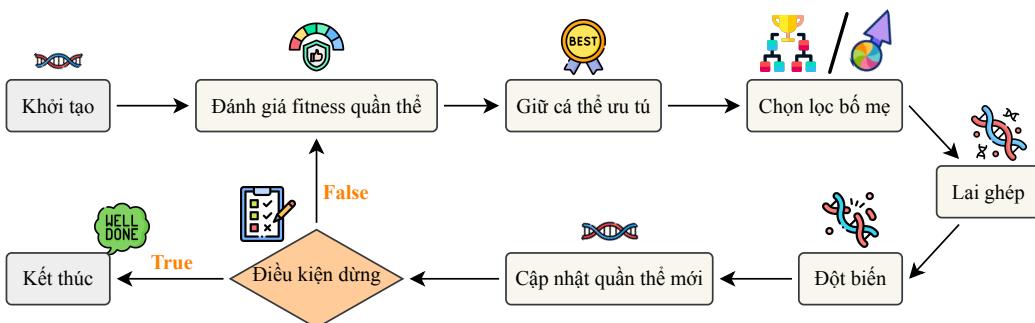


Hình 23.1: Các bước sẽ thực hiện trong bài tập Genetic Algorithm (GA).

Theo đó, nội dung bài tập lần này sẽ gồm có ba bài tập thực hành lớn và các câu trắc nghiệm đi kèm. Tóm tắt nội dung của các bài thực hành được thể hiện qua bảng sau:

Tổng quan 3 bài thực hành			
Bài	Mục tiêu	Biểu diễn	Hàm Fitness
1	Tối đa hoá số bit 1.	Chuỗi nhị phân	Tổng số bit 1 hoặc <i>trap</i> theo block
2	Tối ưu vector trọng số hồi quy tuyến tính để giảm MSE.	Vector thực $\mathbf{w} \in \mathbb{R}^d$	$\text{fitness} = \frac{1}{\text{MSE}_{\text{val}} + \varepsilon}$ ($\varepsilon = 10^{-8}$)
3a	Chọn tập đặc trưng TF-IDF (1000 chiều) cho phân loại đa lớp, tối ưu <i>macro-F1</i> và phạt số đặc trưng.	biểu diễn nhị phân $\mathbf{m} \in \{0, 1\}^{1000}$	$\text{fitness} = \text{F1}_{\text{macro}, \text{val}} - \lambda \ \mathbf{m}\ _0$
3b	Dùng GA chọn tham số tối ưu cho <i>XGBoost</i> .	Nhiễm sắc thể tham số hỗn hợp $\boldsymbol{\theta} = [\text{real}/\text{int}/\text{cat}]$	$\text{fitness} = \text{F1}_{\text{macro}, \text{val}}$ (hoặc điểm CV trên tập validation)

Mặc dù bối cảnh ứng dụng khác nhau, song cả ba bài đều tuân theo cùng một chu trình của **GA**: khởi tạo quần thể, *đánh giá fitness*, *chọn lọc*, *lai ghép*, *đột biến* (kèm *elitism*) và cập nhật thế hệ. Dưới đây là *pipeline* tổng quát các bước cài đặt GA trong Python:



Hình 23.2: Pipeline tổng quan các bước tính toán trong GA.

1. Nhập thư viện

- Tải các gói/phụ thuộc cho sinh ngẫu nhiên, xử lý mảng, trực quan hoá và đánh giá.
- Định nghĩa `set_random_seed(seed)` để cố định nguồn ngẫu nhiên cho toàn bộ thí nghiệm.

2. Chuẩn bị dữ liệu

- Xác định nguồn dữ liệu và mục tiêu tối ưu.
- Tiền xử lý theo yêu cầu: chuẩn hoá/thang đo, mã hoá đặc trưng, tách train/validation/test hoặc cross-validation; mọi biến đổi được *fit* trên train và áp dụng nhất quán lên các tập còn lại.

3. Các hàm/toán tử GA cần thiết

- Khởi tạo cá thể/quần thể: `create_individual()`, `init_population()` (nhi phân, số thực hoặc hỗn hợp tùy bài).
- Đánh giá độ thích nghi: `compute_fitness()` (định nghĩa phù hợp với mục tiêu; có thể cần dịch/chuẩn hoá miền giá trị).
- Chọn lọc: `selection_operator()` (ví dụ: tournament, proportional, rank, stochastic universal sampling).
- Lai ghép và đột biến: `crossover_operator()` (một điểm, hai điểm, đồng nhất, hoặc kết hợp tuyến tính cho biến thực); `mutation_operator()` (lật bit, nhiễu Gauss, hoán vị, kẹp biên/chiều miền khả thi khi cần).

4. Cấu hình thực nghiệm

- Thiết lập siêu tham số: `pop_size`, `generations`, `elitism`, `crossover_rate`, `mutation_rate` và tham số riêng của toán tử (ví dụ `tournament_size`).
- Xác định các kịch bản so sánh (thay đổi `selection`, tỷ lệ `crossover/mutation`, lịch `mutation`, v.v.) để phân tích độ nhạy.

5. Tiết hoá GA

- Khởi tạo quần thể và đánh giá fitness ban đầu.
- Lặp theo thê hệ:
 - (a) Bảo toàn elitism (sao chép một số cá thể tốt nhất).
 - (b) Chọn cặp cha mẹ bằng `selection_operator()`.
 - (c) Tạo con bằng `crossover_operator()` và áp dụng `mutation_operator()`.
 - (d) Đánh giá fitness thê hệ con, hình thành quần thể mới (gộp elite).
 - (e) Ghi nhật ký (ít nhất `best/avg fitness`); áp dụng dừng sớm khi thoả tiêu chí (ngưỡng, patience, hội tụ).
- Trả về cá thể tốt nhất và lịch sử fitness.

6. Đánh giá kết quả

- Khoá cấu hình theo cá thể tốt nhất trên validation và báo cáo trên test/hold-out/CV.
- Trình bày chỉ số phù hợp mục tiêu (ví dụ: sai số, độ chính xác, F-measure, mục tiêu đa mục tiêu nếu có).
- Trực quan hoá: đường hội tụ fitness, phân tích ổn định/độ nhạy, và các biểu đồ hỗ trợ giải thích tùy bài toán.

Thông qua các bước thực hiện cơ bản của GA được mô tả ở trên, chúng ta sẽ tiến hành ứng dụng vào các bài tập ở phần sau.

23.2 Thực hành

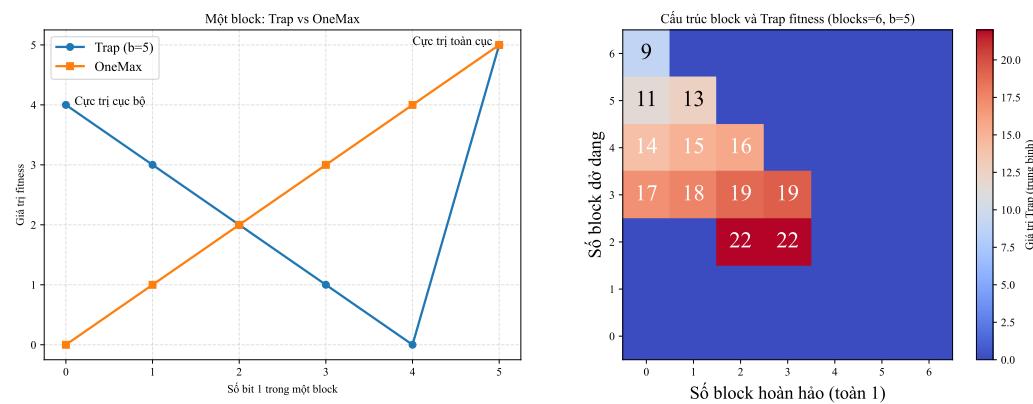
23.2.1 Bài tập 1: OneMax và Trap Fitness

Phát biểu bài toán

Nội dung

Triển khai **GA** trên không gian nhị phân và so sánh hai cơ chế chọn lọc: Tournament Selection và Proportional Selection (Roulette-Wheel). Hai hàm fitness sử dụng là OneMax và Trap. Sau khi hoàn thành chương trình, ta quan sát đường hội tụ và làm rõ vai trò của chọn lọc, đột biến và elitism trong việc cân bằng giữa khai thác và khám phá.

Trap là một hàm fitness được thiết kế để tạo ra bẫy cục bộ. Ở mỗi block độ dài b (thường dùng $b = 5$), cá thể chỉ nhận điểm tối đa khi toàn bộ bit trong block đều bằng 1; nếu còn thiếu bất kỳ bit 1 nào thì điểm bị phạt theo mức độ “dở dang”. Cấu trúc thường-phạt này khiến quần thể dễ mắc kẹt tại các nghiệm cục bộ dù còn xa tối ưu toàn cục, do các thay đổi nhỏ xung quanh nghiệm cục bộ làm giảm fitness. So với **OneMax** - nơi fitness tăng đều theo số bit 1 - Trap khó hơn đáng kể vì các bước cải thiện cục bộ không dẫn tới tiến bộ toàn cục và dễ khiến lựa chọn tham lam hoặc áp lực chọn lọc cao hội tụ sớm.



Hình 23.3: So sánh landscape giữa OneMax và Trap. (Trái) Một block đơn ($b = 5$); (Phải) Cấu trúc nhiều block ($n_{\text{blocks}} = 6, b = 5$).

Hình 23.3 cho thấy rõ cách mà hàm **Trap** tạo ra **bẫy cục bộ** trong không gian tìm kiếm:

- **Biểu đồ bên trái:** minh họa hình dạng của hàm fitness với một block gồm 5 bit. Với **OneMax** (đường cam), giá trị fitness tăng tuyến tính theo số bit 1, được cho bởi

$$f(\mathbf{x}) = f_{\text{ONEMAX}}(u), \quad u = \sum_{i=1}^5 x_i,$$

$$f_{\text{ONEMAX}}(u) = u \quad (\mathbf{x} \in \{0, 1\}^5).$$

Nghĩa là càng nhiều bit 1 thì fitness càng cao, dẫn thẳng tới nghiệm toàn cục tại $u = 5$. Trái lại, với **Trap** (đường xanh), chỉ khi toàn bộ bit trong block đều bằng 1 mới đạt điểm tối đa; các trạng thái “dở dang” bị phạt. Ví dụ, với block $b = 5$:

$$f(\mathbf{x}) = f_{\text{TRAP}}(u), \quad u = \sum_{i=1}^5 x_i,$$

$$f_{\text{TRAP}}(u) = \begin{cases} 5, & \text{nếu } u = 5, \\ 4 - u, & \text{nếu } u < 5 . \end{cases}$$

Các chuỗi gần đúng (như $u = 4$) vẫn bị giảm điểm, tạo một **local optimum** tại $u = 0$. Do đó, quần thể tiến hoá tuyến tính dễ “rơi vào bẫy” này và khó vượt ngưỡng trung gian để đạt tối ưu toàn cục.

- **Biểu đồ bên phải:** mở rộng thêm nhiều block ($n_{\text{blocks}} = 6$, mỗi block dài $b = 5$). Trục hoành thể hiện số block hoàn hảo (chứa toàn bit 1), trục tung là số block chưa hoàn thiện. Mỗi ô trong heatmap biểu diễn **giá trị fitness trung bình** của các nhiễm sắc thể có cấu trúc tương ứng. Màu đỏ thể hiện vùng fitness cao, xanh là thấp. Dễ thấy các tổ hợp có vài block hoàn hảo xen lẫn block dở dang (góc giữa bên phải) lại bị phạt nặng, dù tổng số bit 1 không nhỏ để minh họa hiện tượng “đánh lừa” của trap: *những tiến bộ cục bộ không dẫn đến lợi ích toàn cục*.

Mục tiêu của bài tập

- Cài đặt và chạy GA cơ bản trên không gian nhị phân, tương ứng với pipeline được mô tả ở Hình 23.2.
- So sánh hành vi hội tụ giữa hai cơ chế chọn lọc: **Tournament** (áp lực chọn cao) và **Proportional** (xác suất tỉ lệ với fitness).
- Quan sát sự khác biệt về hình dạng của hàm fitness với **OneMax** và **Trap**.
- Đánh giá ảnh hưởng của **elitism** và **mutation rate** đến khả năng thoát khỏi cực trị cục bộ.

Yêu cầu

Dựa vào file code [**Hint**]_Exercise_01 trong Phụ lục, các bạn hãy hoàn thiện các phần còn thiếu để chạy trọn vẹn pipeline của bài. Cụ thể:

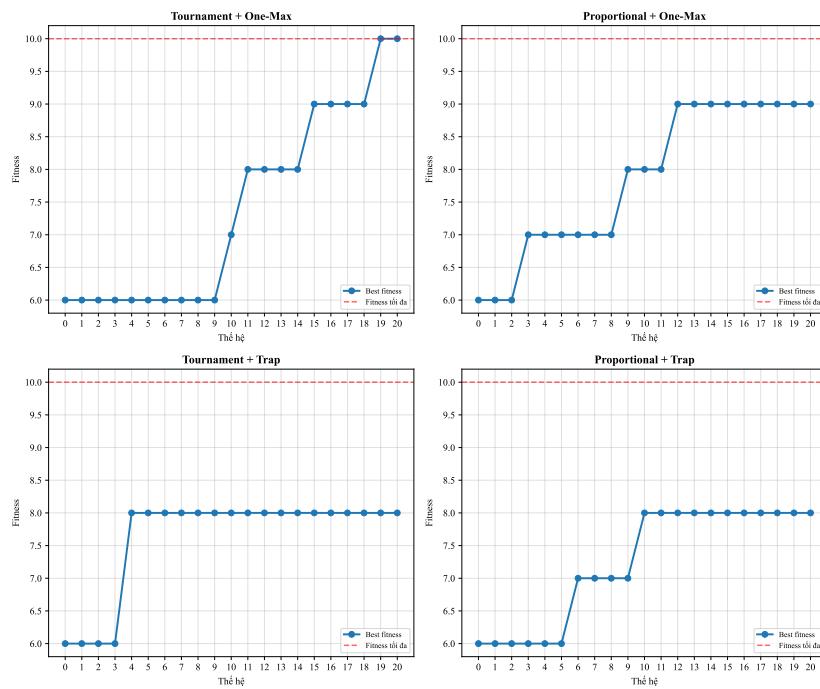
- Fitness: hỗ trợ OneMax (đếm tổng bit 1) và Trap (chia block 5 bit, chỉ thường khi cả block đều là 1, phạt trạng thái chưa hoàn chỉnh).
- Toán tử crossover/mutation/elitism: hiện thực `uniform_crossover`, `bitflip_mutation` và cơ chế elitism.
- Cơ chế selection: triển khai `tournament` và `proportional` (Roulette–Wheel).

Cấu hình mặc định dùng trong bài:

Bảng cấu hình siêu tham số - Bài 1

Tham số	Giá trị
Kích thước quần thể (pop_size)	5
Số thế hệ (generations)	21
Số cá thể ưu tú (elitism)	2
Tỉ lệ lai ghép (crossover_rate)	0.9
Tỉ lệ đột biến (mutation_rate)	0.1
Độ dài chuỗi (CHROM_SIZE)	10
Seed cố định (seed)	42

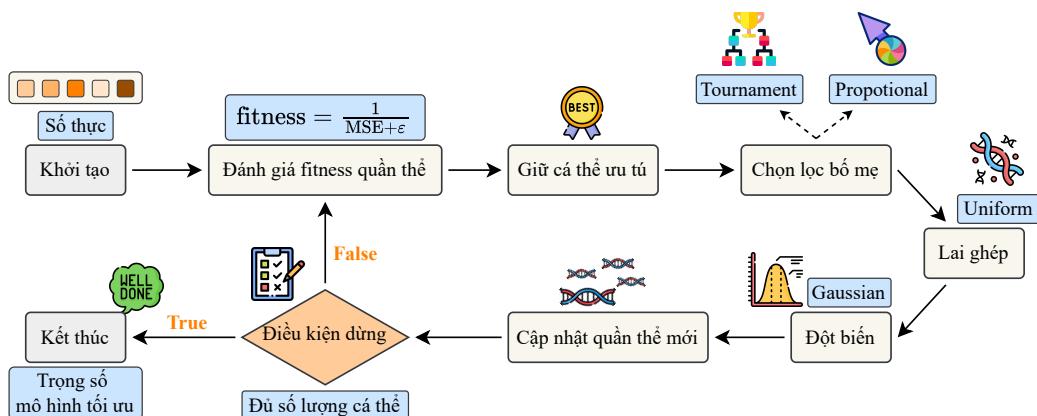
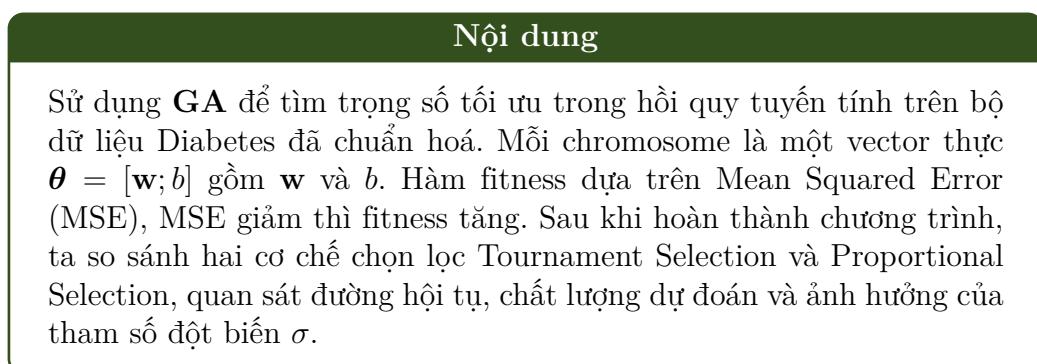
Sau khi hoàn thành phần code trên, các bạn có thể kỳ vọng kết quả thu được như sau:



Hình 23.4: Minh họa việc so sánh fitness đạt được trên hàm OneMax và Trap của hai chiến lược chọn lọc khác nhau.

23.2.2 Bài tập 2: Linear Regression

Phát biểu bài toán



Hình 23.5: Minh họa bài toán hồi quy tuyến tính được tối ưu bằng GA.

So với Bài 1 (nhị phân), bài này khác ở ba điểm chính để phù hợp với hồi quy tuyến tính: (i) biểu diễn cá thể là vector thực $\theta = [w; b]$ thay cho chuỗi bit; (ii) fitness được xây dựng từ MSE trên validation); (iii) đột biến dùng nhiễu Gaussian $\mathcal{N}(0, \sigma^2)$ thay cho bit-flip. Các thành phần còn lại giữ nguyên tinh thần của Bài 1: chọn lọc (Tournament/Proportional), uniform crossover trên tham số thực, elitism và vòng lặp **GA**. Khuyến nghị chuẩn hoá đặc trưng trước khi tối ưu, báo cáo MSE trên train/validation/test và vẽ đường hội tụ.

Mục tiêu của bài tập

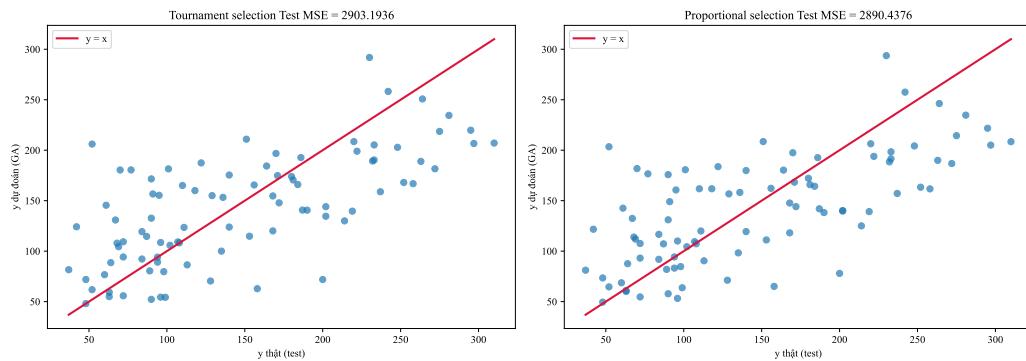
- Mô hình hoá chromosome thực $\theta = [\mathbf{w}; b]$ và xây dựng fitness từ MSE.
- Cài đặt và so sánh **Tournament Selection** và **Proportional Selection**.
- Phân tích hội tụ (best-train MSE) và năng lực tổng quát hoá (MSE trên test).

Yêu cầu

Dựa vào file code **[Hint]_Exercise_02** trong Phụ lục, hãy hoàn thiện các phần còn thiếu để chạy trọn vẹn pipeline. Cụ thể:

- Fitness/MSE: điền `predict_linear` và `mse_loss` để tính MSE trên dữ liệu đã chuẩn hoá.
- Selection: hoàn thiện `tournament_select` (tham số `tournament_size`) và `proportional_select` (xây `probs` từ $1/\text{loss}$).
- Crossover/Mutation: hiện thực `uniform_crossover` (biểu diễn 0.5 khi thoả `crossover_rate`) và `gaussian_mutation` (nhiều $\mathcal{N}(0, \sigma^2)$ áp dụng theo `mutation_rate`).
- Vòng lặp GA: hoàn thiện `evaluate_population` (sắp theo loss tăng dần) và `collect_elites`.

Sau khi hoàn thành cài đặt, các bạn có thể ghi nhận lại kết quả đạt được với GA đối với bài toán Linear Regression trên hai chiến lược chọn lọc:



Hình 23.6: Minh họa kết quả MSE đạt được trên tập test sau của hai chiến lược chọn lọc khác nhau.

23.2.3 Bài tập 3: Text Classification

Phát biểu bài toán

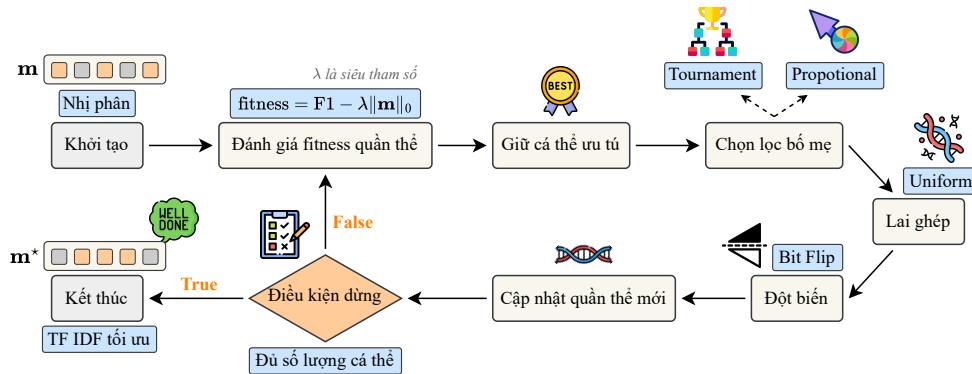
Nội dung

Xử lý bài toán phân loại văn bản trên tập 20 Newsgroups với bốn chủ đề: `comp.graphics`, `rec.sport.hockey`, `sci.space`, `talk.politics.mideast` với mô hình XGBoost. Dữ liệu văn bản được biểu diễn bằng TF-IDF (tối đa 1000 đặc trưng) và tách train/validation/test như trong file mã nguồn tham khảo. Trong đó, ta ứng dụng GA để giải quyết hai phần sau:

- **3a – GA chọn đặc trưng:** dùng biểu diễn nhị phân (binary mask) trên không gian TF-IDF để tối ưu macro-F1 trên validation, kèm điểm phạt theo số đặc trưng được chọn.
- **3b – GA tinh chỉnh tham số XGBoost:** tối ưu bộ siêu tham số `n_estimators`, `max_depth`, `learning_rate`, `subsample`, `colsample_bytree`, `reg_lambda` nhằm cải thiện macro-F1 trên validation, sau đó đánh giá trên test.

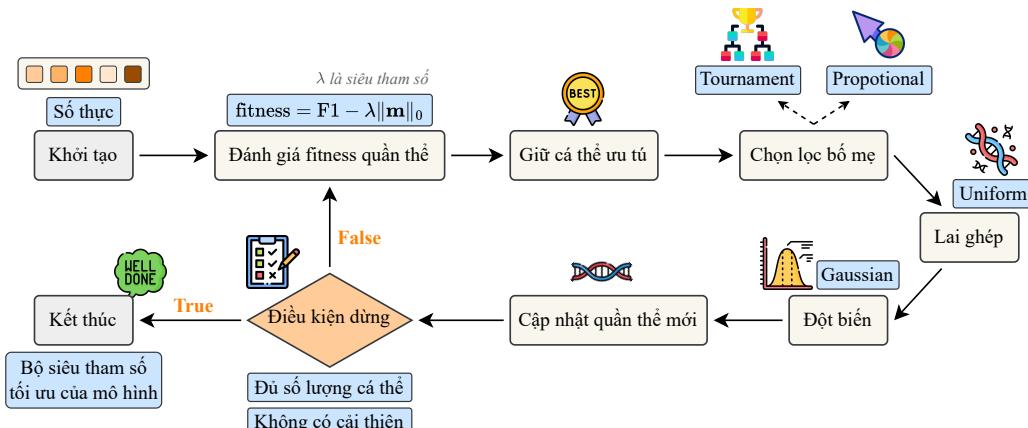
Ở phần bài tập này, khác với hai bài trước, ta sẽ ứng dụng GA vào một bài toán học máy thực tế hơn. Cụ thể, ta sẽ triển khai một bài toán liên quan đến xử lý ngôn ngữ tự nhiên, làm việc với văn bản nên đặc trưng đầu vào được mã hoá TF-IDF có kích thước lớn và thừa. Với hai ngữ cảnh:

1. Sử dụng GA để tìm đặc trưng tối ưu trong việc biểu diễn văn bản.



Hình 23.7: Minh họa bài toán tìm đặc trưng tối ưu trong việc biểu diễn văn bản bằng GA.

2. Với đặc trưng tối ưu đã tìm được, ta ứng dụng để huấn luyện mô hình phân loại XGBoost, đồng thời ứng dụng GA để tìm bộ siêu tham số tốt nhất cho mô hình.



Hình 23.8: Minh họa bài toán tìm các siêu tham số tối ưu cho mô hình XGBoost bằng GA.

Fitness ưu tiên chất lượng phân loại (macro-F1 trên validation) và có thể kèm phạt độ thừa để khuyến khích mô hình gọn nhẹ. Hai kiểu chromosome sẽ xuất hiện: biểu diễn nhị phân để chọn đặc trưng (3a) và chuỗi tham số hỗn hợp cho XGBoost (3b). Khi đánh giá cần giữ tỉ lệ lớp ổn định (tách dữ liệu theo kiểu stratified) và báo cáo thêm confusion matrix để quan sát lỗi.

Mục tiêu của bài tập

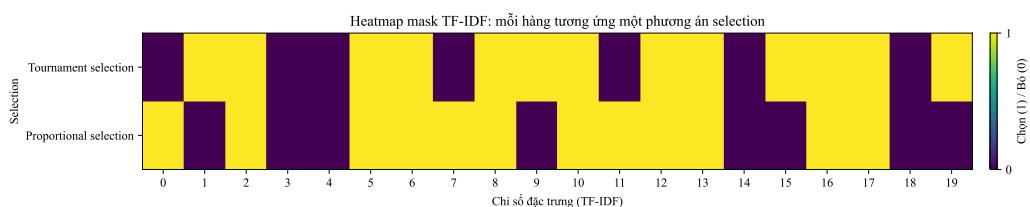
- Xây dựng pipeline tiền xử lý văn bản với TF-IDF, tách train/validation/test theo stratified split.
- Dùng **GA** để tối ưu macro-F1: (3a) bằng chọn đặc trưng với biểu diễn nhị phân, (3b) bằng tinh chỉnh siêu tham số XGBoost.
- Phân tích ảnh hưởng của selection, crossover, mutation và elitism lên tốc độ hội tụ và chất lượng mô hình.

Yêu cầu

Dựa vào file code [**Hint**]_Exercise_03 trong Phụ lục, hãy hoàn thiện các phần còn thiếu để chạy trọn vẹn hai phần 3a và 3b.

3a – GA chọn đặc trưng TF-IDF

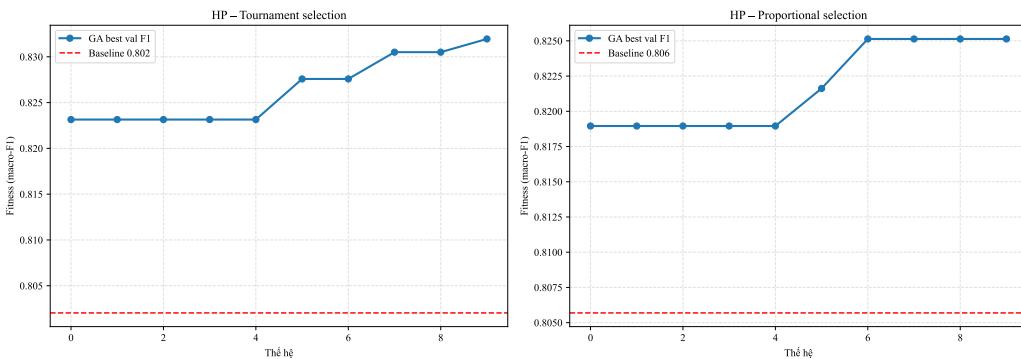
- Biểu diễn: chromosome là biểu diễn nhị phân m có kích thước bằng số đặc trưng TF-IDF.
- Fitness: `macro_F1_val` trừ đi hệ số phạt theo số đặc trưng được chọn $\lambda \|m\|_0$.
- Toán tử: `uniform_crossover`, `bitflip_mutation`, elitism; selection gồm `tournament` và `proportional`.
- Đánh giá: huấn luyện mô hình XGBoost trên đặc trưng đã chọn, báo cáo kết quả thông qua độ đo macro-F1 trên validation và test; trực quan hóa bằng đường hội tụ và confusion matrix.



Hình 23.9: Minh họa trực quan các vị trí/bit trên mask TF-IDF tối ưu sau khi thực hiện GA.

3b – GA tinh chỉnh siêu tham số XGBoost

- Biểu diễn: chromosome tham số hỗn hợp [real/int/categorical] gồm các trường như `n_estimators`, `max_depth`, `learning_rate`, `subsample`, `colsample_bytree`, `reg_lambda` (xem danh sách chính xác trong mã).
- Fitness: macro-F1 trên validation (có thể dùng trung bình k-fold nếu trong mã có hỗ trợ).
- Toán tử: crossover theo biểu diễn cho tham số số thực/nguyên, hoán đổi giá trị cho tham số rời rạc; mutation kiểu Gaussian cho tham số số thực (kép về miền hợp lệ), lật sang lựa chọn khác cho tham số rời rạc; elitism; selection gồm `tournament` và `proportional`.
- Đánh giá: báo cáo macro-F1 validation và test, in bộ tham số tốt nhất, vẽ đường hội tụ.



Hình 23.10: Minh họa kết quả fitness giữa mô hình baseline với bộ siêu tham số mặc định và mô hình với bộ siêu tham số được lựa chọn bởi GA.

23.3 Câu hỏi trắc nghiệm

1. Trong một vòng lặp GA điển hình, trình tự thao tác hợp lý nhất là:
 - (a) Chọn cá thể cha mẹ → đánh giá fitness → lai ghép → đột biến → tạo quần thể mới.
 - (b) Lai ghép → đột biến → chọn cá thể cha mẹ → đánh giá fitness → tạo quần thể mới.
 - (c) Đánh giá fitness → chọn cá thể cha mẹ → lai ghép → đột biến → tạo quần thể mới (kèm elitism nếu có).
 - (d) Chuẩn hoá dữ liệu → đánh giá fitness → huấn luyện mô hình sâu → tạo quần thể mới.
2. Với hàm Trap theo block kích thước $b = 5$: một block đạt điểm 5 nếu có 5 bit bằng 1, ngược lại nhận điểm $5 - 1 - (\text{số bit } 1)$. Tính fitness của chuỗi gồm hai block
 $[1, 1, 1, 0, 0 | 1, 1, 1, 1, 1]$.
 - (a) 4.
 - (b) 5.
 - (c) 6.
 - (d) 7.
3. Phát biểu nào sau đây là đúng khi so sánh Tournament Selection và Proportional Selection (Roulette-Wheel)?
 - (a) Tournament luôn đảm bảo chọn cá thể tốt nhất toàn cục ở mọi vòng.
 - (b) Proportional có thể bị chi phối mạnh bởi chênh lệch fitness, dễ mất đa dạng khi chênh lệch quá lớn.
 - (c) Tournament không phụ thuộc vào tham số nào.
 - (d) Proportional không thể chọn cá thể có fitness thấp.
4. Chọn phát biểu đúng về mutation trong GA:
 - (a) Bit-flip mutation phù hợp cho biểu diễn nhị phân; Gaussian mutation phù hợp cho vector thực.

- (b) Gaussian mutation chỉ áp dụng cho bài toán phân loại.
- (c) Bit-flip mutation làm thay đổi giá trị gene theo phân phối chuẩn.
- (d) Cả hai loại mutation đều không ảnh hưởng đến khả năng thoát cực trị cục bộ.
5. Cho ba cá thể có fitness lần lượt là 2, 3, 5. Dùng Proportional Selection, xác suất chọn cá thể có fitness 3 bằng:
- (a) 0.20.
- (b) 0.30.
- (c) 0.40.
- (d) 0.50.
6. Xét đoạn code sau:

Hàm proportional_select()

```

1 def proportional_select(population, fitness_scores):
2     total_fitness = sum(fitness_scores)
3     if total_fitness == 0:
4         return random.choice(population)
5     threshold = random.uniform(0, total_fitness)
6     cumulative = 0.0
7     for individual, fitness in zip(population, fitness_scores):
8         cumulative += fitness
9         if cumulative >= threshold:
10             return individual
11     return population[-1]

```

Hãy cho biết kết quả trả về của hàm khi:

`population = [[1, 0, 1], [0, 1, 0], [1, 1, 1]], fitness_scores = [2, 3, 5],`

và giá trị ngẫu nhiên lấy từ `random.uniform(0, total_fitness)` giả định bằng 4.7.

- (a) [1, 0, 1].
- (b) [0, 1, 0].

- (c) [1, 1, 1].
 (d) Không xác định vì còn phụ thuộc seed.

7. Xét đoạn code sau:

Hàm tournament_select()

```

1 def tournament_select(population, fitness_scores, tournament_size
                      =2):
2     if not population:
3         raise ValueError("Population must contain at least one
                           individual.")
4
5     k = min(tournament_size, len(population))
6     candidate_indices = random.sample(range(len(population)), k=k)
7     best_idx = max(candidate_indices, key=lambda idx:
                      fitness_scores[idx])
8     return population[best_idx]

```

Hãy cho biết hàm trên sẽ trả về giá trị gì trong tình huống sau:

`population = [A, B, C, D], fitness_scores = [3, 7, 5, 2], tournament_size = 3,`
 và giả sử `random.sample(range(4), k=3)` trả về [0, 2, 3].

- (a) A.
 (b) B.
 (c) C.
 (d) D.

8. Xét đoạn code sau:

Chuyển loss thành xác suất chọn trong Proportional Selection

```

1 def selection_probs_from_losses(losses, eps=1e-8):
2     inv = 1.0 / (np.array(losses) + eps)
3     p = inv / np.sum(inv)
4     return p.tolist()

```

Hãy cho biết xác suất chọn (làm tròn 4 chữ số thập phân) khi

$$\text{losses} = [4.0, 1.0, 5.0].$$

- (a) [0.2000, 0.6000, 0.2000].
- (b) [0.2500, 0.5000, 0.2500].
- (c) [0.1000, 0.8000, 0.1000].
- (d) [0.1724, 0.6897, 0.1379].

9. Xét đoạn code sau:

Hàm proportional_select() cho feature selection

```

1 def proportional_select(population, scores):
2     shifted = np.array(scores) - min(scores)
3     probs = shifted + 1e-6
4     probs /= probs.sum()
5     pick = np.random.choice(len(population), p=probs)
6     return population[pick]

```

Hãy cho biết vector xác suất probs (làm tròn 4 chữ số thập phân) khi:

$$\text{scores} = [0.2, 0.5, 0.5].$$

- (a) [0.2000, 0.5000, 0.3000].
- (b) [0.0000, 0.5000, 0.5000].
- (c) [0.1667, 0.4167, 0.4167].
- (d) [0.2500, 0.3750, 0.3750].

10. Xét đoạn code sau:

Giải mã chromosome tham số decode_hp

```

1 HP_RANGES = {
2     "n_estimators": (10, 100),
3     "max_depth": (2, 10),
4     "learning_rate": (0.01, 0.3),
5     "subsample": (0.5, 1.0),

```

```

6      "colsample_bytree": (0.5, 1.0),
7      "reg_lambda": (0.0, 2.0),
8  }
9 def decode_hp(individual):
10    params = {}
11    for value, key in zip(individual, HP_RANGES):
12        low, high = HP_RANGES[key]
13        val = low + value * (high - low)
14        params[key] = int(val) if key in {"n_estimators", "max_depth"} else float(val)
15    params["n_estimators"] = max(10, int(params["n_estimators"]))
16    params["max_depth"] = max(2, int(params["max_depth"]))
17    params["subsample"] = float(np.clip(params["subsample"], *
18                                  HP_RANGES["subsample"]))
19    params["colsample_bytree"] = float(np.clip(params[
19                                  "colsample_bytree"], *HP_RANGES["colsample_bytree"]))
20
21    return params

```

Giả sử `individual = [0, 1, 0.5, 0.5, 0, 1]`. Bộ tham số giải mã được là gì?

- (a) • `n_estimators` = 10.
 • `max_depth` = 10.
 • `learning_rate` = 0.155.
 • `subsample` = 0.75.
 • `colsample_bytree` = 0.50.
 • `reg_lambda` = 2.0.
- (b) • `n_estimators` = 55.
 • `max_depth` = 6.
 • `learning_rate` = 0.155.
 • `subsample` = 0.50.
 • `colsample_bytree` = 0.75.
 • `reg_lambda` = 1.0.
- (c) • `n_estimators` = 10.
 • `max_depth` = 2.
 • `learning_rate` = 0.300.

- `subsample` = 1.00.
 - `colsample_bytree` = 0.50.
 - `reg_lambda` = 2.0.
- (d)
 - `n_estimators` = 100.
 - `max_depth` = 10.
 - `learning_rate` = 0.155.
 - `subsample` = 0.75.
 - `colsample_bytree` = 1.00.
 - `reg_lambda` = 0.0.

1. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
2. **Solution:** Các file code đáp án có thể được tải tại [đây](#).
3. **Rubric:**

Mục	Kiến thức	Đánh giá
I.	<ul style="list-style-type: none"> - Biểu diễn nhị phân, toán tử GA cơ bản (crossover, mutation, elitism). - Khác biệt OneMax vs. Trap; selection: tournament vs. proportional. 	<ul style="list-style-type: none"> - Hoàn thiện hàm cần thiết; chạy đủ kịch bản; báo cáo đường hội tụ. - Nhận xét ngắn gọn về ảnh hưởng selection, mutation, elitism.
II.	<ul style="list-style-type: none"> - Biểu diễn thực cho tham số hồi quy; dự đoán tuyến tính; MSE làm thước đo. - Gaussian mutation và so sánh selection. 	<ul style="list-style-type: none"> - Điền đúng các hàm cốt lõi; vẽ lịch sử MSE; minh họa \hat{y} vs. y. - So sánh ngắn gọn tốc độ/độ chính xác giữa các cấu hình.
III.	<ul style="list-style-type: none"> - TF-IDF và chọn đặc trưng bằng mặt nạ; penalty $\lambda \cdot \ \mathbf{m}\ _0$. - GA cho tham số XGBoost, macro-F1 (CV/validation). 	<ul style="list-style-type: none"> - Báo cáo F1, số đặc trưng (3a) và bộ tham số tốt nhất (3b). - So sánh với baseline; cung cấp biểu đồ hội tụ/ngắn gọn.

4. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại [đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 tiếng.

AIO_QAs-Verified

🔒 Private group · 1.4K members



Hình 23.11: Hình ảnh group facebook AIO Q&A.

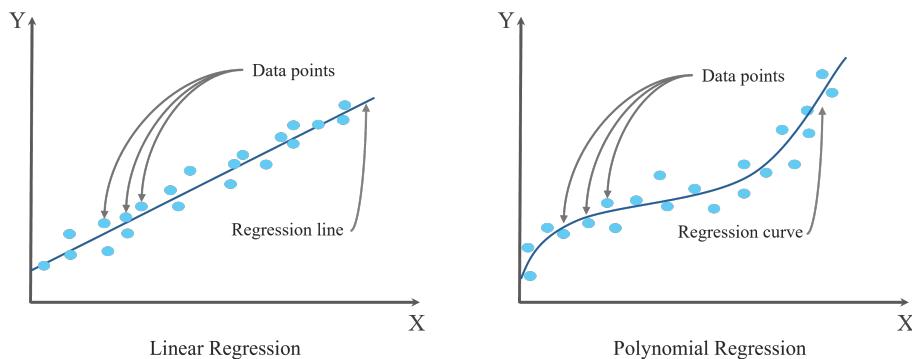
Chương 24

Project 1: Đánh giá các thuật toán trên bài toán xâm nhập mặn (Linear and non-linear regressions)

24.1 Giới thiệu

Hồi quy tuyến tính (Linear Regression) là một trong những mô hình nền tảng và quan trọng nhất trong học máy, được sử dụng để dự đoán các giá trị liên tục dựa trên mối quan hệ giữa các biến đầu vào và đầu ra. Trong quá trình huấn luyện, mô hình điều chỉnh các tham số để giảm sai số giữa giá trị dự đoán và giá trị thực, thông qua các thuật toán tối ưu như Gradient Descent. Mô hình dần học được mối quan hệ tuyến tính tối ưu nhất thể hiện xu hướng của dữ liệu.

Tuy nhiên, dữ liệu trong thực tế thường không hoàn toàn tuyến tính. Nhiều hiện tượng có mối quan hệ phi tuyến và phức tạp hơn, khiến mô hình tuyến tính đơn thuần không còn phù hợp. Khi đó, việc mở rộng sang các mô hình hồi quy phi tuyến, chẳng hạn như hồi quy đa thức Polynomial Regression), giúp mô hình học được các mối quan hệ linh hoạt và chính xác hơn.



Hình 24.1: Hình ảnh minh họa về Linear Regression và Polynomial Regression.

Bên cạnh đó, khi mô hình trở nên phức tạp hơn, nguy cơ quá khớp (overfitting)

cũng tăng lên, làm giảm khả năng tổng quát hóa của mô hình trên dữ liệu mới. Để khắc phục điều này, các kỹ thuật điều chuẩn (Regularization) như Ridge Regression (L2) và Lasso Regression (L1) được sử dụng nhằm kiểm soát độ lớn của các tham số và duy trì tính ổn định của mô hình.

Trong project này, chúng ta sẽ tìm hiểu chi tiết hơn về hồi quy đa thức (Polynomial Regression), cũng như hai phương pháp điều chuẩn phổ biến là Ridge Regression (L2) và Lasso Regression (L1).

Để việc đọc thuận tiện hơn, bảng sau liệt kê và chuẩn hoá các ký hiệu sẽ xuất hiện xuyên suốt nội dung bài.

Bảng Ký hiệu Toán học	
Ký hiệu	Ý nghĩa
x	Biến đầu vào (trường hợp 1 biến).
y	Giá trị mục tiêu (giá trị thực).
y_i	Giá trị mục tiêu của mẫu thứ i .
\hat{y}_i	Giá trị dự đoán cho mẫu thứ i .
w_0	Hệ số chặn (intercept) của mô hình.
w_j	Hệ số ứng với bậc j của x ($j = 1, \dots, n$).
w	Vector hệ số $[w_0, \dots, w_n]^\top$.
ϵ	Sai số ngẫu nhiên (noise).
n	Bậc đa thức (số mũ cao nhất trong mô hình).
m	Số mẫu dữ liệu (dùng trong tổng $\sum_{i=1}^m$).
i	Chỉ số mẫu trong các tổng.
j	Chỉ số hệ số/đặc trưng trong các tổng.
λ	Hệ số điều chỉnh: kiểm soát độ lớn hệ số, giảm overfitting.
$J(w)$	Hàm mất mát: giá trị sai số để tối ưu.
$\sum_{i=1}^m$	Ký hiệu tổng cộng dồn theo chỉ số từ 1 đến m .

24.2 Mô hình Polynomial Regression

Polynomial Regression là sự mở rộng của hồi quy tuyến tính, trong đó mô hình không chỉ xem xét mối quan hệ tuyến tính giữa biến đầu vào và đầu ra, mà còn thêm vào các bậc cao hơn của biến đầu vào. Điều này cho phép mô hình học được các mối quan hệ phi tuyến phức tạp hơn, giúp đường hồi quy có thể uốn cong theo xu hướng của dữ liệu thực tế thay vì chỉ là một đường thẳng.

Giả sử biến đầu vào là x , mô hình hồi quy đa thức được mô tả như sau:

$$y = w_0 + w_1x + w_2x^2 + \cdots + w_nx^n + \epsilon$$

Trong đó ϵ là sai số ngẫu nhiên (noise). Khi bậc n tăng lên, mô hình có khả năng mô tả dữ liệu tốt hơn, nhưng đồng thời cũng có nguy cơ học quá kỹ dữ liệu huấn luyện (overfitting), khiến mô hình hoạt động kém trên dữ liệu mới. Vì vậy, việc lựa chọn bậc đa thức phù hợp là rất quan trọng để đạt được sự cân bằng giữa độ chính xác và khả năng tổng quát hóa.

Data		Create polynomial feature		
Level	Salary	Input	1	$\psi(\varphi(i))$
0	45000	0	1	0
1	50000	1	1	1
2	60000	2	1	2
3	80000	3	1	3
4	110000	4	1	4
5	160000	5	1	5

Hình 24.2: Ví dụ về cách tạo ra các đặc trưng đa thức trong mô hình Polynomial Regression.

24.3 Mô hình Ridge Regression (L2)

Ridge Regression được phát triển để khắc phục hiện tượng quá khớp trong hồi quy tuyến tính, đặc biệt khi dữ liệu có nhiều đặc trưng hoặc các đặc

trưng có tương quan cao (multicollinearity). Phương pháp này bổ sung một thành phần phạt (regularization term) L2 vào hàm mất mát, nhằm kiểm soát độ lớn của các hệ số trong mô hình.

Hàm mất mát của Ridge Regression được viết như sau:

$$J(w) = \sum_{i=1}^m (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^n w_j^2$$

Trong đó, λ là hệ số điều chỉnh (regularization parameter) quy định mức độ phạt cho các trọng số lớn. Khi λ tăng, mô hình càng bị ràng buộc chặt hơn, khiến các hệ số w_j bị “co nhỏ” về gần 0.

Nhờ cơ chế này, Ridge giúp mô hình trở nên ổn định hơn, giảm độ nhạy với nhiễu trong dữ liệu và tránh việc học quá chi tiết những biến động nhỏ trong tập huấn luyện. Tuy nhiên, khác với Lasso, Ridge không loại bỏ hoàn toàn đặc trưng nào (các hệ số không trở về 0), mà chỉ làm giảm ảnh hưởng của chúng. Do đó, Ridge thường được dùng khi ta muốn giữ lại tất cả đặc trưng nhưng hạn chế ảnh hưởng của những đặc trưng kém quan trọng.

24.4 Mô hình Lasso Regression (L1)

Lasso Regression (Least Absolute Shrinkage and Selection Operator) cũng được sử dụng để tránh overfitting, nhưng thay vì phạt bình phương của trọng số như Ridge, nó sử dụng giá trị tuyệt đối của trọng số (L1 penalty). Nhờ vậy, mô hình có đặc tính đặc biệt là có thể đưa một số hệ số về đúng 0, tức là loại bỏ hoàn toàn các đặc trưng ít quan trọng.

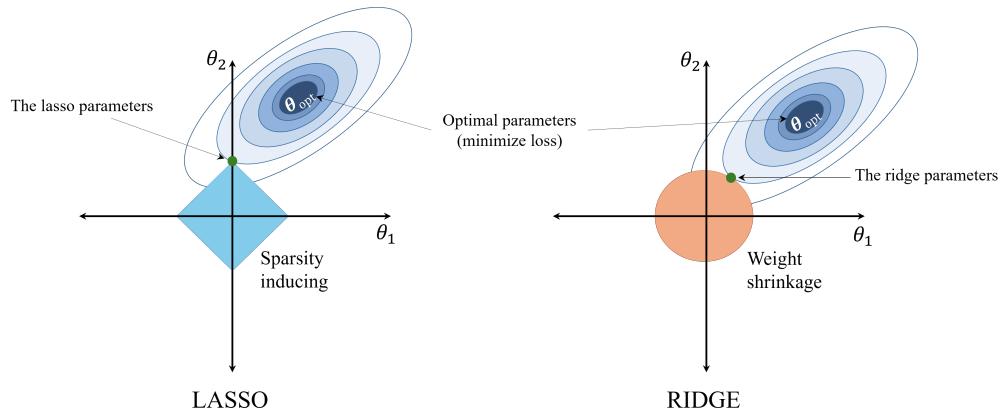
Hàm mất mát của Lasso được mô tả như sau:

$$J(w) = \sum_{i=1}^m (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^n |w_j|$$

Thành phần phạt $\lambda \sum |w_j|$ giúp mô hình vừa giới hạn độ lớn của các trọng số, vừa tự động lựa chọn những đặc trưng quan trọng nhất. Khi λ lớn, nhiều trọng số sẽ bị triệt tiêu về 0, làm cho mô hình đơn giản và dễ hiểu hơn.

Khác với Ridge, Lasso mang lại mô hình “thưa” (sparse model), trong đó chỉ còn lại một số ít đặc trưng có ảnh hưởng đáng kể. Vì vậy, Lasso rất hữu ích

khi làm việc với dữ liệu có nhiều đặc trưng nhưng chỉ một phần nhỏ thực sự liên quan đến đầu ra, giúp tăng khả năng giải thích và giảm nhiễu hiệu quả.



Hình 24.3: So sánh ý nghĩa hình học của Lasso Regression (L1) và Ridge Regression (L2).

24.5 Xây dựng mô hình hồi quy nâng cao cho bài toán dự đoán giá nhà

Ở phần này ta tập trung xây dựng mô hình dự đoán giá nhà dựa trên tập dữ liệu chứa nhiều đặc trưng mô tả thông tin về ngôi nhà, khu đất và điều kiện bán (như diện tích, vị trí, kiểu nhà, năm xây dựng, tình trạng bán, ...).

Mục tiêu là ước lượng chính xác giá bán (*SalePrice*) bằng cách áp dụng các kỹ thuật hồi quy nâng cao, bao gồm **Polynomial Regression**, **Ridge Regression** và **Lasso Regression**, nhằm mô hình hóa các quan hệ phi tuyến và giảm thiểu hiện tượng *overfitting*.

Bước 1: Import thư viện và tải dữ liệu

```
1 !gdown 1Dh_y7gFDUa2sD72_cK1a209dhbMVoGEd
```

```
1 import numpy as np
2 import pandas as pd
3 import seaborn as sns
4 import matplotlib.pyplot as plt
5
6 import warnings
7 warnings.filterwarnings("ignore")
8
9 #importing Dataset
10 house_df = pd.read_csv("/content/train-house-prices-advanced-
11 regression-techniques.csv")
12
13 #checking dataset
14 house_df.head()
```

Trong bước đầu tiên, các thư viện cần thiết cho việc xử lý dữ liệu, trực quan hóa và xây dựng mô hình được import, bao gồm NumPy, Pandas, Seaborn và Matplotlib.

Sau đó, tập dữ liệu *House Prices - Advanced Regression Techniques* được tải

lên và đọc bằng hàm `read_csv()` từ thư viện Pandas.

Cuối cùng, một số dòng đầu tiên của dữ liệu được hiển thị để kiểm tra cấu trúc tổng quan của tập dữ liệu.

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.0	8450	Pave	Nan	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.0	9600	Pave	Nan	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.0	11250	Pave	Nan	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.0	9550	Pave	Nan	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	5	60	RL	84.0	14260	Pave	Nan	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

5 rows × 81 columns

Hình 24.4: Một vài mẫu dữ liệu của dataset House Prices

Bước 2: Khám phá dữ liệu

```

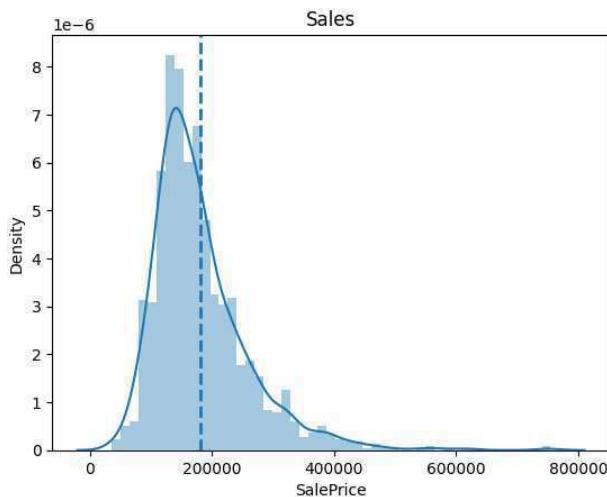
1 house_df.shape
2 house_df.describe()
3
4 # target variable
5 sns.distplot(house_df["SalePrice"])
6 plt.axvline(x=house_df["SalePrice"].mean(), linestyle="--",
              linewidth=2)
7 plt.title("Sales")

```

Trong bước này, tiến hành khám phá dữ liệu (EDA) nhằm hiểu rõ hơn về cấu trúc và đặc điểm của tập dữ liệu.

Cụ thể, hàm `shape` được sử dụng để xác định kích thước của bộ dữ liệu, trong khi `describe()` cung cấp các thống kê mô tả cơ bản như giá trị trung bình, độ lệch chuẩn, và các phân vị.

Biến mục tiêu `SalePrice` được trực quan hóa thông qua biểu đồ phân phối nhằm quan sát xu hướng và mức độ phân tán của giá nhà. Đường gạch ngang biểu thị giá trị trung bình, giúp nhận biết sự chênh lệch và độ lệch của phân phối dữ liệu.



Hình 24.5: Biểu đồ phân phối nhầm quan sát xu hướng và mức độ phân tán của giá nhà.

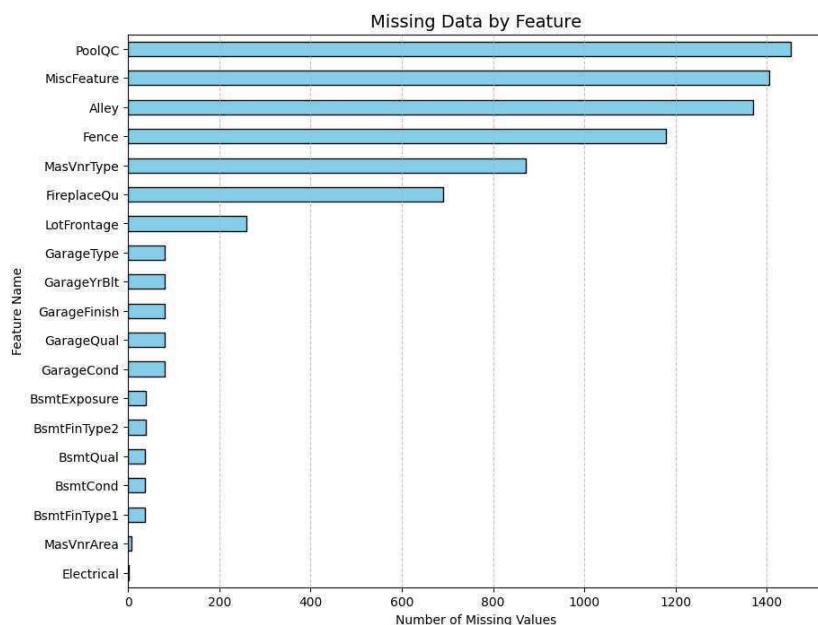
```

1 # Checking missing values (horizontal view)
2 missing = house_df.isnull().sum()
3 missing = missing[missing > 0]
4 missing = missing.sort_values(ascending=False)
5
6 plt.figure(figsize=(10, 8))
7 missing.plot.bart(color="skyblue", edgecolor="black")
8 plt.title("Missing Data by Feature", fontsize=14)
9 plt.xlabel("Number of Missing Values")
10 plt.ylabel("Feature Name")
11 plt.grid(axis="x", linestyle="--", alpha=0.7)
12 plt.gca().invert_yaxis()
13 plt.show()

```

Biểu đồ cột ngang dưới đây thể hiện số lượng giá trị bị thiếu trong từng đặc trưng của tập dữ liệu. Trục tung biểu diễn tên các thuộc tính, trong khi trục hoành biểu diễn số lượng mẫu bị khuyết tương ứng.

Việc sắp xếp dữ liệu theo thứ tự giảm dần giúp nhanh chóng nhận diện các đặc trưng có tỉ lệ thiếu dữ liệu cao, từ đó hỗ trợ việc lựa chọn chiến lược xử lý phù hợp như loại bỏ hoặc thay thế giá trị thiếu trong giai đoạn tiền xử lý.



Hình 24.6: Trực quan hóa số lượng các giá trị bị thiếu trong tập dữ liệu.

```

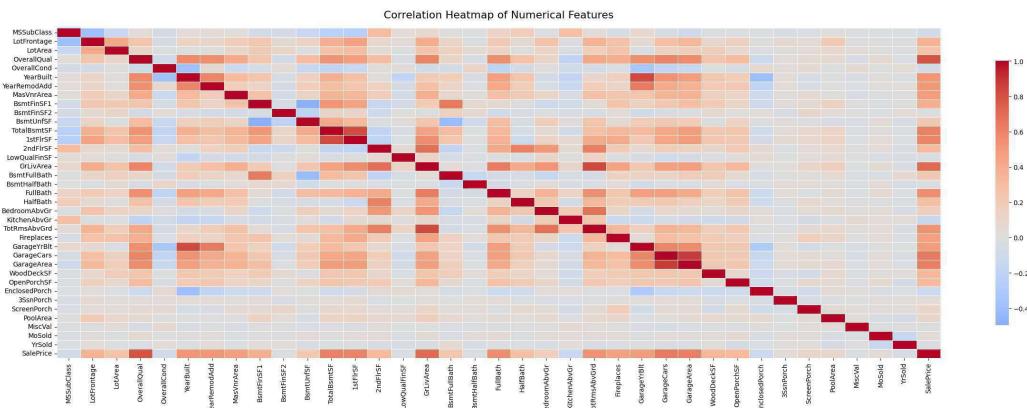
1 plt.figure(figsize=(30, 9))
2 sns.heatmap(
3     house_df.corr(numeric_only=True),
4     cmap="coolwarm",
5     linewidths=0.5,
6     center=0,
7     cbar_kws={"shrink": 0.8}
8 )
9 plt.title("Correlation Heatmap of Numerical Features", fontsize=16,
10           pad=15)
11 plt.show()

```

Biểu đồ ma trận tương quan (*correlation heatmap*) dưới đây thể hiện mức độ tương quan giữa các biến số trong tập dữ liệu. Các giá trị tương quan được mã hóa bằng thang màu, trong đó tông xanh biểu thị tương quan âm và tông đỏ biểu thị tương quan dương.

Việc trực quan hóa tương quan giúp nhận diện các đặc trưng có mối quan hệ chặt chẽ với biến mục tiêu **SalePrice**, từ đó hỗ trợ việc lựa chọn và đánh

giá các đặc trưng quan trọng trong quá trình xây dựng mô hình.



Hình 24.7: Biểu đồ ma trận tương quan giữa các biến số trong tập dữ liệu.

```

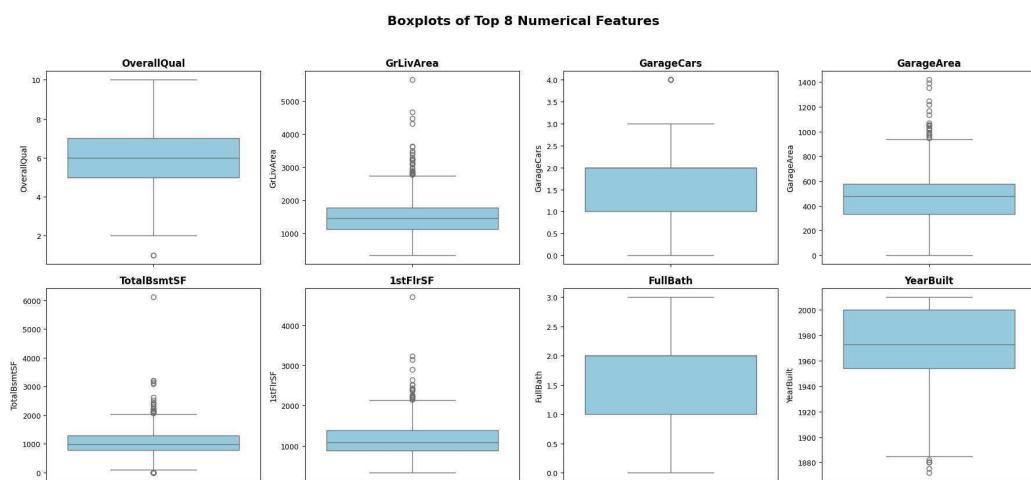
1 # Chọn 8 đặc trưng numeric quan trọng nhất dựa trên tương quan với
   SalePrice
2 important_cols = [
3     "OverallQual", "GrLivArea", "GarageCars", "GarageArea",
4     "TotalBsmtSF", "1stFlrSF", "FullBath", "YearBuilt"
5 ]
6
7 fig, axes = plt.subplots(2, 4, figsize=(18, 8))
8 axes = axes.flatten()
9
10 for i, col in enumerate(important_cols):
11     sns.boxplot(data=house_df, y=col, ax=axes[i], color="skyblue")
12     axes[i].set_title(col, fontsize=12, fontweight="bold")
13     axes[i].tick_params(labelsize=9)
14
15 for i in range(len(important_cols), len(axes)):
16     axes[i].set_visible(False)
17
18 plt.suptitle("Boxplots of Top 8 Numerical Features", fontsize=16,
19               fontweight="bold", y=1.03)
20 plt.tight_layout()
21 plt.show()

```

Biểu đồ hộp (*boxplot*) dưới đây thể hiện phân phối giá trị của tám đặc trưng

số có tương quan mạnh nhất với biến mục tiêu **SalePrice**, bao gồm các yếu tố như chất lượng tổng thể, diện tích sinh hoạt, số lượng chỗ để xe, diện tích tầng hầm và năm xây dựng.

Các biểu đồ giúp quan sát phạm vi giá trị, xu hướng phân phối cũng như phát hiện các giá trị ngoại lệ (*outliers*) trong từng thuộc tính. Điều này hỗ trợ đánh giá đặc trưng nào có khả năng ảnh hưởng lớn đến giá nhà và cần được xử lý phù hợp trong bước tiền xử lý dữ liệu.



Hình 24.8: Trực quan hóa phân phối giá trị của tám đặc trưng số có tương quan mạnh nhất với biến mục tiêu SalePrice

Bước 3: Tiền xử lý dữ liệu

```

1 # droping values more than 50/100
2 house_df = house_df.drop(["Id", "Alley", "PoolQC", "Fence", "MiscFeature"], axis=1)

```

Trong bước này, các thuộc tính có tỷ lệ giá trị bị thiếu vượt quá 50% được loại bỏ khỏi tập dữ liệu, bao gồm các cột Id, Alley, PoolQC, Fence và MiscFeature.

Việc loại bỏ này giúp giảm nhiễu và tránh ảnh hưởng tiêu cực đến quá trình

huấn luyện mô hình, đồng thời giữ lại những đặc trưng có thông tin đủ mạnh và đáng tin cậy hơn.

```

1 # create training and validation sets
2 from sklearn.model_selection import train_test_split
3
4 train_df, test_df = train_test_split(
5     house_df,
6     test_size=0.25,
7     random_state=42
8 )
9
10 y_train = train_df["SalePrice"].values
11 y_test = test_df["SalePrice"].values
12 train_df = train_df.drop(["SalePrice"], axis=1)
13 test_df = test_df.drop(["SalePrice"], axis=1)
14
15 num_cols = [col for col in train_df.columns if train_df[col].dtype
16             in ["float64", "int64"]]
17 cat_cols = [col for col in train_df.columns if train_df[col].dtype
18             not in ["float64", "int64"]]
19
20 # fill none for categorical columns
21 train_df[cat_cols] = train_df[cat_cols].fillna("none")
22 test_df[cat_cols] = test_df[cat_cols].fillna("none")

```

Dữ liệu được chia thành hai phần: *tập huấn luyện* và *tập kiểm tra* với tỷ lệ 75% – 25% nhằm đánh giá khả năng tổng quát hóa của mô hình trên dữ liệu chưa từng thấy. Biến mục tiêu SalePrice được tách riêng khỏi các đặc trưng đầu vào để phục vụ quá trình huấn luyện.

Các thuộc tính được phân loại thành hai nhóm: đặc trưng dạng số (*numerical*) và đặc trưng dạng phân loại (*categorical*). Đối với các cột phân loại, các giá trị khuyết được thay thế bằng nhãn "none" nhằm đảm bảo tính nhất quán của dữ liệu đầu vào trước khi mã hóa.

```

1 from sklearn.preprocessing import OneHotEncoder
2 from sklearn.impute import SimpleImputer
3
4 # One-hot encode categorical columns

```

```

5 encoder = OneHotEncoder(handle_unknown="ignore", sparse_output=False
6                                     )
7 encoder.fit(train_df[cat_cols])
8 encoded_cols = list(encoder.get_feature_names_out(cat_cols))
9
10 train_df[encoded_cols] = encoder.transform(train_df[cat_cols])
11 test_df[encoded_cols] = encoder.transform(test_df[cat_cols])
12
13 imputer = SimpleImputer()
14 train_df[num_cols] = imputer.fit_transform(train_df[num_cols])
15 test_df[num_cols] = imputer.transform(test_df[num_cols])

```

Trong bước này, dữ liệu được tiền xử lý để đảm bảo tính đầy đủ và định dạng phù hợp cho mô hình học máy. Các đặc trưng dạng phân loại (*categorical features*) được mã hóa bằng phương pháp **One-Hot Encoding**, chuyển đổi từng giá trị rời rạc thành các biến nhị phân. Tùy chọn `handle_unknown="ignore"` được sử dụng nhằm bỏ qua các nhãn chưa từng xuất hiện trong tập huấn luyện, tránh gây lỗi trong quá trình biến đổi dữ liệu kiểm tra.

Đối với các đặc trưng dạng số (*numerical features*), các giá trị bị khuyết được xử lý bằng kỹ thuật **Simple Imputer**, với cơ chế thay thế giá trị thiếu bằng trung bình của cột tương ứng. Việc này giúp giảm thiểu mất mát thông tin và đảm bảo toàn bộ dữ liệu đầu vào đều sẵn sàng cho giai đoạn huấn luyện mô hình.

```

1 from sklearn.preprocessing import MinMaxScaler
2
3 scaler = MinMaxScaler()
4 train_num_features = scaler.fit_transform(train_df[num_cols])
5 test_num_features = scaler.transform(test_df[num_cols])
6
7 X_train = np.hstack([train_num_features, train_df[encoded_cols].values])
8 X_test = np.hstack([test_num_features, test_df[encoded_cols].values])

```

Để đảm bảo các đặc trưng dạng số có cùng thang đo, dữ liệu được chuẩn

hóa bằng kỹ thuật **Min-Max Scaling**. Phương pháp này biến đổi giá trị của từng đặc trưng về khoảng [0, 1], giúp tăng tốc độ hội tụ của mô hình và tránh việc các thuộc tính có giá trị lớn chi phối kết quả huấn luyện.

Sau khi chuẩn hóa, các đặc trưng dạng số và đặc trưng đã được mã hóa (*One-Hot Encoded*) được ghép lại thành ma trận đặc trưng đầu vào hoàn chỉnh (X_{train} , X_{test}), sẵn sàng cho quá trình huấn luyện mô hình hồi quy.

Bước 4: Huấn luyện model không sử dụng đặc trưng đặc

```
1 from sklearn.linear_model import LinearRegression, Ridge, Lasso
2 from sklearn.metrics import mean_squared_error, r2_score
3
4 # Tạo danh sách model
5 models = {
6     ***Your code here***
7 }
8
9 # Khởi tạo list lưu kết quả
10 train_rmse_results = []
11 test_rmse_results = []
12 train_r2_results = []
13 test_r2_results = []
14 model_names = []
15
16 # Huấn luyện và tính metric
17 for name, model in models.items():
18     regressor = ***Your code here***
19
20     # Dự đoán
21     y_train_pred = ***Your code here***
22     y_test_pred = ***Your code here***
23
24     # Tính RMSE
25     train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
26     test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
27
28     # Tính r2
29     train_r2 = r2_score(y_train, y_train_pred)
30     test_r2 = r2_score(y_test, y_test_pred)
```

```

31
32     # Lưu kết quả
33     model_names.append(name)
34     train_rmse_results.append(train_rmse)
35     test_rmse_results.append(test_rmse)
36     train_r2_results.append(train_r2)
37     test_r2_results.append(test_r2)
38
39 # Tạo DataFrame tổng hợp
40 df_results = pd.DataFrame([
41     "Model": model_names,
42     "Train_RMSE": train_rmse_results,
43     "Test_RMSE": test_rmse_results,
44     "Train_R2": train_r2_results,
45     "Test_R2": test_r2_results
46 ]).sort_values(by="Test_R2", ascending=False)
47
48 df_results

```

Trong bước này, bạn sẽ huấn luyện các mô hình hồi quy tuyến tính cơ bản gồm Linear Regression, Ridge Regression và Lasso Regression (chưa sử dụng đặc trưng đa thức).

Hãy hoàn thiện các phần còn thiếu (**Your code here***) để khởi tạo và huấn luyện các mô hình, thực hiện dự đoán trên tập train và test, tính toán các chỉ số đánh giá (RMSE, R²) và tổng hợp kết quả vào bảng DataFrame nhằm so sánh hiệu suất giữa các mô hình.

	Model	Train_RMSE	Test_RMSE	Train_R2	Test_R2
2	Lasso	20032.668558	26362.305805	0.933902	0.900794
0	LinearRegression	20029.220254	28136.997313	0.933925	0.886987
1	Ridge	22488.286583	28587.343381	0.916705	0.883340

Hình 24.9: Kết quả huấn luyện khi không sử dụng PolynomialFeatures

Bước 5: Huấn luyện model sử dụng đặc trưng đa thức

```
1 from sklearn.preprocessing import PolynomialFeatures
2 from sklearn.preprocessing import MinMaxScaler
3 from sklearn.linear_model import LinearRegression, Ridge, Lasso
4 from sklearn.metrics import mean_squared_error, r2_score
5
6 poly_features = PolynomialFeatures(
7     degree=2, interaction_only=True, include_bias=False
8 )
9
10 train_poly_features = ***Your code here***
11 test_poly_features = ***Your code here***
12
13 scaler = MinMaxScaler()
14 train_poly_features = scaler.fit_transform(train_poly_features)
15 test_poly_features = scaler.transform(test_poly_features)
16
17 X_train_poly = np.hstack([train_poly_features, train_df[encoded_cols]
18                           .values])
18 X_test_poly = np.hstack([test_poly_features, test_df[encoded_cols].
19                           values])
19
20 # making dictionary of models
21 models = {
22     ***Your code here***
23 }
24
25 # lists to store results
26 r2_results = []
27 rmse_results = []
28 train_rmse_results = []
29 train_r2_results = []
30 model_names = []
31
32 # training and evaluating each model
33 for name, model in models.items():
34     regressor = ***Your code here***
35
36     # predict
37     y_train_pred = ***Your code here***
38     y_test_pred = ***Your code here***
```

```

40     # metrics
41     train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
42     test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
43     train_r2 = r2_score(y_train, y_train_pred)
44     test_r2 = r2_score(y_test, y_test_pred)
45
46     # store
47     model_names.append(name)
48     train_rmse_results.append(train_rmse)
49     rmse_results.append(test_rmse)
50     train_r2_results.append(train_r2)
51     r2_results.append(test_r2)
52
53
54 # create dataframe
55 df_results = pd.DataFrame({
56     "Model": model_names,
57     "Train_RMSE": train_rmse_results,
58     "Test_RMSE": rmse_results,
59     "Train_R2": train_r2_results,
60     "Test_R2": r2_results
61 }).sort_values(by="Test_R2", ascending=False)
62
63 df_results

```

Trong bước này, bạn sẽ mở rộng mô hình hồi quy bằng cách sử dụng các đặc trưng đa thức (Polynomial Features) để giúp mô hình học được các quan hệ phi tuyến giữa các biến đầu vào và đầu ra.

Hãy hoàn thiện các phần còn thiếu (**Your code here***) để thực hiện việc tạo đặc trưng đa thức, chuẩn hóa dữ liệu, huấn luyện các mô hình Linear Regression, Ridge và Lasso, sau đó đánh giá hiệu suất của chúng thông qua các chỉ số (RMSE, R²) trên cả tập train và test, rồi tổng hợp kết quả vào bảng DataFrame.

	Model	Train_RMSE	Test_RMSE	Train_R2	Test_R2
1	Ridge	16057.215172	27763.366559	0.957533	0.889969
2	Lasso	12063.654479	34402.874562	0.976030	0.831048
0	LinearRegression	9545.514116	55032.412230	0.984993	0.567676

Hình 24.10: Kết quả huấn luyện khi sử dụng PolynomialFeatures



Lưu ý rằng việc thêm đặc trưng đa thức không phải lúc nào cũng giúp mô hình hoạt động tốt hơn. Hiệu quả của nó phụ thuộc vào đặc điểm của dữ liệu và độ phức tạp của mô hình. Một số điểm quan trọng cần ghi nhớ:

- **Nên sử dụng:** Khi dữ liệu có mối quan hệ phi tuyến giữa các biến đầu vào và đầu ra, chẳng hạn như xu hướng cong, tương tác bậc hai giữa các đặc trưng hoặc các quan hệ phức tạp không thể mô tả bằng đường thẳng.
- **Không nên sử dụng:** Khi dữ liệu không có dấu hiệu phi tuyến rõ ràng hoặc số đặc trưng đầu vào đã lớn, việc thêm các đa thức sẽ làm tăng số lượng biến rất nhanh, khiến mô hình trở nên phức tạp, tốn tài nguyên tính toán và dễ dẫn đến **overfitting**.
- **Cách khắc phục:** Khi áp dụng Polynomial Features, nên kết hợp với các kỹ thuật điều chỉnh như **Ridge** (L2) hoặc **Lasso** (L1) để giới hạn độ lớn của hệ số, giúp mô hình ổn định và tổng quát hóa tốt hơn trên dữ liệu kiểm tra.

24.6 Câu hỏi trắc nghiệm

1. (Lý thuyết) Khi nào nên sử dụng Polynomial Features?
 - (a) Khi dữ liệu có quan hệ phi tuyến rõ ràng hoặc có tương tác giữa các biến
 - (b) Khi dữ liệu hoàn toàn tuyến tính
 - (c) Khi muốn giảm số đặc trưng
 - (d) Khi dữ liệu không có nhiễu
2. (Lý thuyết) Lasso Regression (L1) khác Ridge Regression (L2) ở điểm nào?
 - (a) L1 làm các hệ số nhỏ dần về 0, L2 làm một số hệ số bằng 0
 - (b) L1 có thể loại bỏ đặc trưng, L2 chỉ làm co nhỏ hệ số
 - (c) L2 thường cho nghiệm thưa (sparse), L1 thì không
 - (d) L1 và L2 hoàn toàn giống nhau
3. (Lý thuyết) Khi tăng giá trị hệ số điều chỉnh λ trong Ridge/Lasso:
 - (a) Các hệ số bị giảm độ lớn, mô hình đơn giản hơn
 - (b) Các hệ số tăng độ lớn, mô hình phức tạp hơn
 - (c) Không ảnh hưởng đến kết quả
 - (d) Mô hình luôn đạt R^2 cao hơn
4. (Lý thuyết) Tại sao cần chuẩn hóa dữ liệu trước khi áp dụng Ridge hoặc Lasso?
 - (a) Để penalty tác động công bằng lên các đặc trưng khác thang đo
 - (b) Để giảm số chiều dữ liệu
 - (c) Để loại bỏ các đặc trưng nhiễu
 - (d) Để tăng độ phức tạp của mô hình
5. (Tính toán) Với $x = 2$, mô hình $\hat{y} = w_1x + w_2x^2 + b$, biết $w_1 = 1, w_2 = 3, b = 1$. Tính \hat{y} theo công thức $\hat{y} = w_1x + w_2x^2 + b$.

- (a) 13
- (b) 14
- (c) 15
- (d) 16

6. (Tính toán) Với $x_1 = 2, x_2 = 5$, sử dụng đặc trưng tương tác $\phi(x)$ gồm có x_1, x_2, x_1x_2 . Cho $\hat{y} = w^\top \phi(x) + b$ với $w = [1, -1, 0.5]^\top, b = 0$.

- (a) 0
- (b) 1
- (c) 2
- (d) 3

7. (Tính toán) Cho Ridge Regression: $\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, y = 4, \boldsymbol{\theta} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \lambda = 1$.

Công thức: $J(\theta) = \|y - \mathbf{x}^\top \boldsymbol{\theta}\|_2^2 + \lambda \|\theta\|_2^2$.

- (a) 4
- (b) 5
- (c) 6
- (d) 7

8. (Tính toán) Cho Lasso Regression: $\mathbf{x} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, y = 5, \boldsymbol{\theta} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \lambda = 0.5$.

Công thức: $J(\theta) = \|y - \mathbf{x}^\top \boldsymbol{\theta}\|_2^2 + \lambda \|\theta\|_1$.

- (a) 8
- (b) 9
- (c) 10
- (d) 11

9. (Tính toán) Với Ridge Regression với batch: $m = 2, \mathbf{X} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \boldsymbol{\theta} = [1], \lambda = 1$. Công thức: $\nabla_{\boldsymbol{\theta}} J = \frac{2}{m} \mathbf{X}^\top (\mathbf{X} \boldsymbol{\theta} - \mathbf{y}) + 2\lambda \boldsymbol{\theta}$.

- (a) -2

(b) -1

(c) 0

(d) 1

10. (Tính toán) Với $p = 3$ đặc trưng gốc, bậc $d = 2$, `include_bias=False`, `interaction_only=False`. Số đặc trưng sau khi mở rộng đa thức là: $\binom{p+d}{d} - 1$.

(a) 6

(b) 9

(c) 12

(d) 15

1. **Hint:** Các file code gợi ý và dữ liệu nếu có được lưu trong thư mục có thể được tải [tại đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại [đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 giờ.

AIO_QAs-Verified

• Nhóm Riêng tư · 1,4K thành viên



Hình 24.11: Hình ảnh group facebook AIO Q&A

4. Đề xuất phương pháp cải tiến project

- **Tiền xử lý và đảm bảo chất lượng dữ liệu**
 - Xử lý thiếu và ngoại lai theo nguyên tắc mô hình-phù hợp: `KNNImputer` hoặc `IterativeImputer`; chuẩn hoá phân phối bằng `winsorization/IQR`; cân nhắc biến đổi log/Box-Cox/Yeo-Johnson cho các biến (đặc biệt là mục tiêu) có độ lệch lớn.
 - Phân tích các vấn đề khác từ bộ dữ liệu để đưa ra giải pháp phù hợp để tối ưu, xử lý và biến đổi dữ liệu.
- **Biểu diễn đặc trưng (Feature Engineering) có kiểm soát**
 - Đánh giá có hệ thống giữa `PolynomialFeatures` *đầy đủ* và *interaction-only*; lựa chọn bậc đa thức tối ưu bằng `cross-validation` nhằm hạn chế “nổ chiều”.
- **Mô hình hoá và quy chuẩn hoá (Regularization) nâng cao**
 - Bổ sung **Elastic Net** để dung hoà chọn biến (L1) và ổn định hoá hệ số (L2); dò tìm α theo thang log và tối ưu $l1_ratio$ bằng lưới hoặc tìm kiếm ngẫu nhiên.

- Xem xét các mô hình hồi quy *robust* trước ngoại lai: `HuberRegressor`, `QuantileRegressor` (MAE/Pinball loss), `RANSACRegressor`; so sánh đánh giá để cải tiến với Ridge/Lasso/ElasticNet trên cùng tập dữ liệu.
- **Giải thích kết quả mô hình**
 - Diễn giải: *Permutation importance* và/hoặc **SHAP** (đối với mô hình tuyến tính cho kết quả nhất quán với hệ số); tham chiếu *Partial Dependence*/ICE cho các tương tác/phi tuyến được tạo bởi đặc trưng bậc cao.
- **Đánh giá mô hình**
 - Áp dụng **cross-validation** (`KFold`/`RepeatedKFold`; `TimeSeriesSplit` cho dữ liệu theo thời gian); báo cáo RMSE/MAE/ R^2 kèm *khoảng tin cây* bằng bootstrap.

5. Rubric:

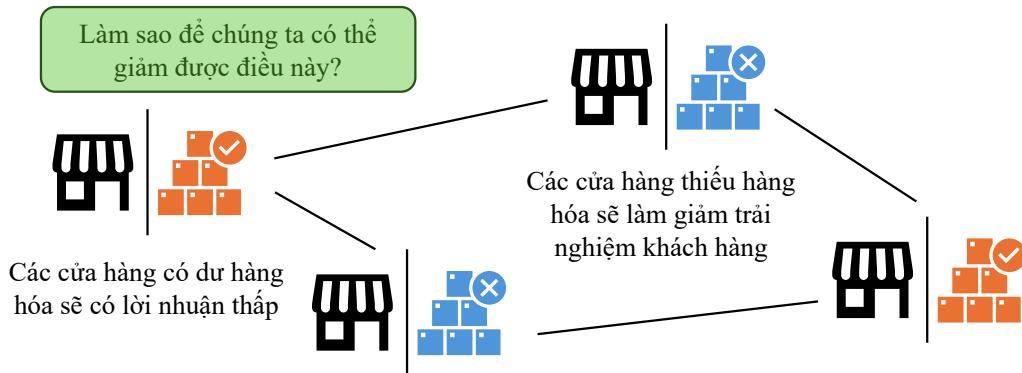
Phần	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Nắm vững kiến thức về linear regression, phân tích được ưu điểm và nhược điểm của mô hình linear regression - Độ đo R^2 đánh giá hiệu quả mô hình regression 	<ul style="list-style-type: none"> - Hoàn thiện code linear regression cho bài toán nhiều giá trị đầu vào - Sử dụng numpy định nghĩa hàm tính độ đo R^2
2.	<ul style="list-style-type: none"> - Nắm vững kiến thức của mô hình nonlinear regression - Nắm vững cách tạo ra các đặc trưng mới trong polynomial regression - Hiểu và vận dụng các kỹ thuật regularization như Lasso Regression (L1) và Ridge Regression (L2) để cải thiện mô hình và giảm overfitting 	<ul style="list-style-type: none"> - Xây dựng mô hình polynomial regression, sử dụng numpy tạo ra các polynomial feature - So sánh mô hình linear regression, polynomial regression, và các biến thể có regularization (Lasso, Ridge) - Thực hiện đánh giá, phân tích tác động của hệ số điều chỉnh λ lên hiệu suất mô hình
3.	<ul style="list-style-type: none"> - Nắm vững một số kỹ thuật chuyển dữ liệu dạng Category thành dữ liệu số: One Hot Encoding, Label Encoding 	<ul style="list-style-type: none"> - Sử dụng thư viện pandas để mã hoá trường dữ liệu dạng Category

Chương 25

Project 2: Xây dựng công cụ tối ưu hoá quy trình phân bổ hàng hoá cho chuỗi cửa hàng thời trang

25.1 Giới thiệu

Trong bối cảnh thị trường bán lẻ cạnh tranh khốc liệt, việc quản lý hàng tồn kho hiệu quả là một trong những yếu tố then chốt quyết định sự thành bại của một doanh nghiệp. Các chuỗi cửa hàng lớn, với mạng lưới hàng trăm chi nhánh, phải đổi mới với bài toán điều phối hàng hóa cực kỳ phức tạp. Sự khác biệt về vị trí địa lý, đặc điểm khách hàng và nhu cầu thị trường tại mỗi điểm bán dẫn đến một thách thức chung: **mất cân bằng tồn kho (inventory imbalance)**. Tình trạng này gây ra hệ quả kép: một số cửa hàng thừa hàng, làm phát sinh chi phí lưu kho và đọng vốn; trong khi các cửa hàng khác lại thiếu hàng, dẫn đến mất doanh thu và suy giảm sự hài lòng của khách hàng.



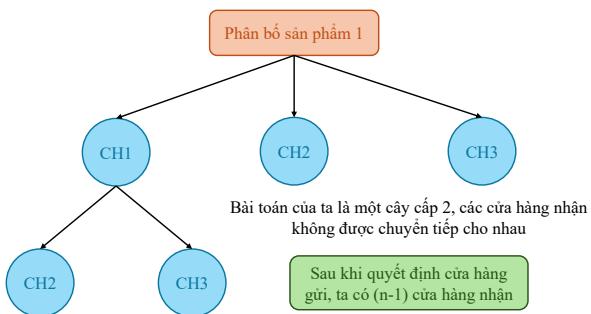
Hình 1: Bối cảnh thực tế của một chuỗi cửa hàng bán lẻ với thách thức về mất cân bằng tồn kho và nhu cầu về một giải pháp chuyển hàng thông minh và nhanh chóng.

Việc điều phối hàng hóa một cách thủ công, dựa nhiều vào kinh nghiệm cá nhân, thường không đủ nhanh nhạy và chính xác để giải quyết triệt để vấn đề. Hình bên phải minh họa sự bùng nổ của các lựa chọn: chỉ với 3 cửa hàng, người quản lý đã phải xem xét $3 \times 2 = 6$ tuyến điều chuyển hàng khác nhau. Nếu áp dụng vào bài toán thực tế với 300 cửa hàng, con số này sẽ là $300 \times 299 = 89,700$ lựa chọn. Việc đánh giá thủ công một khung gian quyết định khổng lồ như vậy là bất khả thi.

	CH1	CH2	CH3
CH1		10	20
CH2	0		0
CH3	0	0	

Hình 2: Sự bùng nổ tổ hợp của các quyết định điều chuyển. Với n cửa hàng, một người quản lý phải cân nhắc $n(n - 1)$ luồng vận chuyển hàng hóa. Với ví dụ $n = 300$, có gần 90,000 quyết định cần được đưa ra.

Do đó, nhu cầu về một hệ thống thông minh, có khả năng tự động hóa và tối ưu hóa các quyết định phân bổ hàng hóa trở nên cấp thiết hơn bao giờ hết. Chuyên đề này sẽ hướng dẫn học viên xây dựng một hệ thống tối ưu hóa phân bổ hàng hóa (Inventory Transfer Optimization System), ứng dụng nội dung bài học của khoa học dữ liệu và giải thuật di truyền (Genetic Algorithm) để giải quyết bài toán thực tiễn này.



Hình 3: Mô hình hóa bài toán phân bổ dưới dạng cây quyết định 2 cấp. Trong thực tế chúng ta có thể góp chuyển chở hàng.

Để hệ thống hóa không gian quyết định này, chúng ta có thể mô hình hóa bài toán dưới dạng một cây quyết định 2 cấp, như được minh họa trong Hình bên. Cách tiếp cận này giúp làm rõ quy trình ra quyết định:

- **Cấp 1 (Chọn cửa hàng gửi):** Đầu tiên, hệ thống phải quyết định sẽ lấy sản phẩm từ cửa hàng nào trong tổng số n cửa hàng.
- **Cấp 2 (Chọn cửa hàng nhận):** Sau khi đã xác định được cửa hàng gửi, hệ thống sẽ tiếp tục chọn một cửa hàng nhận từ $n - 1$ cửa hàng còn lại.

Một giả định quan trọng được thể hiện trong mô hình này là các cửa hàng nhận sẽ không thực hiện chuyển tiếp hàng hóa đi nơi khác. Đây là một bài toán điều chuyển một lượt (**single-hop**), giúp đơn giản hóa vấn đề nhưng vẫn giữ được tính thực tiễn cốt lõi. Cách cấu trúc này là tiền đề để chúng ta xây dựng một mô hình tối ưu hóa chính xác.

Mục tiêu bài viết: Chuyên đề này được thiết kế nhằm giúp học viên hiểu rõ cách ứng dụng khoa học dữ liệu vào một bài toán tối ưu thực tế, thay vì tập trung vào việc xây dựng một hệ thống tối ưu phức tạp. Người học sẽ được tiếp cận hai hướng giải quyết khác nhau là: rule-based và genetic algorithm. Qua đó rèn luyện tư duy phân tích, kỹ năng mô hình hóa và khả năng đánh giá trade-offs giữa các thuật toán.

Nội dung được xây dựng phù hợp cho:

- **Sinh viên ngành AI/Data Science:** Học cách áp dụng optimization

vào bài toán thực tế và thực hành trực tiếp với Pandas/NumPy.

- **Người trái ngành muốn chuyển sang Data/AI:** Làm một dự án thực hành có tính ứng dụng cao, dễ dàng trình bày trong portfolio.
- **Tư vấn viên mới vào nghề (Beginner Consultants):** Hiểu và áp dụng framework tư duy tối ưu, nắm được cách cân nhắc giữa các giải pháp kỹ thuật.

Bài học tập trung vào ý tưởng và phương pháp tiếp cận, mang lại nền tảng vững chắc để người học mở rộng sang các bài toán lớn hơn trong tương lai, đồng thời đảm bảo tuân thủ tính bảo mật của dự án đã triển khai trong thực tế.

Danh sách các module trong project. Để tiện theo dõi, bảng dưới đây tổng hợp các phần chính trong module của chúng ta. Toàn bộ dự án được chứa trong link Github [đây](#).

Bảng Thành phần Project

Thành phần	Ý nghĩa và Chức năng
<code>src/main.py</code>	File thực thi chính của dự án. Chịu trách nhiệm điều phối toàn bộ quy trình: khởi tạo, tạo dữ liệu (nếu cần), chạy phân tích, thực thi các thuật toán tối ưu và lưu kết quả.
<code>src/config.py</code>	Tệp cấu hình trung tâm, chứa tất cả các hằng số và tham số quan trọng như đường dẫn thư mục, nguồn tồn kho an toàn (MIN_INVENTORY_DAYS, MAX_INVENTORY_DAYS), và các tham số cho thuật toán di truyền (kích thước quần thể, tỷ lệ lai ghép, đột biến).
<code>src/engine/analyzer.py</code>	Chứa lớp <code>InventoryAnalyzer</code> . Module này thực hiện các phân tích: tính toán doanh số trung bình hàng ngày, xác định mức tồn kho an toàn, và quan trọng nhất là xác định các sản phẩm đang thừa (<i>excess</i>) và thiếu (<i>needed</i>) tại mỗi cửa hàng cho các bước tiếp theo.
<code>src/engine/rule_based.py</code>	Triển khai thuật toán tối ưu dựa trên luật (<code>RuleBasedOptimizer</code>). Đây là một phương pháp tiếp cận tham lam (greedy), ưu tiên đáp ứng các nhu cầu từ nguồn cung có chi phí vận chuyển thấp nhất.
<code>src/engine/genetic_algorithm.py</code>	Triển khai thuật toán di truyền (<code>GeneticAlgorithmOptimizer</code>). Đây là phương pháp tối ưu hóa phức tạp hơn, tìm kiếm giải pháp tối ưu toàn cục bằng cách mô phỏng quá trình tiến hóa tự nhiên (chọn lọc, lai ghép, đột biến).

25.2 Lý thuyết

25.2.1 Tại sao cần tối ưu hóa phân bổ hàng hóa?

Trước khi đi vào giải pháp kỹ thuật, chúng ta cần hiểu rõ các vấn đề mà một doanh nghiệp phải đối mặt khi hệ thống phân bổ hàng hóa không hiệu quả. Như được tóm tắt trong Hình 4, có bốn động lực chính thúc đẩy nhu cầu cấp thiết này.

Quay trở lại vấn đề, như được tóm tắt trong Hình 4, có bốn động lực chính thúc đẩy nhu cầu cấp thiết về tối ưu hóa phân bổ hàng hóa.

Vấn đề thực tế	Chi phí cao
Một số cửa hàng thừa hàng, gây lãng phí; số khác lại thiếu hàng, bỏ lỡ doanh thu. <i>Cửa hàng A tồn đọng 100 sản phẩm, cửa hàng B hết sạch</i>	Tồn kho quá nhiều dẫn đến chi phí lưu trữ lớn. Vận chuyển không hiệu quả làm tăng chi phí hoạt động. <i>Tồn kho dư thừa gây lãng phí vốn lưu động</i>
Quyết định chậm	Ứng dụng Data Science
Việc phân bổ hàng hóa phụ thuộc nhiều vào kinh nghiệm cá nhân, dẫn đến quyết định chậm và thiếu chính xác. <i>Quyết định thủ công dễ bị ảnh hưởng bởi yếu tố chủ quan</i>	Data-driven có thể tự động hóa, phân tích dữ liệu và tối ưu hóa quy trình phân bổ, mang lại hiệu quả vượt trội. <i>Hệ thống tự động để xuất chuyên hàng tối ưu</i>

Hình 4: Tóm tắt và ví dụ ba vấn đề và một hướng giải quyết cho bài toán tối ưu.

Mỗi khía cạnh này đều ẩn chứa những thách thức và cơ hội riêng:

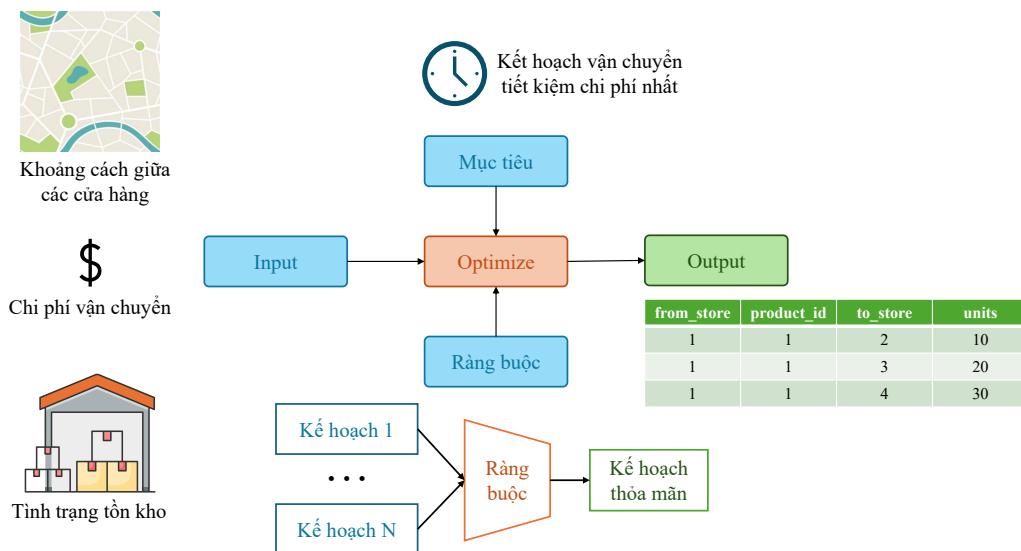
- Về mặt thực tế, sự mất cân bằng tồn kho là một lãng phí tài nguyên hữu hình. Ví dụ về “Cửa hàng A tồn kho 100 sản phẩm, cửa hàng B hết sạch” không chỉ đơn thuần là vấn đề lưu trữ, mà nó trực tiếp dẫn đến việc bỏ lỡ doanh thu tại cửa hàng B và làm suy giảm trải nghiệm của khách hàng.
- Về chi phí, bài toán không chỉ dừng lại ở chi phí vận chuyển. Tồn kho dư thừa gây lãng phí vốn lưu động, phát sinh chi phí bảo quản, bảo

hiểm, và đối mặt với rủi ro hàng hóa lỗi thời hoặc hư hỏng. Một hệ thống không tối ưu sẽ khuếch đại tất cả các loại chi phí này.

- Về quyết định, sự phụ thuộc vào kinh nghiệm cá nhân khiến quy trình trở nên chậm chạp và thiếu nhất quán. Các quyết định thủ công khó có thể phản ứng kịp thời trước các biến động đột ngột của thị trường (ví dụ: một xu hướng mới, một chương trình khuyến mãi của đối thủ) và dễ bị sai lệch bởi yếu tố chủ quan.
- Về cơ hội, đây chính là khía cạnh then chốt. Dữ liệu bán hàng, tồn kho, và lưu lượng khách hàng không còn là những con số. Khi được khai thác đúng cách, chúng trở thành nền tảng cho một hệ thống **data-driven** có khả năng tự động hóa toàn bộ quy trình: từ dự báo nhu cầu, phân tích các kịch bản điều chuyển, cho đến việc đề xuất lộ trình tối ưu nhất. Đây chính là giải pháp mà chúng ta sẽ xây dựng.

25.2.2 Định nghĩa bài toán

Một bài toán tối ưu về cơ bản là việc tìm ra phương án tốt nhất (ví dụ: kế hoạch luân chuyển hàng) dựa trên một bộ tiêu chí (mục tiêu) và tuân thủ các điều kiện cho trước (ràng buộc).



Hình 5: Các thành phần cốt lõi của một bài toán tối ưu:
Mục tiêu, Dữ liệu đầu vào, Ràng buộc và Kết quả đầu ra.

Để máy tính có thể hiểu và giải quyết, chúng ta cần diễn đạt bài toán này bằng ngôn ngữ toán học. Mô hình toán học giúp chúng ta lượng hóa mục tiêu và các ràng buộc một cách chính xác.

<p>1. Mục tiêu: Giảm thiểu chi phí</p> <p>Tìm cách chuyển hàng giữa các cửa hàng để tổng chi phí vận chuyển là thấp nhất.</p> <p>Minimize:</p> $\sum_i \sum_j \sum_p cost[i, j] \times x[i, j, p]$ <ul style="list-style-type: none"> • $cost[i, j]$: Chi phí chuyển 1 đơn vị hàng từ cửa hàng i đến cửa hàng j. • $x[i, j, p]$: Số lượng sản phẩm p được chuyển từ cửa hàng i đến cửa hàng j. 	<p>2. Ràng buộc cung cấp (Supply)</p> <p>Lượng hàng chuyển đi từ một cửa hàng không thể vượt quá số lượng hàng hóa dư thừa tại đó.</p> $\sum_j x[i, j, p] \leq excess[i, p]$ <ul style="list-style-type: none"> • $excess[i, p]$: Lượng hàng hóa p đang dư thừa tại cửa hàng i. 	<p>3. Ràng buộc nhu cầu (Demand)</p> <p>Lượng hàng nhận về của một cửa hàng không thể vượt quá số lượng hàng hóa đang thiếu tại đó.</p> $\sum_i x[i, j, p] \leq needed[j, p]$ <ul style="list-style-type: none"> • $needed[j, p]$: Lượng hàng hóa p đang cần tại cửa hàng j. • Ví dụ: Nếu cửa hàng A có dư 10 áo sơ mi, nó chỉ có thể chuyển tối đa 10 áo sơ mi. • Ví dụ: Nếu cửa hàng B thiếu 5 đôi giày, nó chỉ có thể nhận tối đa 5 đôi giày.
<p>4. Ràng buộc về số lượng chuyển</p> <p>Số lượng hàng hóa được chuyển phải là số nguyên không âm.</p> $x[i, j, p] \geq 0, \text{integer } \forall i, j, p$	<p>5. Ràng buộc không tự chuyển</p> <p>Một cửa hàng không thể chuyển hàng cho chính nó.</p> $x[i, j, p] = 0, \forall i, p$	

Hình 6: Mô hình toán học cho bài toán phân bổ hàng hóa nhằm tối thiểu hóa chi phí vận chuyển.

Thay vì diễn giải lại từng công thức, chúng ta sẽ tập trung vào bước quan trọng tiếp theo: làm thế nào để lập trình dự án này để có thể tìm ra lời giải. Quá trình này bao gồm ba bước chính:

- Cấu trúc dữ liệu đầu vào:** Đầu tiên, chúng ta cần biểu diễn các dữ liệu đầu vào dưới dạng các cấu trúc mà chương trình có thể xử lý. Trong các dự án phân tích dữ liệu với Python, cách tiếp cận hiệu quả nhất là sử dụng thư viện **Pandas**. Dữ liệu về chi phí, hàng thừa và hàng thiếu sẽ được chứa trong các đối tượng **DataFrame**.

Ví dụ cấu trúc dữ liệu trong Python với Pandas

```

1 # Dữ liệu về chi phí vận chuyển được tải từ file CSV
2 # transport_costs_df có các cột: from_store_id, to_store_id,
   cost
3 transport_costs_df.head()
4 #      from_store_id to_store_id    cost
5 # 0      CH01          CH02      15000
6 # 1      CH01          CH03      25000
7 # ...
8
9 # Dữ liệu hàng thừa và hàng thiếu được tính toán bởi

```

```

    InventoryAnalyzer
10 # và trả về dưới dạng DataFrame.
11 excess_df, needed_df = analyzer.identify_inventory_imbalances()
12
13 # excess_df chứa thông tin các sản phẩm đang thừa ở cửa hàng
14 excess_df[['store_id', 'product_id', 'excess_units']].head()
15 #   store_id  product_id  excess_units
16 # 5   CH01      AO_SO_MI    10
17 # 12  CH01      QUAN_JEAN     5
18 # ...
19
20 # needed_df chứa thông tin các sản phẩm đang thiếu ở cửa hàng
21 needed_df[['store_id', 'product_id', 'needed_units']].head()
22 #   store_id  product_id  needed_units
23 # 3   CH02      AO_SO_MI     8
24 # 9   CH03      AO_SO_MI    12
25 # ...

```

2. **Biểu diễn lời giải:** Thành phần chính của dự án chúng ta là cấu trúc của một “kế hoạch luân chuyển”. Trong dự án này, một kế hoạch hoàn chỉnh là một danh sách các giao dịch vận chuyển. Mỗi giao dịch là một bộ thông tin xác định:

- **Từ đâu:** Cửa hàng gửi (`from_store_id`).
- **Đến đâu:** Cửa hàng nhận (`to_store_id`).
- **Mặt hàng gì:** Mã sản phẩm (`product_id`).
- **Số lượng:** Bao nhiêu đơn vị sản phẩm sẽ được chuyển (`units`).

Đây chính là cấu trúc dữ liệu đầu ra mà các thuật toán của chúng ta phải tạo ra.

3. **Xây dựng thuật toán:** Với dữ liệu đầu vào và cấu trúc lời giải đã định nghĩa, bước cuối cùng là hiện thực hóa các thuật toán để tìm ra kế hoạch tốt nhất. Dự án này tiếp cận bài toán bằng hai phương pháp:

- **Phương pháp dựa trên luật (Rule-Based):** Đây là một chiến lược đơn giản và trực quan. Thuật toán sẽ duyệt qua từng món hàng đang thiếu, tìm kiếm trong danh sách hàng thừa và chọn ra

phương án vận chuyển có chi phí thấp nhất để đáp ứng nhu cầu đó.

- **Phương pháp giải thuật di truyền:** Đây là một hướng tiếp cận phức tạp hơn, lấy cảm hứng từ quá trình tiến hóa tự nhiên. Thuật toán sẽ khởi tạo một “quần thể” gồm nhiều kế hoạch vận chuyển ngẫu nhiên, sau đó đánh giá, lai tạo và đột biến chúng qua nhiều thế hệ để “tiến hóa” ra lời giải tối ưu nhất dựa trên tổng chi phí.

Bằng cách chuyển đổi mô hình toán học trừu tượng thành các cấu trúc dữ liệu và thuật toán cụ thể, chúng ta đã sẵn sàng để yêu cầu máy tính tìm ra kế hoạch vận chuyển tối ưu nhất.

25.3 Xây dựng hệ thống

Sau khi đã có cái nhìn tổng quan về mô hình toán học và các hướng tiếp cận, chúng ta sẽ đi sâu vào chi tiết kỹ thuật của dự án. Phần này sẽ giải thích cách hệ thống được xây dựng, từ việc đọc dữ liệu thô cho đến khi tạo ra một kế hoạch luân chuyển hàng hóa hoàn chỉnh.

25.3.1 Đọc dữ liệu

Dự án sử dụng 6 file CSV trong thư mục ‘data/’, nhưng trong luồng phân tích chính, chúng ta chỉ làm việc trực tiếp với 5 file.

- **Các file đầu vào chính:** ‘stores.csv’, ‘products.csv’, ‘sales_data.csv’, ‘inventory_data.csv’
- **File chi phí:** ‘transport_cost_matrix.csv’ chứa chi phí vận chuyển (VND) giữa các cửa hàng.

Lớp `InventoryAnalyzer` sẽ đọc 4 file dữ liệu chính, trong khi file chi phí được đọc riêng để truyền vào các thuật toán. Chúng ta có cách chạy lớp `InventoryAnalyzer` như sau.

Đọc dữ liệu đầu vào

```

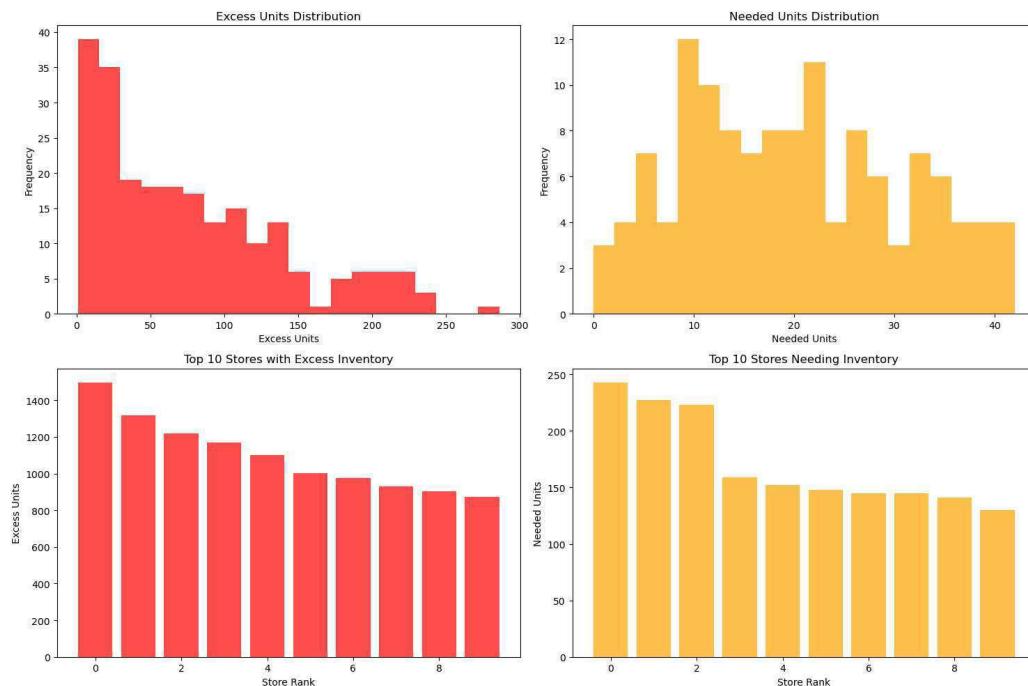
1 import pandas as pd
2 from src.engine.analyzer import InventoryAnalyzer
3
4 # Khởi tạo analyzer
5 analyzer = InventoryAnalyzer()
6
7 # Tải 4 file dữ liệu kinh doanh cốt lõi
8 analyzer.load_data(
9     sales_path="data/sales_data.csv",
10    inventory_path="data/inventory_data.csv",
11    stores_path="data/stores.csv",
12    products_path="data/products.csv"
13)
14
15 # Dữ liệu chi phí vận chuyển cũng được đọc riêng
16 transport_costs_df = pd.read_csv("data/transport_cost_matrix.csv")

```

Sau khi tải, phương thức `identify_inventory_imbalances` sẽ tạo ra hai “bảng” dữ liệu, là đầu vào cốt lõi cho cả hai thuật toán tối ưu.

- **Bảng hàng thừa** (`‘excess_df’`): Liệt kê những sản phẩm tại các cửa hàng có lượng tồn kho vượt quá `MAX_INVENTORY_DAYS`. Bảng này sẽ có thêm cột `excess_units` cho biết số lượng đơn vị đang thừa.
- **Bảng hàng thiếu** (`‘needed_df’`): Liệt kê những sản phẩm tại các cửa hàng có lượng tồn kho thấp hơn `MIN_INVENTORY_DAYS`. Bảng này có cột `needed_units` cho biết số lượng cần bổ sung.

Với hai bảng dữ liệu này cùng với ma trận chi phí vận chuyển, chúng ta đã có đầy đủ thông tin cần thiết để bắt đầu tìm kiếm một kế hoạch luân chuyển hàng hóa hiệu quả.



Hình 7: Phân phối của dữ liệu. Ta thấy rằng số lượng hàng dư ít hơn số lượng hàng cần rất nhiều. Đây là một bộ dữ liệu bị mất cân bằng.

25.3.2 Phương pháp dựa trên các luật

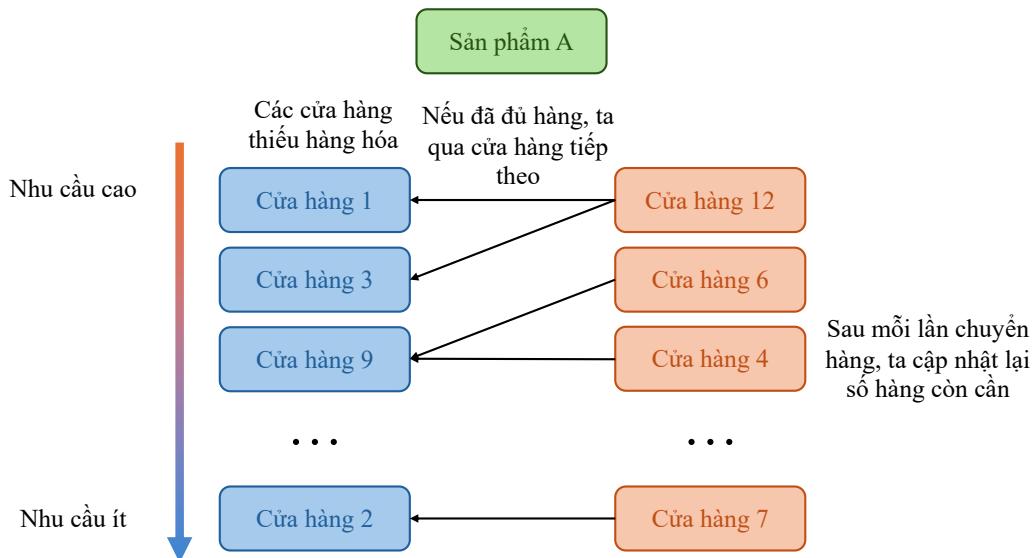
Đây là phương pháp tiếp cận đầu tiên và đơn giản nhất để giải quyết bài toán. Ý tưởng cốt lõi của nó là một chiến lược “tham lam” (greedy): với mỗi món hàng đang thiếu, chúng ta sẽ ngay lập tức tìm cách đáp ứng nó bằng nguồn cung có chi phí vận chuyển rẻ nhất. Phương pháp này không đảm bảo sẽ tìm ra lời giải tối ưu nhất trên toàn cục, nhưng nó rất nhanh, dễ hiểu và thường cho ra một kết quả “đủ tốt”.

Logic hoạt động Thuật toán được thực hiện trong lớp `RuleBasedOptimizer`. Luồng của thuật toán có thể được mô tả như sau:

1. **Sắp xếp nhu cầu:** Đầu tiên, thuật toán sắp xếp bảng `needed_df` để ưu tiên những nhu cầu lớn nhất. Điều này giúp giải quyết các vấn đề thiếu hụt nghiêm trọng trước.

2. **Duyệt qua từng nhu cầu:** Với mỗi dòng trong bảng `needed_df` (tức là mỗi sản phẩm p đang thiếu tại cửa hàng j), thuật toán sẽ:
 - (a) Tìm tất cả các cửa hàng i có thừa sản phẩm p từ bảng `excess_df`.
 - (b) Với mỗi cửa hàng i tìm được, lấy ra chi phí vận chuyển từ i đến j trong ma trận chi phí.
 - (c) Sắp xếp các cửa hàng cung cấp tiềm năng này theo chi phí vận chuyển tăng dần.
3. **Thực hiện vận chuyển:** Thuật toán bắt đầu duyệt qua danh sách các nguồn cung đã sắp xếp (từ rẻ nhất đến đắt nhất). Nó sẽ lấy hàng từ nguồn rẻ nhất cho đến khi nhu cầu được đáp ứng, hoặc nguồn cung đó hết hàng. Nếu nhu cầu vẫn chưa đủ, nó sẽ tiếp tục lấy từ nguồn rẻ thứ hai, và cứ thế tiếp tục.
4. **Cập nhật và lặp lại:** Sau mỗi lần vận chuyển, số lượng hàng thừa và thiếu ở các kho liên quan sẽ được cập nhật lại. Thuật toán tiếp tục cho đến khi tất cả các nhu cầu đã được xem xét.

Logic trên có thể được biểu diễn bằng đoạn mã giả sau, giúp làm rõ cấu trúc của vòng lặp chính trong phương thức `optimize`. Ở đây, ta có thể thấy thuật toán có độ phức tạp là $O(N \times M \times \log(M))$ với N là số lượng nhu cầu, và M là số lượng hàng thừa.



Hình 8: Hình minh họa thuật toán rule-based dựa trên một sản phẩm. Tiêu chí chúng ta chọn cửa hàng cung cấp hàng hóa cần là chi phí vận chuyển thấp nhất.

Sau khi đã định nghĩa thuật toán xong, việc sử dụng thuật toán này rất đơn giản. Sau khi đã có `excess_df` và `needed_df` từ `InventoryAnalyzer`, chúng ta chỉ cần khởi tạo `RuleBasedOptimizer` và gọi phương thức `optimize`.

Thực thi thuật toán Rule-Based

```

1 # Khởi tạo Rule-Based Optimizer với ma trận chi phí
2 rule_optimizer = RuleBasedOptimizer(
3     transport_cost_matrix=transport_cost_matrix
4 )
5
6 # Chạy thuật toán
7 rule_transfer_plan = rule_optimizer.optimize(excess_df, needed_df)
8
9 # In ra kết quả
10 print(f"Tổng chi phí vận chuyển: {rule_transfer_plan['transport_cost'
11     ',].sum():,.0f} VND")
    display(rule_transfer_plan.head())

```

Kết quả trả về là một DataFrame chứa kế hoạch chi tiết: sản phẩm nào cần được chuyển, từ đâu, đến đâu, với số lượng và chi phí là bao nhiêu. Đây là một giải pháp nhanh chóng và dễ dàng để cải thiện tình hình tồn kho.

idx	from_-store_id	to_-store_id	product_-id	units	distance (km)	transport_-cost (VND)
0	5	7	3	42	20.53	2.07×10^6
1	19	14	23	42	6.79	6.84×10^5
2	5	2	4	42	7.41	7.47×10^5
3	18	15	24	40	6.20	5.95×10^5
4	19	17	19	39	2.69	2.52×10^5
5	19	20	19	19	12.91	5.89×10^5
6	14	20	19	20	14.61	7.01×10^5
7	8	10	25	22	3.34	1.76×10^5
8	20	10	25	16	596.07	1.14×10^7
9	8	15	28	38	611.75	2.79×10^7

Bảng 1: Ví dụ về kết quả đầu ra của kế hoạch vận chuyển.

Với mỗi hàng là kế hoạch vận chuyển của một sản phẩm,
và một sản phẩm có thể được vận chuyển nhiều lần.

25.3.3 Phương pháp thuật toán di truyền

Nếu phương pháp dựa trên luật là một chiến lược đơn giản và nhanh chóng, thì thuật toán di truyền (GA) là một cách tiếp cận phức tạp hơn, được thiết kế để khám phá không gian lời giải một cách rộng hơn nhằm tìm ra kết quả tiềm cận tối ưu toàn cục. Thuật toán này mô phỏng lại quá trình tiến hóa tự nhiên: tạo ra một “quần thể” các lời giải, cho chúng “giao phối” và “đột biến” qua nhiều thế hệ để tìm ra lời giải tốt nhất (có chi phí thấp nhất).

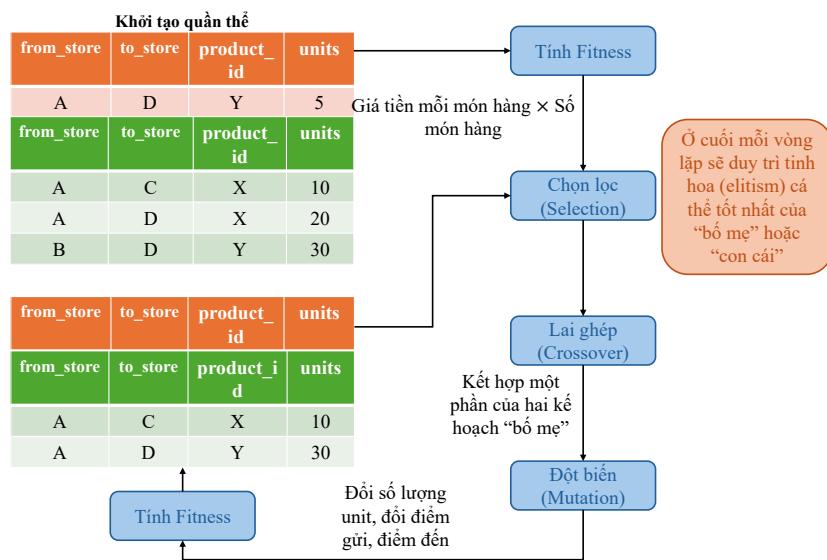
Thuật toán được xây dựng trong lớp `GeneticAlgorithmOptimizer`. Để hiểu nó, chúng ta cần nắm các khái niệm sau:

- **Cá thể (Individual):** Mỗi cá thể là một lời giải hoàn chỉnh, tức là một kế hoạch luân chuyển hàng hóa.

- **Quần thể (Population):** Là một tập hợp gồm nhiều cá thể. Ví dụ, một quần thể có 50 cá thể tức là 50 kế hoạch luân chuyển khác nhau.
- **Hàm thích nghi (Fitness Function):** Đây là hàm đánh giá chất lượng của một cá thể. Trong bài toán này, “thích nghi” đồng nghĩa với “chi phí thấp”. Hàm này tính tổng chi phí vận chuyển của một kế hoạch. Cá thể nào có fitness thấp hơn thì được coi là tốt hơn.
- **Chọn lọc (Selection):** Quá trình chọn ra các cá thể “tốt” (chi phí thấp) từ quần thể để làm “cha mẹ” cho thế hệ tiếp theo. Dự án sử dụng phương pháp *Tournament Selection*.
- **Lai ghép (Crossover):** Quá trình hai cha mẹ “giao phối” để tạo ra con. Về bản chất, nó trộn lẫn các phần của hai kế hoạch vận chuyển để tạo ra một kế hoạch mới, hy vọng sẽ tốt hơn.
- **Đột biến (Mutation):** Thay đổi nhỏ, ngẫu nhiên trên một cá thể con để tạo ra sự đa dạng trong quần thể, tránh việc bị “mắc kẹt” ở một lời giải cục bộ.

Để vận hành thuật toán, chúng ta cần tinh chỉnh các tham số cốt lõi như sau. *Kích thước quần thể (population size)* quyết định số lượng lời giải được duy trì trong mỗi vòng lặp, trong khi *số thế hệ (generations)* xác định độ dài của quá trình tiến hóa. Bên cạnh đó, *tỷ lệ lai ghép (crossover rate)* và *tỷ lệ đột biến (mutation rate)* là hai xác suất quan trọng, điều khiển tần suất các cá thể mới được tạo ra thông qua việc kết hợp từ cha mẹ hoặc thay đổi ngẫu nhiên, từ đó cân bằng giữa việc khai thác các lời giải tốt hiện có và khám phá những vùng lời giải tiềm năng mới. Các tham số được nói trên có thể được tin chỉnh ở file ‘src/config.py’.

Hàm thích nghi (_calculate_fitness) Đây là hàm lõi để đánh giá mỗi kế hoạch vận chuyển. Logic của nó dựa trên hàm mục tiêu mà ta đã định nghĩa ở trên: duyệt qua từng hàng trong kế hoạch, tra cứu chi phí trên mỗi đơn vị từ ma trận chi phí, nhân với số lượng, rồi cộng dồn lại.



Hình 9: Quá trình thuật toán Genetic sẽ thực hiện trong hệ thống.

Giả mã hàm _calculate_fitness

```

1 def _calculate_fitness(transfer_plan, transport_cost_matrix):
2     total_cost = 0
3
4     for transfer in transfer_plan:
5         from_store = transfer['from_store']
6         to_store = transfer['to_store']
7         units = transfer['units']
8
9         # Tra cứu chi phí trên mỗi đơn vị
10        cost_per_unit = transport_cost_matrix.loc[from_store,
11                                                 to_store]
12
13        # Cộng dồn vào tổng chi phí
14        total_cost += cost_per_unit * units
15
16    return total_cost

```

Lai ghép (_crossover) Hàm này nhận hai kế hoạch (cha và mẹ) và tạo ra hai kế hoạch mới (con). Dự án sử dụng phương pháp lai ghép một điểm (single-point crossover) đơn giản.

Giả mã hàm _crossover

```

1 def _crossover(parent1_plan, parent2_plan):
2     # Chọn một điểm ngẫu nhiên để cắt
3     crossover_point = random.randint(1, min(len(parent1_plan), len(
4                                     parent2_plan)) - 1)
5
6     # Tạo ra hai cá thể con
7     # Con 1 = đầu của cha 1 + đuôi của cha 2
8     child1_plan = parent1_plan[:crossover_point] + parent2_plan[
9                                     crossover_point:]
10
11    # Con 2 = đầu của cha 2 + đuôi của cha 1
12    child2_plan = parent2_plan[:crossover_point] + parent1_plan[
13                                     crossover_point:]
14
15    # "Sửa chữa" các kế hoạch con để đảm bảo chúng hợp lệ
16    child1_plan = _repair_solution(child1_plan)
17    child2_plan = _repair_solution(child2_plan)
18
19    return child1_plan, child2_plan

```

Đột biến (_mutate) Để duy trì sự đa dạng, một cá thể con có thể bị đột biến ngẫu nhiên. Hàm _mutate sẽ chọn một trong bốn loại đột biến sau:

Giả mã các loại đột biến trong hàm _mutate

```

1 def _mutate(individual_plan):
2     mutation_type = random.choice(['change_quantity', '
3                                         'remove_transfer', 'add_transfer',
4                                         'change_route'])
5
6     if mutation_type == 'change_quantity':
7         # 1. Thay đổi số lượng của một giao dịch ngẫu nhiên
8         random_transfer = random.choice(individual_plan)
9         max_possible_units = get_max_transfer(random_transfer['
10                                         from_store'], ...)

```

```

8     random_transfer['units'] = random.randint(1,
9                               max_possible_units)
10
11    elif mutation_type == 'remove_transfer':
12        # 2. Xóa một giao dịch ngẫu nhiên
13        if len(individual_plan) > 1:
14            individual_plan.pop(random.randint(0, len(
15                                individual_plan) - 1))
16
17    elif mutation_type == 'add_transfer':
18        # 3. Thêm một giao dịch hoàn toàn mới và ngẫu nhiên
19        new_transfer = _create_random_transfer()
20        if new_transfer:
21            individual_plan.append(new_transfer)
22
23    elif mutation_type == 'change_route':
24        # 4. Thay đổi nguồn hoặc đích của một giao dịch
25        random_transfer = random.choice(individual_plan)
26        if random.random() < 0.5:
27            # Thay đổi cửa hàng gửi (from_store)
28            new_from_store = find_another_store_with_excess(
29                            random_transfer['product_id'])
30            random_transfer['from_store'] = new_from_store
31        else:
32            # Thay đổi cửa hàng nhận (to_store)
33            new_to_store = find_another_store_with_need(
34                            random_transfer['product_id'])
35            random_transfer['to_store'] = new_to_store
36
37    # Cuối cùng, sửa chữa lại kế hoạch để đảm bảo tính hợp lệ
38    individual_plan = _repair_solution(individual_plan)
39    return individual_plan

```

Tương tự như thuật toán trước, việc gọi GA cũng rất đơn giản. Chúng ta khởi tạo lớp GAOptimizer và gọi phương thức optimize, truyền vào các tham số cấu hình như kích thước quần thể, số thế hệ, v.v. Ta có thể thấy rằng với thiết lập của thuật toán này, thời gian cần để đưa ra đáp án sẽ nhiều hơn đáng kể so với thuật toán Rule-based được nói ở trên.

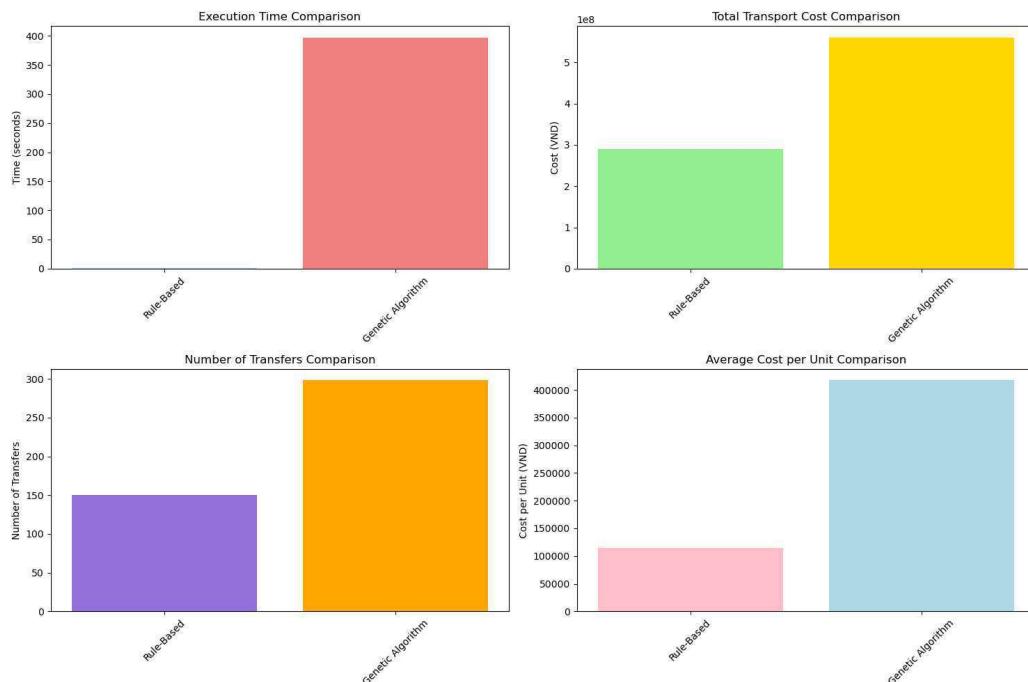
Thực thi thuật toán Genetic Algorithm

```
1 # Khởi tạo GA Optimizer
2 ga_optimizer = GeneticAlgorithmOptimizer(
3     transport_cost_matrix=transport_cost_matrix
4 )
5
6 # Chạy thuật toán với các tham số từ file config
7 ga_transfer_plan = ga_optimizer.optimize(
8     excess_inventory=excess_df,
9     needed_inventory=needed_df,
10    population_size=GA_POPULATION_SIZE,
11    num_generations=GA_GENERATIONS,
12    verbose=True # Hiển thị tiến trình qua các thế hệ
13 )
14
15 # In ra kết quả
16 print(f"Tổng chi phí vận chuyển (GA): {ga_transfer_plan['"
17             "transport_cost'].sum():,.0f} VND")
18 display(ga_transfer_plan.head())
```

Quá trình này sẽ mất nhiều thời gian hơn so với phương pháp dựa trên luật, nhưng nó có khả năng tìm ra một kế hoạch luân chuyển với tổng chi phí thấp hơn đáng kể.

25.3.4 So sánh hiệu năng và lựa chọn thuật toán

Sau khi đã hiện thực hóa cả hai phương pháp, chúng ta cần so sánh và lựa chọn ra chiến lược tối ưu nhất cho bài toán hiện tại. Dựa trên kết quả thực thi được ghi nhận trong notebook (xem Hình 10), chúng ta có những kết luận như sau.



Hình 10: Biểu đồ so sánh hiệu năng giữa thuật toán Rule-Based và thuật toán di truyền trên bốn tiêu chí: thời gian thực thi, tổng chi phí, số lượng chuyển hàng, và chi phí trung bình trên mỗi đơn vị.

Về **thời gian thực thi**, sự khác biệt của hai thuật toán là rất lớn: thuật toán Rule-Based hoàn thành gần như tức thì chỉ trong vài giây, trong khi thuật toán di truyền đòi hỏi năng lực tính toán lớn và mất vài trăm giây để chạy qua tất cả các thế hệ (thay đổi mỗi lần chạy). Tuy nhiên, kết quả bất ngờ và quan trọng nhất nằm ở **tổng chi phí vận chuyển**. Thuật toán Rule-Based, với logic “tham lam” đơn giản, lại tìm ra một kế hoạch có tổng chi phí thấp hơn đáng kể so với thuật toán di truyền. Điều này cho thấy chiến lược “ưu tiên tuyến đường rẻ nhất” tỏ ra rất hiệu quả với bộ dữ liệu hiện tại. Phân tích sâu hơn về **số lượng chuyển hàng**, thuật toán di truyền có xu hướng tạo ra một kế hoạch “phân mảnh” hơn với số lượng giao dịch gần như gấp đôi, trong khi Rule-Based lại gộp các giao dịch hiệu quả hơn. Cuối cùng, **chi phí trung bình trên mỗi đơn vị** một lần nữa khẳng định điều này, khi chi phí của GA cao hơn nhiều so với Rule-Based, chứng tỏ các quyết định của thuật toán đơn giản lại hiệu quả hơn về mặt chi phí.

Vì sao Rule-Based lại hiệu quả hơn?

Lý giải cho kết quả nói trên nằm ở cấu trúc rất rõ ràng của bài toán. Luồng xử lý được định nghĩa rất cụ thể: mỗi sản phẩm chỉ được phép di chuyển từ một cửa hàng đang thừa sang một cửa hàng đang thiếu, và phải tuân thủ ràng buộc chặt chẽ là không được vận chuyển vượt quá số lượng tồn kho. Mục tiêu duy nhất là tối thiểu hóa chi phí. Nhờ cấu trúc chặt chẽ được nói trên, phương pháp Rule-Based có thể tận dụng logic đặc thù của bài toán một cách triệt để, giúp nó nhanh chóng tìm ra lời giải tốt, dễ kiểm soát và mang lại hiệu quả cao.

25.4 Hướng cải tiến

Hệ thống hiện tại đã cung cấp một giải pháp hoàn chỉnh cho bài toán luân chuyển tồn kho. Tuy nhiên, trong một kịch bản vận hành thực tế, luôn có những cơ hội để cải tiến và tối ưu hơn nữa. Dưới đây là hai hướng đi tiềm năng có thể giúp nâng cao đáng kể hiệu quả của hệ thống.

25.4.1 Tích hợp dữ liệu thực và mô hình dự báo

Một trong những hạn chế lớn nhất của hệ thống hiện tại là nó mang tính “phản ứng” (reactive). Nó xác định hàng thừa và thiếu dựa trên tốc độ bán hàng *trong quá khứ* được ghi ở trong file “sales_data.csv”. Một cải tiến mà chúng ta có thể thực hiện ở đây chuyển hệ thống sang phương pháp “chủ động” (proactive) bằng cách tích hợp các mô hình dự báo (forecasting models) để dự đoán nhu cầu tương lai.

Ví dụ với dữ liệu thực tế Để đưa hệ thống đến gần hơn với thực tế, chúng ta có thể sử dụng các bộ dữ liệu bán lẻ công khai. Mặc dù dữ liệu nội bộ của doanh nghiệp là không có sẵn, chúng ta có thể dùng một bộ dữ liệu tương đương là **Brazilian E-Commerce Public Dataset by Olist**.

Áp dụng bộ dữ liệu Olist vào dự án

Nguồn dữ liệu: [Brazilian E-Commerce Public Dataset by Olist](#) trên Kaggle.

Bộ dữ liệu này chứa thông tin của gần 100,000 đơn hàng và có thể được áp dụng vào dự án của chúng ta như sau:

- **Dữ liệu bán hàng ('sales_data.csv')**: Có thể được tổng hợp từ file `olist_order_items_dataset.csv`. Mỗi `seller_id` sẽ được coi là một "cửa hàng".
- **Dữ liệu sản phẩm ('products.csv')**: Lấy trực tiếp từ file `olist_products_dataset.csv`.
- **Dữ liệu cửa hàng ('stores.csv')**: Lấy từ `olist_sellers_dataset.csv`, trong đó mỗi người bán (`seller_id`) là một cửa hàng.
- **Ma trận khoảng cách và chi phí**: File `olist_geolocation_dataset.csv` cung cấp tọa độ địa lý thực của các mã zip cho mỗi người bán ở trên, cho phép chúng ta tính toán ma trận khoảng cách và chi phí vận chuyển cho bài toán hiện tại.
- **Dữ liệu tồn kho ('inventory_data.csv')**: Đây là phần duy nhất bị thiếu của bộ dữ liệu E-Commerce nói trên. Chúng ta cần tự tạo ra nó (`synthesize`) bằng cách tính tốc độ bán hàng trung bình của mỗi sản phẩm tại mỗi cửa hàng, sau đó gán một mức tồn kho ngẫu nhiên nhưng hợp lý (ví dụ: tồn kho bằng 20-100 ngày bán hàng). Các bạn có thể tham khảo folder '`src/data-generator`' trong bài toán để có thể biết cách tạo dữ liệu ở đây.

Việc sử dụng bộ dữ liệu này giúp kiểm thử các thuật toán trên một quy mô lớn hơn và với các đặc tính phân phối địa lý thực tế.

Sau khi có dữ liệu bán hàng theo chuỗi thời gian từ Olist, chúng ta có thể huấn luyện các mô hình dự báo như **ARIMA**, **Prophet**, hoặc **LSTM**. Thay vì tính `avg_daily_sales` dựa trên quá khứ, chúng ta sẽ dùng `forecasted_daily_sales` cho 30 ngày tới. Từ đó, hệ thống có thể xác định các sản phẩm sẽ sớm bị thiếu hoặc thừa, phù hợp hơn trong các bài toán thực tế của chúng

ta.

25.4.2 Cải tiến và tối ưu hóa giải thuật di truyền

Bên cạnh việc tích hợp dữ liệu dự báo, bản thân thuật toán di truyền cũng có rất nhiều vị trí cần để cải thiện, giúp nó tìm ra lời giải tốt hơn cho bài toán đơn mục tiêu hiện tại. Hướng cải tiến đầu tiên và cơ bản nhất chính là **tối ưu hóa tham số**. Các tham số như kích thước quần thể, tỷ lệ lai ghép, và tỷ lệ đột biến có ảnh hưởng trực tiếp đến sự cân bằng giữa tốc độ hội tụ và khả năng khám phá không gian lời giải. Việc đánh giá và so sánh kết quả khi thay đổi từng tham số sẽ giúp chúng ta hiểu rõ mức độ ảnh hưởng của chúng đến hiệu quả thuật toán.

Các hướng nâng cấp cấu trúc cho thuật toán i truyền

Khi bài toán trở nên phức tạp hơn, chúng ta có thể xem xét các nâng cấp sâu hơn về mặt cấu trúc như sau:

- **Cải tiến khởi tạo quần thể:** Thay vì khởi tạo hoàn toàn ngẫu nhiên, ta có thể tạo ra một quần thể ban đầu “thông minh” hơn bằng cách kết hợp một tỷ lệ nhỏ (ví dụ: 20%) các cá thể tốt được tạo ra từ phương pháp Rule-Based với phần còn lại (80%) được tạo ngẫu nhiên.
- **Thiết kế toán tử lai ghép đặt biệt cho bài toán này (Problem-aware Crossover):** Phép lai ghép mặc định có thể phá vỡ các tuyến vận chuyển tốt đang có. Một phép lai ghép được thiết kế riêng cho bài toán (ví dụ: route-preserving crossover) sẽ giữ lại các chuỗi quyết định hợp lý từ cặp “cha mẹ”.
- **Sử dụng toán tử thích ứng (Adaptive Operators):** Thay vì giữ cố định các tỷ lệ như tỉ lệ đột biến, tỉ lệ lai, ..., thuật toán có thể tự động điều chỉnh chúng trong quá trình chạy. Ví dụ, giảm tỷ lệ đột biến khi thuật toán gần hội tụ để tinh chỉnh lời giải một cách chính xác hơn.
- **Sử dụng phương pháp lai ghép (Hybrid Approach):** Đây là một kỹ thuật kết hợp GA với một thuật toán tìm kiếm cục bộ (local search). Sau mỗi thế hệ, những cá thể tốt nhất sẽ được một thuật toán cục bộ tối ưu hóa thêm. Cách tiếp cận này còn được gọi là **Memetic Algorithm** và đã được áp dụng hiệu quả trong dự án thực tế.

Những cải tiến này đặc biệt phù hợp khi bài toán có **quy mô lớn** (ví dụ: trên 500 cửa hàng, 1,000 sản phẩm) hoặc có các **ràng buộc động** thay đổi theo thời gian (chi phí, giới hạn kho, điều kiện vận hành), khiến cho các lời giải đơn giản không còn hiệu quả.

25.4.3 Áp dụng thuật toán đa mục tiêu (NSGA-II)

Thuật toán di truyền hiện tại được thiết kế để tối ưu một mục tiêu duy nhất: **tổng chi phí vận chuyển**. Tuy nhiên, trên thực tế, một doanh nghiệp có

thể muốn cân bằng nhiều mục tiêu khác nhau và chúng thường mâu thuẫn với nhau. Ví dụ:

- **Giảm chi phí vận chuyển** (mục tiêu hiện tại).
- **Giảm số lượng chuyến hàng** (giảm chi phí nhân công, vận hành).
- **Tối đa hóa mức độ đáp ứng nhu cầu** (fill rate).
- **Cân bằng tồn kho sau luân chuyển** (giảm rủi ro trên toàn hệ thống).

Để giải quyết bài toán này, chúng ta có thể sử dụng một biến thể của thuật toán di truyền có tên là **NSGA-II (Non-dominated Sorting Genetic Algorithm II)**. Thay vì tìm ra một lời giải “tốt nhất”, NSGA-II sẽ tìm ra một tập hợp các lời giải tối ưu được gọi là **mặt trận Pareto (Pareto Front)**. Mỗi lời giải trên mặt trận này đại diện cho một sự đánh đổi khác nhau giữa các mục tiêu (ví dụ: một lời giải chi phí thấp nhưng đáp ứng nhu cầu thấp, một lời giải khác chi phí cao nhưng đáp ứng nhu cầu cao). Điều này cho phép người quản lý lựa chọn kịch bản phù hợp nhất với chiến lược kinh doanh tại từng thời điểm.

25.5 Câu hỏi trắc nghiệm

1. Trong file `src/config.py`, biến `MIN_INVENTORY_DAYS` được sử dụng để làm gì?
2. Xác định tuổi tối thiểu của một sản phẩm để được coi là hàng tồn kho.
3. Tính toán mức tồn kho an toàn tối thiểu mà một cửa hàng nên duy trì cho mỗi sản phẩm.
4. Quy định thời gian vận chuyển tối thiểu giữa hai cửa hàng bất kỳ.
5. Đặt ra số ngày tối thiểu mà một sản phẩm phải có trong kho trước khi được bán.
6. Lớp `InventoryAnalyzer` trong file `src/engine/analyzer.py` xác định một sản phẩm tại một cửa hàng là “hàng thừa” (excess) dựa trên tiêu chí chính nào?
7. Khi số lượng tồn kho của nó lớn hơn tổng nhu cầu của toàn bộ hệ thống.
8. Khi chỉ số `days_of_supply` của nó cao hơn giá trị `MAX_INVENTORY_DAYS`.
9. Khi chỉ số `days_of_supply` của nó thấp hơn giá trị `MIN_INVENTORY_DAYS`.
10. Khi sản phẩm đó không bán được trong suốt `MAX_INVENTORY_DAYS` ngày.
11. Logic cốt lõi của thuật toán `RuleBasedOptimizer` là gì?
12. Nó tạo ra các cặp điều chuyển ngẫu nhiên giữa nơi thừa và nơi thiếu.
13. Nó ưu tiên thực hiện các lệnh điều chuyển có số lượng lớn nhất trước.
14. Đây là một thuật toán tham lam (greedy), với mỗi mặt hàng đang thiếu, nó sẽ tìm nguồn hàng thừa có chi phí vận chuyển rẻ nhất để đáp ứng.
15. Nó xây dựng một đồ thị phức tạp và tìm đường đi ngắn nhất cho tất cả các lệnh điều chuyển.
16. Trong thuật toán di truyền (`GeneticAlgorithmOptimizer`), hàm `_calculate_fitness` đánh giá một “cá thể” dựa trên tiêu chí nào là chính?
17. Số lượng đơn vị hàng hóa được vận chuyển nhiều nhất.

18. Thời gian thực thi nhanh nhất của giải pháp.
19. Tổng chi phí vận chuyển của toàn bộ kế hoạch điều chuyển (càng thấp càng tốt).
20. Số lượng cửa hàng tham gia vào quá trình điều chuyển.
21. Tại sao thuật toán di truyền (GA) thường tìm ra giải pháp có tổng chi phí thấp hơn so với thuật toán dựa trên luật (Rule-based)?
22. Vì GA luôn chạy trong thời gian lâu hơn nên có nhiều thời gian để tính toán hơn.
23. Vì GA có khả năng thoát khỏi các “tối ưu cục bộ” bằng cách khám phá một không gian giải pháp rộng lớn hơn thông qua lai ghép và đột biến.
24. Vì GA luôn thực hiện ít lệnh điều chuyển hơn, do đó giảm chi phí.
25. Vì thuật toán dựa trên luật không tính đến chi phí vận chuyển.
26. Trong file `notebooks/01_full_process_step_by_step.ipynb`, mục đích chính của việc chạy hai thuật toán và so sánh kết quả là gì?
27. Để chứng minh rằng cả hai thuật toán đều cho ra cùng một kết quả chính xác.
28. Để kiểm tra xem Python đã được cài đặt đúng hay chưa.
29. Để đánh giá sự đánh đổi (trade-off) giữa tốc độ thực thi (Rule-based nhanh hơn) và chất lượng giải pháp (GA thường có chi phí thấp hơn).
30. Để xác định xem nên sử dụng Pandas hay NumPy cho việc phân tích.
31. Hàm `_repair_solution` trong `GeneticAlgorithmOptimizer` có vai trò gì?
32. Sửa các lỗi cú pháp trong mã nguồn của thuật toán.
33. Đảm bảo rằng một giải pháp (sau khi lai ghép hoặc đột biến) không vi phạm các ràng buộc cơ bản, ví dụ như chuyển nhiều hơn số lượng đang có.
34. Tối ưu hóa lại chi phí vận chuyển sau khi đã có giải pháp.

35. Làm tròn số lượng hàng hóa vận chuyển thành số nguyên gần nhất.
36. Dữ liệu trong file `transport_cost_matrix.csv` biểu diễn điều gì?
37. Chi phí để vận chuyển một đơn vị sản phẩm giữa hai cửa hàng bất kỳ.
38. Khoảng cách (km) giữa các cửa hàng.
39. Thời gian vận chuyển trung bình (giờ) giữa các cửa hàng.
40. Chi phí lưu kho cho mỗi sản phẩm tại mỗi cửa hàng.
41. Trong thuật toán di truyền, toán tử “lai ghép” (Crossover) thực hiện nhiệm vụ gì?
42. Thay đổi ngẫu nhiên một vài gen trong một cá thể để tạo sự đa dạng.
43. Đánh giá chất lượng của từng cá thể trong quần thể.
44. Kết hợp thông tin từ hai cá thể “cha mẹ” để tạo ra các cá thể “con” mới, kế thừa các đặc tính tốt.
45. Lựa chọn những cá thể tốt nhất từ quần thể hiện tại để giữ lại cho thế hệ sau.
46. Yếu tố nào sau đây là một ưu điểm của thuật toán RuleBasedOptimizer so với GeneticAlgorithmOptimizer?
47. Luôn tìm ra giải pháp có tổng chi phí vận chuyển thấp nhất.
48. Tốc độ thực thi rất nhanh và logic đơn giản, dễ hiểu.
49. Có khả năng xử lý các bài toán có nhiều mục tiêu cùng lúc.
50. Đảm bảo đáp ứng 100% nhu cầu của tất cả các cửa hàng.

1. **Code:** Các file code được đề cập trong bài có thể được tải tại [đây](#).
2. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại [đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng 3-4 tiếng.

AIO_QAs-Verified

🔒 Private group · 1.4K members



Hình 11: Hình ảnh group facebook AIO Q&A.

Phần VI

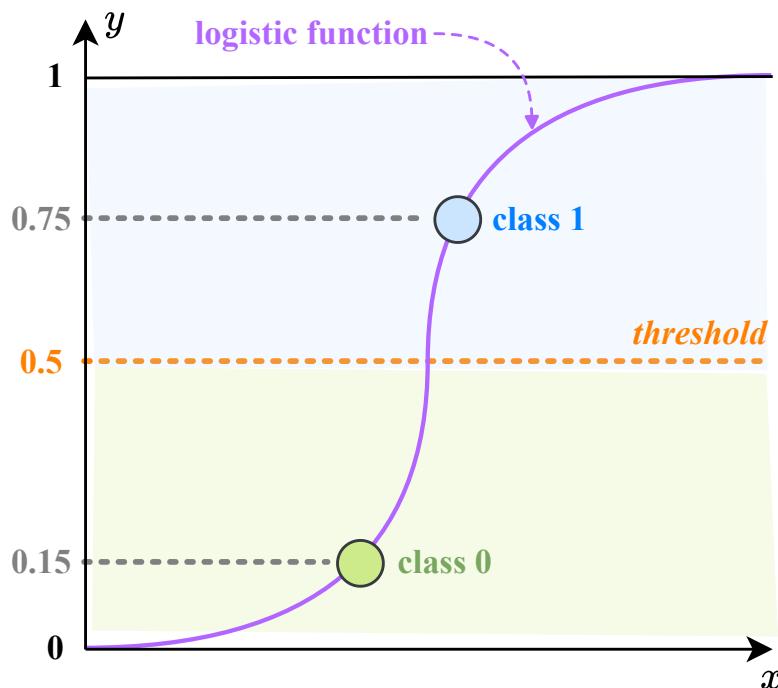
Module 6: Nền tảng cho deep learning (2)

Chương 26

Logistic Regression

26.1 Giới thiệu

Logistic Regression là một mô hình supervised learning trong học máy dùng để giải quyết bài toán phân loại nhị phân. Trong đó, thay vì dự đoán nhãn trực tiếp, mô hình ước lượng xác suất thuộc lớp dương bằng hàm *sigmoid* áp lên tổ hợp tuyến tính của đặc trưng và tối ưu với *Binary Cross-Entropy*. Logistic Regression thường được coi là một trong những kiến thức nền tảng, quan trọng khi tìm hiểu về học máy. Bởi lẽ, đây không chỉ là một phương thức giải quyết bài toán phân loại nhị phân hiệu quả, mà còn có liên hệ đến các kiến trúc phức tạp hơn về sau.



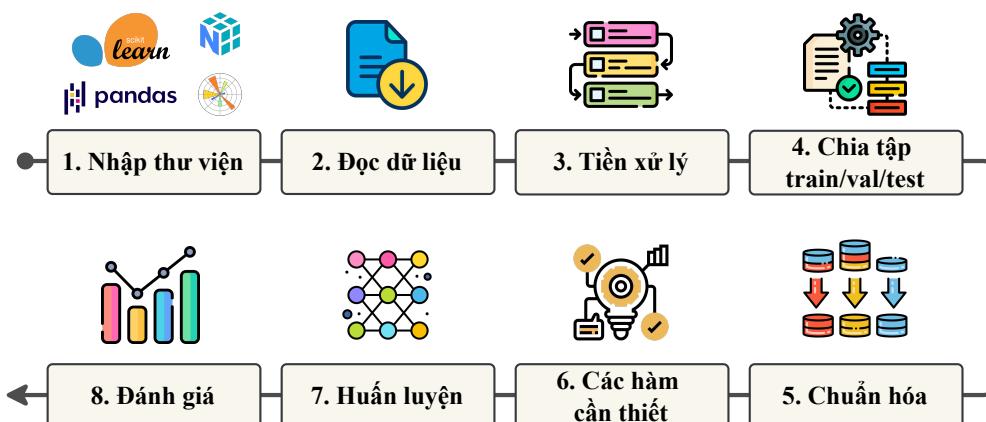
Hình 26.1: Ảnh minh họa về Logistic Regression.

Trong nội dung bài tập này, chúng ta sẽ thực hành với ba bài tập lập trình liên quan đến Logistic Regression. Cụ thể, có ba bài tập lập trình và các câu trắc nghiệm đi kèm. Phần cài đặt mô hình Logistic Regression sẽ ưu tiên triển khai từ đầu từng thành phần một. **Về nhiệm vụ chính**, các bạn sẽ cần phải dựa vào các file code gợi ý trong thư mục Hint ở phần Phụ lục và hoàn thành các phần code còn thiếu được yêu cầu trong mô tả của từng bài. Tóm tắt nội dung của các bài lập trình được thể hiện qua bảng sau:

Tổng quan các bài tập lập trình			
Bài	Mục tiêu	Biểu diễn/Đầu vào	Hàm mục tiêu/Đánh giá
1	Xây dựng Logistic Regression từ đầu cho Titanic, huấn luyện và đánh giá.	Vector đặc trưng đã chuẩn hóa, thêm bias, $X \in \mathbb{R}^{n \times d}$, $y \in \{0, 1\}^n$.	Binary Cross-Entropy, theo dõi loss và accuracy, so sánh theo các epoch.
2	Pipeline tiền xử lý văn bản và huấn luyện Logistic Regression cho Twitter Sentiment.	Đặc trưng 3 chiều [1, #neg, #pos] và chuẩn hóa.	Binary Cross-Entropy, đánh giá accuracy trên validation và test, đường trực quan đường hội tụ train và validation.
3	Xây dựng Logistic Regression cho bài toán phân loại đa lớp bằng chiến lược One-vs-All.	Ảnh 28×28 làm phẳng $\rightarrow 784$ chiều, thêm bias; $X \in \mathbb{R}^{n \times d}$ và chuẩn hóa, $y \in \{0, \dots, 9\}^n$.	Macro Binary Cross-Entropy qua 10 lớp (OvA), accuracy đa lớp (argmax) trên validation/test, trực quan đường hội tụ train/validation.

Mặc dù có thể có sự khác biệt về bối cảnh ứng dụng của Logistic Regression, việc cài đặt và huấn luyện mô hình này đều tuân theo quy trình chung như sau:

- 1. Nhập thư viện:** Tải các gói cho xử lý mảng, chia tập, chuẩn hóa, trực quan hóa.
- 2. Chuẩn bị dữ liệu:** Xác định nguồn dữ liệu, tách X , y , thêm bias vào cột đầu tiên, chia train/validation/test theo tỉ lệ, *fit* scaler trên train và *transform* đồng nhất cho các tập còn lại.
- 3. Định nghĩa mô hình/loss/metric:** Hàm sigmoid, `predict(X, theta)`, `compute_loss` (Binary Cross-Entropy), `compute_gradient`, `update_theta`, và `compute_accuracy`.
- 4. Huấn luyện:** Khởi tạo θ , lặp qua epoch và mini-batch: tính dự đoán \hat{y} , loss, gradient, cập nhật θ , ghi lại loss/accuracy cho train/validation để theo dõi hội tụ.
- 5. Đánh giá và trực quan:** Báo cáo accuracy trên validation/test, vẽ biểu đồ loss/accuracy để phân tích ổn định hội tụ và hiện tượng overfitting.



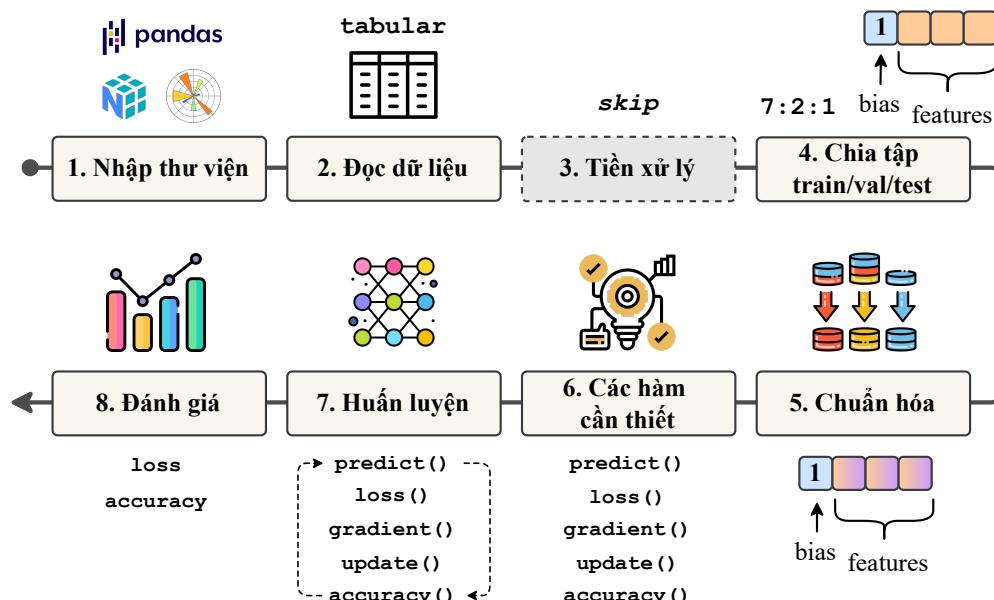
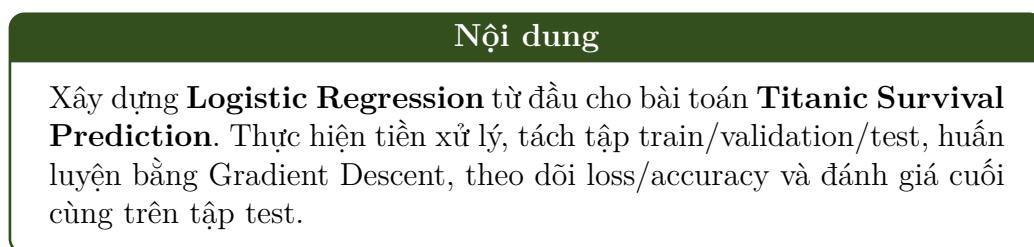
Hình 26.2: Quy trình chung cho từng các bước cài đặt xây dựng Logistic Regression.

Thông qua các bước trên, ở phần tiếp theo ta sẽ lần lượt triển khai các bài tập thực hành đã đề ra.

26.2 Thực hành

26.2.1 Bài tập 1: Titanic Survival Prediction

Phát biểu bài toán



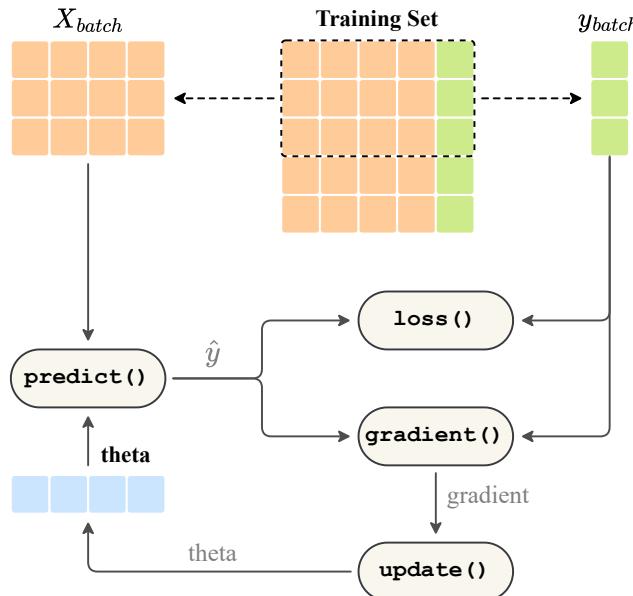
Hình 26.3: Quy trình tổng quát cho bài tập 1.

Trong bài tập này, ta làm quen với Logistic Regression thông qua việc xây dựng mô hình từ đầu, từng thành phần một và sau đó thực hiện huấn luyện mô hình với bài toán dự đoán xác suất sống sót của một người trên con tàu Titanic. Các bước tính toán trong quá trình huấn luyện mô hình này có thể được minh họa qua Hình 26.4. Việc ước lượng được thông qua hàm sigmoid

áp dụng lên tổ hợp tuyến tính của đặc trưng cho trước của bộ dữ liệu. Việc chuẩn hóa đặc trưng, giúp tối ưu ổn định hơn, chỉ số đánh giá chính là Binary Cross-Entropy và accuracy.

Mục tiêu của bài tập

- Tự cài đặt đầy đủ các hàm: `sigmoid`, `predict`, `compute_loss` (BCE), `compute_gradient`, `update_theta`, `compute_accuracy`.
- Thiết lập pipeline: thêm bias, chia train/validation/test, fit scaler trên train và transform đồng nhất các tập còn lại.
- Huấn luyện bằng mini-batch gradient descent, ghi nhận đường hội tụ loss/accuracy trên train/validation qua các epoch.
- Đánh giá độ tổng quát hóa bằng accuracy trên validation và test.



Hình 26.4: Các bước tính toán trong quá trình huấn luyện Logistic Regression.

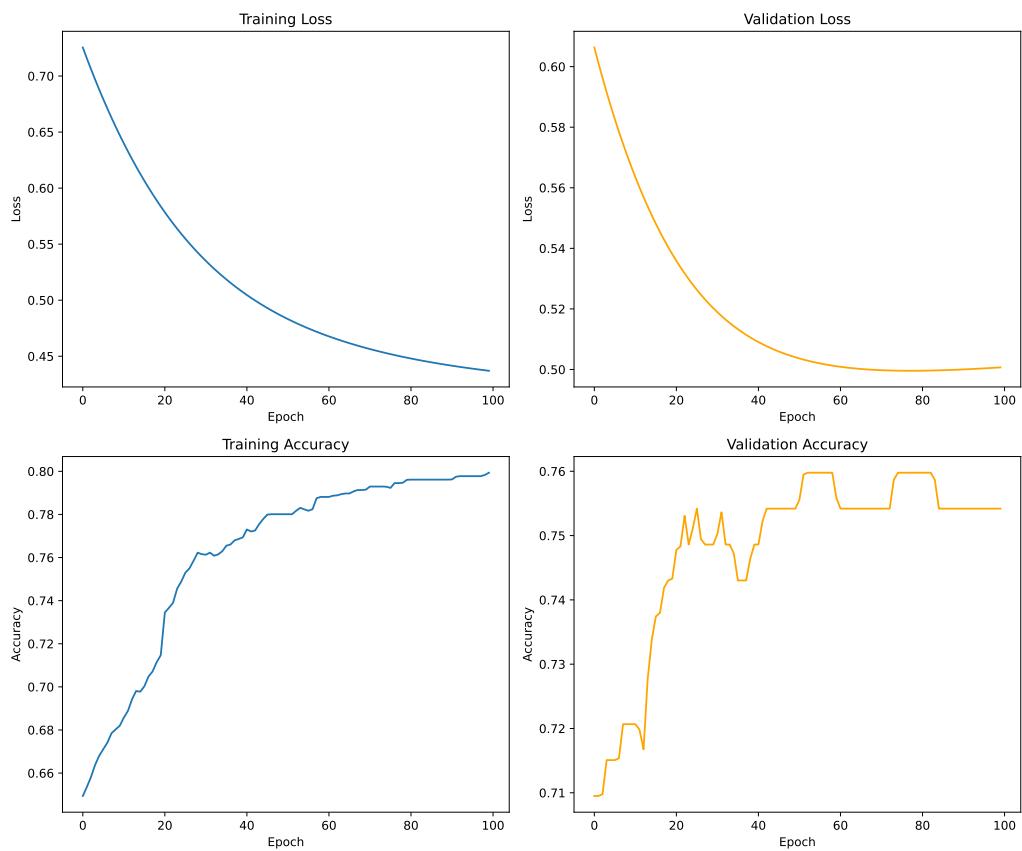
Yêu cầu

Dựa vào mã trong phần gợi ý, hãy hoàn thiện các phần còn thiếu để chạy trọn vẹn pipeline:

- **Tiền xử lý:** tách X, y , thêm cột bias vào cột đầu tiên, chia train/validation/test theo tỉ lệ, chuẩn hóa các cột đặc trưng nhưng không chuẩn hóa bias.
- **Hàm cốt lõi:** hiện thực chính xác các hàm ở mục tiêu, clip \hat{y} để tránh $\log(0)$ khi tính BCE.
- **Vòng lặp huấn luyện:** cập nhật θ theo learning rate, lưu lịch sử train/validation loss và accuracy mỗi epoch.
- **Trực quan hóa:** vẽ các đồ thị gồm train/validation loss và train/validation accuracy theo epoch.

Bảng cấu hình siêu tham số – Bài 1 (Titanic)

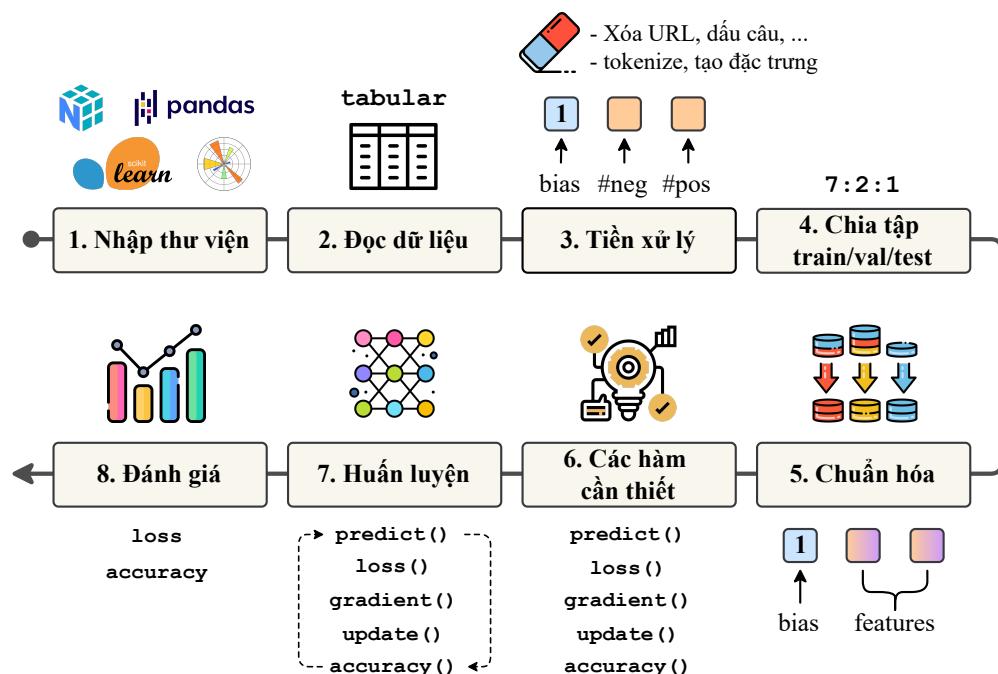
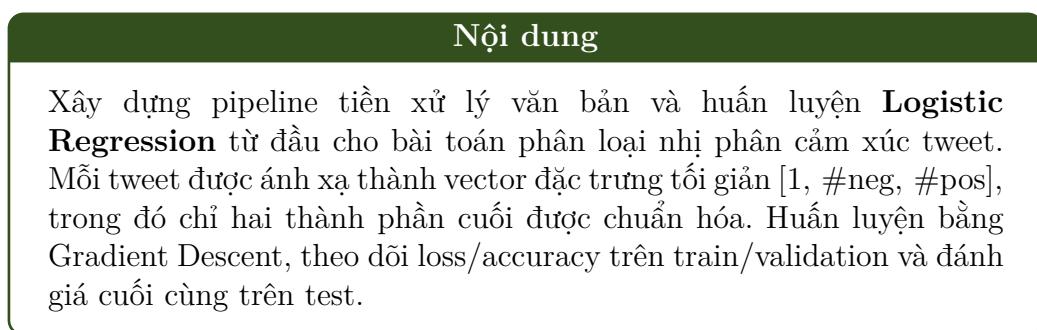
Tham số	Giá trị
Learning rate (lr)	0.005
Số epoch (epochs)	100
Batch size (batch_size)	32
Val/Test split	0.2 / 0.125
Seed cố định (random_state)	2



Hình 26.5: Minh họa kết quả loss/accuracy thu được trên tập train và val của bộ Titanic trong quá trình huấn luyện.

26.2.2 Bài tập 2: Twitter Sentiment Analysis

Phát biểu bài toán



Hình 26.6: Quy trình tổng quát cho bài tập 2.

Trong bài tập này, ta ứng dụng Logistic Regression dùng để giải quyết bài toán về phân loại nhị phân trên dữ liệu văn bản. Cụ thể, ta cần xác định cảm xúc của các tweet được thu thập từ trang mạng xã hội Twitter (nay được gọi là mạng xã hội X) là tích cực hay tiêu cực. Với Logistic Regression

là trung tâm, ta sẽ cần có một cách thức tạo đặc trưng văn bản phù hợp cho bài toán này để có thể giúp mô hình hiểu rõ kiểu dữ liệu này tốt hơn, từ đó hướng đến việc cải thiện hiệu suất của mô hình.

Mục tiêu của bài tập

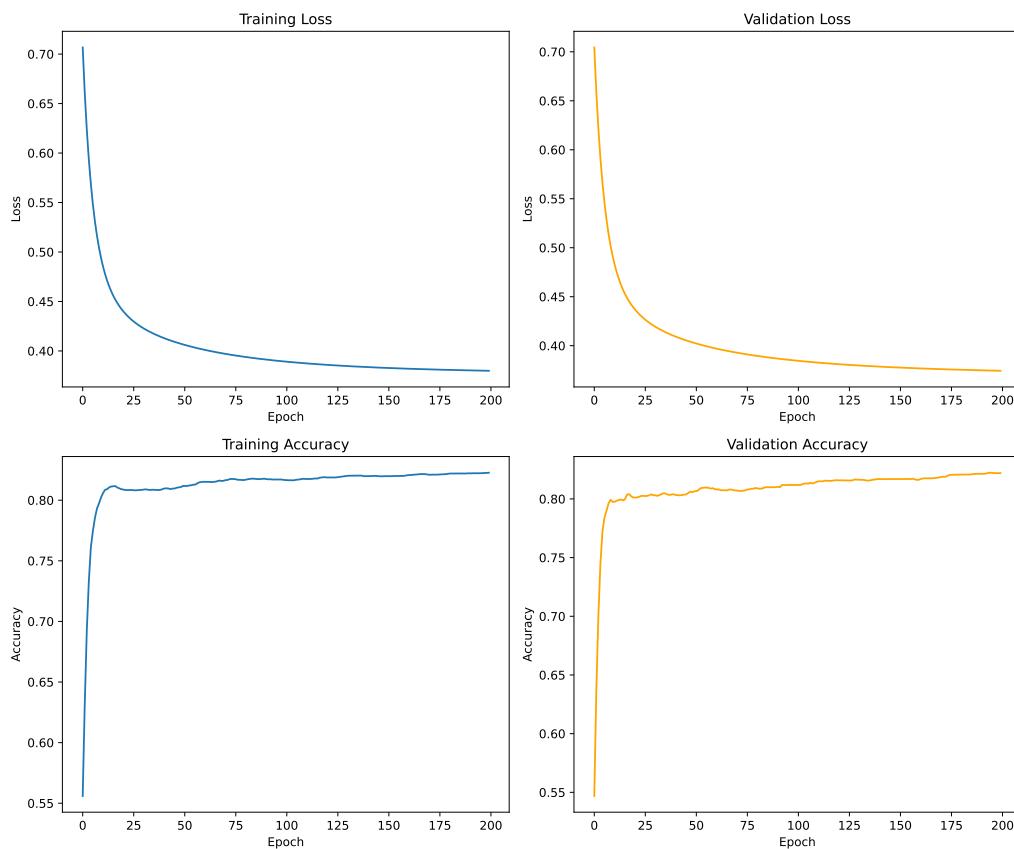
- Hoàn thiện pipeline NLP tối giản: chuẩn hóa văn bản, xây dựng từ điển tần suất theo nhãn, và ánh xạ mỗi tweet thành vector [1, #neg, #pos].
- Tái sử dụng các hàm cốt lõi Logistic Regression (Bài 28.2.1) để huấn luyện/đánh giá.
- Trực quan hóa đường hội tụ loss/accuracy và so sánh xu hướng với bài Titanic.

Yêu cầu

- **Chuẩn hóa văn bản:** bỏ “RT”, liên kết, dấu câu, dấu #, tokenization, chữ thường, bỏ mention.
- **Từ điển tần suất:** đếm số lần xuất hiện của mỗi token theo nhãn (negative/positive) trên toàn bộ dữ liệu.
- **Vector hóa:** mỗi văn bản thành [1, #neg, #pos] theo từ điển tần suất.
- **Chia dữ liệu:** train/validation/test, fit scaler trên train và transform nhất quán cho validation/test.
- **Huấn luyện:** mini-batch gradient descent, ghi nhận loss/accuracy trên tập train/validation theo epoch, clip \hat{y} khi tính BCE.
- **Đánh giá:** accuracy trên validation/test và vẽ các đường hội tụ.

Bảng cấu hình siêu tham số – Bài 2 (Twitter)	
Tham số	Giá trị
Learning rate (<code>lr</code>)	0.01
Số epoch (<code>epochs</code>)	200
Batch size (<code>batch_size</code>)	128
Train/Val/Test split	7/2/1
Seed cố định (<code>random_state</code>)	2

Sau khi hoàn thiện, kỳ vọng kết quả trực quan tương tự:



Hình 26.7: Minh họa kết quả loss/accuracy thu được trên tập train và val của bộ Twitter Sentiment trong quá trình huấn luyện.

26.2.3 Bài tập 3: MNIST Classification (One-vs-All)

Phát biểu bài toán

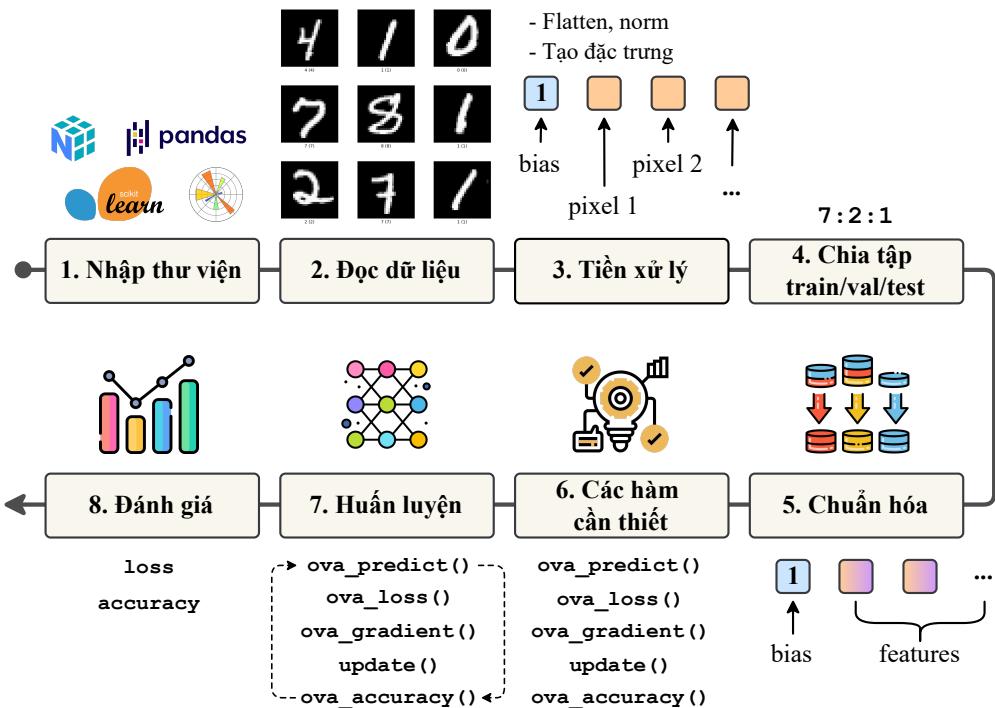
Nội dung

Xây dựng mô hình **Logistic Regression đa lớp** cho bộ dữ liệu **MNIST (10 lớp)** bằng chiến lược **One-vs-All (OvA)**. Ta huấn luyện đồng thời $K = 10$ mô hình phân loại nhị phân, mỗi mô hình tách *một* lớp (k) khỏi *phần còn lại*. Với đặc trưng ảnh được làm phẳng thành vector 1 chiều (784 chiều) đã *chuẩn hóa trừ bias*, dự đoán xác suất theo lớp k là:

$$p_k(\mathbf{x}) = \sigma(\mathbf{w}_k^\top \mathbf{x}), \quad \hat{y} = \arg \max_{k \in \{0, \dots, 9\}} p_k(\mathbf{x}).$$

Hàm mất mát dùng **macro BCE** trên K bộ phân loại:

$$\mathcal{L} = \frac{1}{K} \sum_{k=0}^{K-1} \text{BCE}(p_k(\mathbf{x}), \mathbb{1}[y=k]).$$



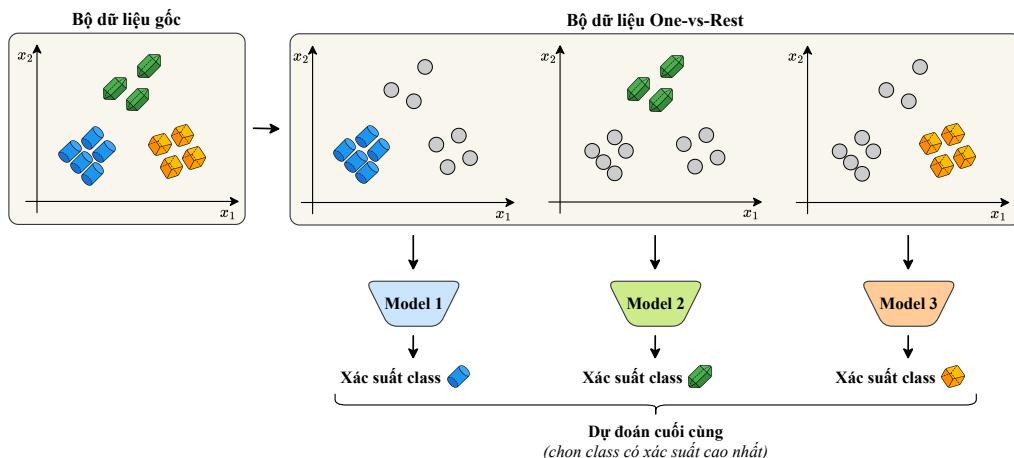
Hình 26.8: Quy trình tổng quát cho bài tập 3.

Khởi nguồn, Logistic Regression được thiết kế cho phân loại nhị phân, tức chỉ có hai lớp. Vậy khi đổi mới với bài toán **multi-class classification** - tức có nhiều hơn hai lớp, chúng ta mở rộng mô hình như thế nào? Trong bài tập này, ta sẽ tìm hiểu cài đặt một chiến lược nhằm giải quyết vấn đề trên với tên gọi là **One-vs-All (OvA)**: huấn luyện đồng thời K bộ phân loại nhị phân, mỗi bộ tách *một* lớp dương khỏi *phần còn lại*. Ở bước dự đoán, mô hình chọn lớp có xác suất cao nhất trong K xác suất dự đoán.

Trong bài tập này, ta sẽ thực hành với bộ dữ liệu **MNIST** với bài toán phân biệt chữ số viết tay trong một ảnh. Bộ dữ liệu gồm 10 chữ số, tương ứng với 10 lớp ($0, 1, 2, \dots, 9$). Khi tiền xử lý dữ liệu, mỗi ảnh 28×28 được làm phẳng và chuẩn hóa đặc trưng để huấn luyện các “đầu ra” OvA song song. Cách tiếp cận này giúp chúng ta giữ nguyên các thành phần cốt lõi của Logistic Regression trong khi mở rộng khả năng dự đoán từ hai lớp sang nhiều lớp theo một cơ chế gọn nhẹ, dễ diễn giải và hiệu quả.

Mục tiêu của bài tập

- Hiểu và hiện thực hóa chiến lược One-vs-All (OvA) cho Logistic Regression phân loại đa lớp trên bộ MNIST ($K=10$).
- Tái sử dụng pipeline LR từ Bài 28.2.1: thêm bias, chuẩn hóa đặc trưng nhưng *trừ cột bias*, chia tập train/validation/test, huấn luyện bằng mini-batch gradient descent.
- Cài đặt các hàm cốt lõi cho chiến lược OvA: dự đoán xác suất từng lớp, macro BCE đa lớp, gradient cho từng head, và accuracy đa lớp.
- Trực quan hóa với biểu đồ đường hội tụ loss/accuracy trên tập train/validation, đánh giá trên tập test, và so sánh xu hướng với hai bài trước.



Hình 26.9: Minh họa chiến lược One-vs-All (OvA). Với 3 lớp đối tượng cần phân loại, ta có thể coi nó giống như việc triển khai với dạng phân loại nhị phân ba lần, mỗi lần ta cố gắng phân biệt giữa một lớp so với các lớp còn lại của bộ dữ liệu.

Yêu cầu

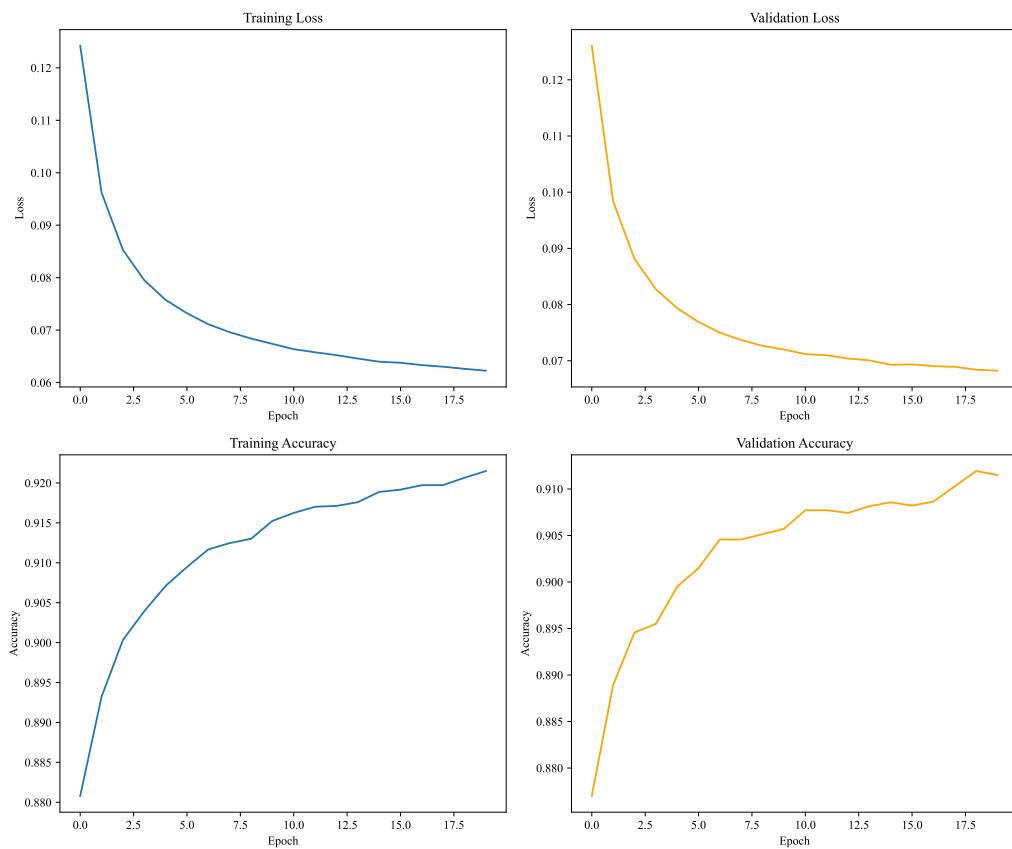
- Tính xác suất theo chiến lược OvA:** hiện thực suy luận xác suất cho từng lớp bằng cách kết hợp dữ liệu (đã có cột bias) với tham số tương ứng của mỗi lớp; kết quả là một ma trận xác suất cho toàn bộ

batch. Yêu cầu viết theo hướng *vector hóa* hoàn toàn, không làm thay đổi cột bias.

- **Hàm măt mát đa lớp (macro-BCE):** xây dựng hàm măt mát thu được bằng cách tính Binary Cross-Entropy cho từng lớp theo thiết lập OvA, rồi lấy trung bình trên tất cả các lớp để được một giá trị vô hướng đại diện cho toàn bộ bài toán. Sử dụng thao tác *clip* xác suất có sẵn để tránh lỗi số học.
- **Gradient gộp cho OvA:** tính đạo hàm cho toàn bộ tham số của các “đầu ra” OvA trong một bước duy nhất, ưu tiên triển khai vector hóa để tránh vòng lặp theo lớp; đảm bảo kích thước kết quả khớp với ma trận tham số mô hình.
- **Hàm tính độ chính xác cho bài toán phân loại đa lớp:** suy ra nhãn dự đoán bằng cách chọn lớp có xác suất lớn nhất cho mỗi mẫu và tính tỉ lệ dự đoán đúng trên batch.

Bảng cấu hình siêu tham số – Bài 3 (MNIST OvA)

Tham số	Giá trị
Learning rate (lr)	0.1
Số epoch (epochs)	20
Batch size (batch_size)	256
Train/Val/Test split	7/2/1
Seed cố định (random_state)	2

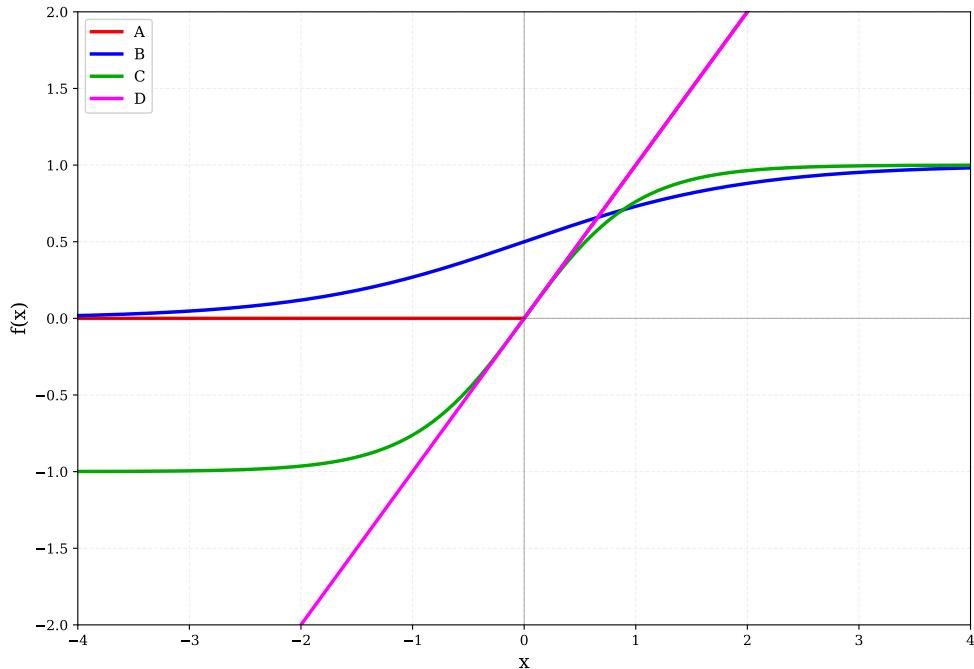


Hình 26.10: Minh họa kết quả loss/accuracy thu được trên tập train và val của bộ MNIST trong quá trình huấn luyện.

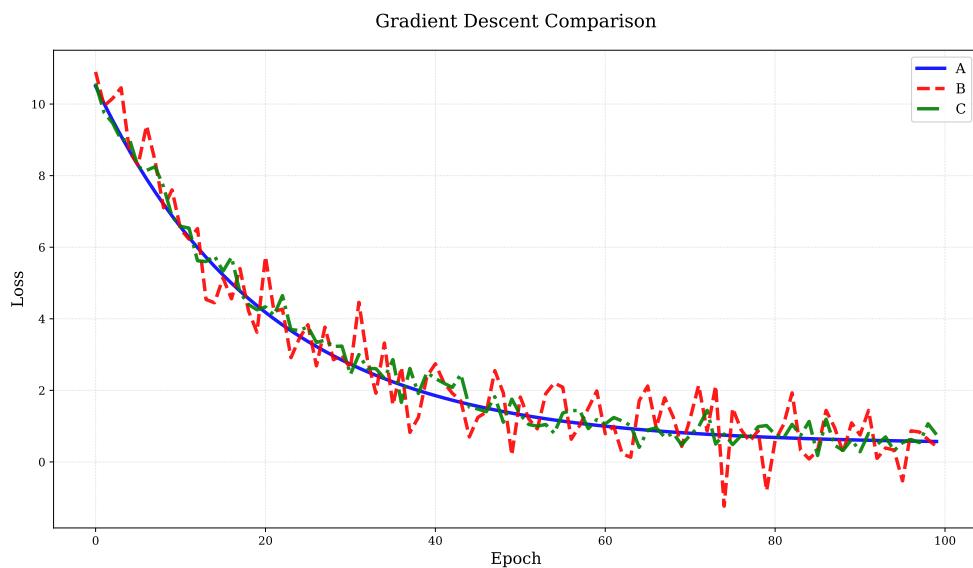
26.3 Câu hỏi trắc nghiệm

1. Tại sao Logistic Regression được gọi là “Regression” mặc dù giải quyết bài toán phân loại?
 - (a) Vì nó ước lượng xác suất (giá trị liên tục) trước khi phân loại.
 - (b) Vì nó sử dụng công thức tương tự Linear Regression.
 - (c) Vì nó chỉ áp dụng cho bài toán hồi quy.
 - (d) Vì tên gọi được đặt nhầm.
2. Giá trị $z = \theta^T X$ trong Logistic Regression được gọi là gì?
 - (a) Activation value.
 - (b) Logit hoặc log-odds.
 - (c) Probability score.
 - (d) Loss value.
3. Đạo hàm của hàm Sigmoid $\sigma(z)$ có công thức nào sau đây?
 - (a) $\sigma'(z) = \sigma(z)$.
 - (b) $\sigma'(z) = 1 - \sigma(z)$.
 - (c) $\sigma'(z) = \sigma(z)^2$.
 - (d) $\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$.
4. Tại sao cần clip giá trị dự đoán \hat{y} (như ví dụ trong bài là giữa 10^{-7} và $1 - 10^{-7}$) khi tính Binary Cross-Entropy?
 - (a) Để tăng độ chính xác của mô hình.
 - (b) Để tránh tính logarit của 0, gây ra lỗi số học.
 - (c) Để giảm thời gian tính toán.
 - (d) Để chuẩn hóa dữ liệu.
5. Đồ thị nào dưới đây biểu diễn hàm Sigmoid được sử dụng trong Logistic Regression?
 - (a) Đồ thị A.

- (b) Đồ thị B.
- (c) Đồ thị C.
- (d) Đồ thị D.



6. Quan sát biểu đồ hội tụ của ba phương pháp Gradient Descent dưới đây gồm Stochastic Gradient Descent (SGD), Mini-batch Gradient Descent và Batch Gradient Descent. Đường nào biểu diễn SGD?



- (a) Đường màu xanh lam (mịn nhất).
- (b) Đường màu xanh lá (dao động vừa phải).
- (c) Đường màu đỏ (dao động mạnh nhất).

7. Cho đoạn chương trình sau:

```

1 def predict(X, theta):
2     z = np.dot(X, theta)
3     return 1 / (1 + np.exp(-z))

```

Khi truyền vector `X = [[-3.2, 0.5, 2.0, 1]]` và vector `theta = [0-0.4, -0.3, 0.2, -0.1]` vào hàm `predict()` trên, kết quả trả về là:

- (a) 0.87988114.
- (b) 0.11868901.
- (c) 0.75620341.
- (d) 0.24416110.

8. Cho đoạn chương trình sau:

```

1 def compute_loss(y_hat, y):
2     return (-y * np.log(y_hat) - (1 - y) * np.log(1 - y_hat)).mean()

```

Khi truyền vector $y = \text{np.array}([1, 1, 0, 0])$ và vector $y_hat = \text{np.array}([0.9, 0.6, 0.2, 0.4])$ vào hàm `compute_loss()` trên, kết quả trả về của hàm là (làm tròn đến hàng thập phân thứ 3):

- (a) 0.338.
- (b) 0.225.
- (c) 0.511.
- (d) 0.693.

9. Cho đoạn chương trình sau:

```

1 def compute_gradient(X, y_true, y_pred):
2     gradient = np.dot(X.T, (y_pred - y_true)) / y_true.size
3     return gradient

```

Khi truyền $X = [[1, 3], [2, 1], [3, 2], [1, 2]]$, $y_true = [1, 0, 1, 0]$ và $y_pred = [0.7, 0.3, 0.8, 0.2]$ vào hàm `compute_gradient()` trên, kết quả trả về là:

- (a) $[-0.050, -0.200]$.
- (b) $[0.025, -0.125]$.
- (c) $[-0.025, -0.150]$.
- (d) $[0.100, 0.150]$.

10. Cho đoạn chương trình sau:

```

1 def compute_accuracy(y_true, y_pred):
2     y_pred_rounded = np.round(y_pred)
3     accuracy = np.mean(y_true == y_pred_rounded)
4     return accuracy

```

Khi truyền vector `y_true = [1, 0, 1, 0, 1]` và `y_pred = [0.84, 0.49, 0.40, 0.20, 0.77]` vào hàm `compute_accuracy()` trên, kết quả trả về là:

- (a) 0.50.
- (b) 0.60.
- (c) 0.70.
- (d) 0.80.

1. **Dataset:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
2. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
3. **Solution:** Các file code đáp án có thể được tải tại [đây](#).
4. **Rubric:**

Mục	Kiến thức	Đánh giá
I.	<ul style="list-style-type: none"> - Kiến thức về việc cài đặt chương trình huấn luyện mô hình Logistic Regression từ đầu. - Kiến thức về các hàm thành phần trong Logistic Regression. - Kiến thức về thuật toán Gradient Descent. - Kiến thức về một số phương pháp tiền xử lý dữ liệu bảng và dữ liệu văn bản. - Kiến thức về chiến lược One-vs-All dùng để giải quyết bài toán phân loại đa lớp với mô hình Logistic Regression. 	<ul style="list-style-type: none"> - Nắm được cách cài đặt mô hình Logistic Regression từ đầu. - Nắm được các hàm thành phần liên quan đến Logistic Regression. - Hiểu được cách vận hành của thuật toán Gradient Descent.
II.	<ul style="list-style-type: none"> - Các kiến thức và nội dung cơ bản về Logistic Regression. 	- Nắm được kiến thức cơ bản về Logistic Regression.

5. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại [đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 tiếng.

AIO_QAs-Verified

🔒 Private group · 1.4K members



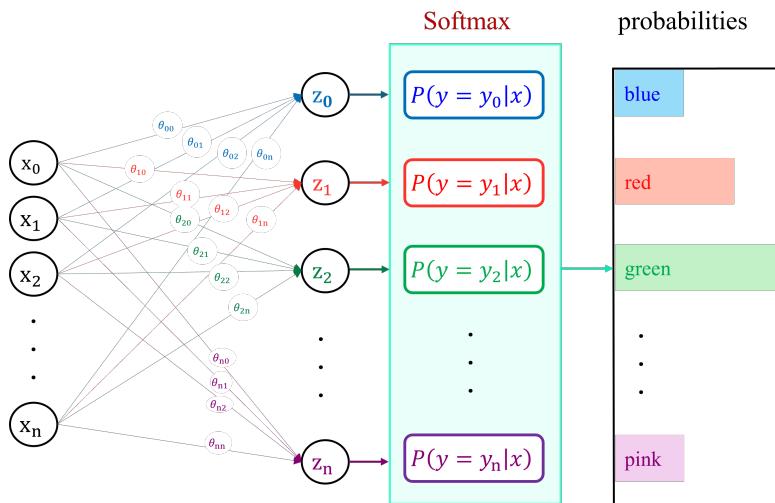
Hình 26.11: Hình ảnh group facebook AIO Q&A.

Chương 27

Softmax Regression

27.1 Giới thiệu

Softmax Regression là một trong những thuật toán học máy có giám sát (supervised learning) nền tảng, được thiết kế chuyên biệt để giải quyết bài toán phân loại đa lớp (multiclass classification). Điểm mấu chốt của thuật toán này là việc sử dụng **hàm Softmax**, có chức năng chuyển đổi các điểm số (logits) thô từ mô hình thành một phân bố xác suất trên tất cả các lớp. Kết quả đầu ra này cho phép chúng ta ước lượng khả năng một mẫu dữ liệu thuộc về từng lớp và đưa ra dự đoán một cách linh hoạt.



Hình 12: Mô phỏng đơn giản về Softmax Regression trong mạng nơ-ron.

Trong bài tập này, chúng ta sẽ vừa tìm hiểu các khái niệm lý thuyết cốt lõi của Softmax Regression, vừa đi sâu vào thực hành bằng cách lập trình mô hình từ đầu (from scratch). Mô hình này sẽ được áp dụng để giải quyết hai bài toán thực tế: thứ nhất là Credit Card Fraud Detection (2 lớp), một bài toán có độ mất cân bằng dữ liệu (imbalanced data) cực kỳ nặng, và thứ hai là Twitter Sentiment Analysis (3 lớp), tập trung vào việc xử lý dữ liệu dạng văn bản.

27.2 Chi tiết về Softmax Regression

Softmax Regression là thuật toán học có giám sát dùng cho phân loại đa lớp, trong đó đầu ra được diễn giải như một phân bố xác suất trên các nhãn. Ý tưởng cốt lõi là biến các *logit* tuyến tính thành xác suất chuẩn hóa, cho phép so sánh, đặt ngưỡng và ra quyết định theo xác suất một cách tự nhiên.

Sau đây, chúng ta sẽ cùng tìm hiểu về các thành phần có trong softmax và ví dụ tính toán để hiểu rõ hơn:

Cho đặc trưng vào $\mathbf{x} \in \mathbb{R}^d$, trọng số $\mathbf{W} \in \mathbb{R}^{K \times d}$ và độ lệch $\mathbf{b} \in \mathbb{R}^K$. Vector *logit* được xác định bởi

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \in \mathbb{R}^K, \quad \mathbf{z} = (z_1, \dots, z_K)^\top,$$

trong đó z_i biểu diễn điểm số (*logit*) gán cho lớp thứ i .

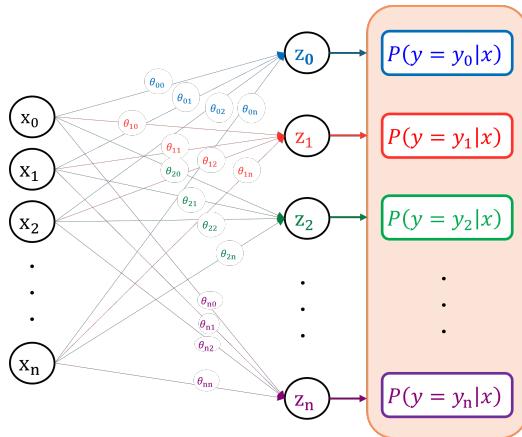
Công thức hàm softmax cho từng lớp:

Xác suất dự đoán cho lớp i là

$$p_i = \text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^K \exp(z_j)}, \quad i \in \{1, \dots, K\}.$$

Trong đó:

- i (chỉ số tự do): xác định *thành phần đầu ra* đang xét trong vector xác suất $\mathbf{p} = (p_1, \dots, p_K)^\top$. Với mỗi i cố định, biểu thức cho p_i trả về một số thực duy nhất trong $[0, 1]$. Khi i thay đổi từ 1 đến K , ta thu được toàn bộ vector \mathbf{p} .
- j (chỉ số bị ràng buộc trong tổng): là chỉ số *chạy* dùng để cộng qua *tất cả* các lớp trong mẫu số, bảo đảm phép chuẩn hóa để tổng xác suất bằng 1. Chỉ số j chỉ xuất hiện bên trong $\sum_{j=1}^K$, không xuất hiện bên ngoài.



Hình 13: Mô phỏng hàm softmax trong mạng nơ-ron.

Ràng buộc chuẩn hoá và miền giá trị. Nhờ tổng ở mẫu số, ta luôn có

$$\sum_{i=1}^K p_i = 1, \quad p_i \geq 0 \quad \forall i$$

Ví dụ chi tiết.

Cho ba logit $\mathbf{z} = [2.0, 1.0, 0.1]$.

Tính từng bước cho xác suất lớp thứ nhất p_1 :

$$\exp(2.0) = 7.3891, \quad \exp(1.0) = 2.7183, \quad \exp(0.1) = 1.1052,$$

$$S = \sum_{j=1}^3 \exp(z_j) = 7.3891 + 2.7183 + 1.1052 = 11.2126,$$

$$p_1 = \frac{\exp(2.0)}{S} = \frac{7.3891}{11.2126} \approx 0.6590.$$

Khi đó hai xác suất còn lại lần lượt là

$$p_2 = \frac{\exp(1.0)}{S} = \frac{2.7183}{11.2126} \approx 0.2424, \quad p_3 = \frac{\exp(0.1)}{S} = \frac{1.1052}{11.2126} \approx 0.0986.$$

💡 Mở rộng: Ổn định số (Numerical Stability) 💡

Khi các giá trị z_i lớn, biểu thức $\exp(z_i)$ có thể gây **tràn số** (overflow). Ví dụ, với kiểu `float64` trong Python, $\exp(x)$ sẽ tràn khi $x \gtrsim 709$, vì $e^{709} \approx 8.2 \times 10^{307}$ đã gần vượt quá giá trị cực đại mà kiểu `float` có thể biểu diễn ($\approx 1.8 \times 10^{308}$).

Để tránh điều này, ta thường trừ đi giá trị lớn nhất của vector trước khi tính mũ:

$$m = \max_j z_j$$

$$p_i = \frac{\exp(z_i - m)}{\sum_j \exp(z_j - m)}.$$

Do hàm Softmax **không thay đổi** khi cộng hoặc trừ cùng một hằng số cho toàn bộ vector z , nên phép trừ này không làm thay đổi kết quả mà chỉ giúp tính toán ổn định hơn về mặt số học.

Tương tự, khi làm việc với *log-xác suất* (ví dụ trong hàm mất mát cross-entropy), ta thường phải tính giá trị dạng $\log \sum_j e^{z_j}$. Tổng này có thể rất lớn, dễ gây tràn số, nên ta sử dụng mẹo **log-sum-exp** để ổn định hơn:

$$\log \sum_j e^{z_j} = m + \log \sum_j e^{z_j - m}, \quad \text{với } m = \max_j z_j.$$

Biến đổi này giữ nguyên giá trị toán học, nhưng giới hạn các số mũ $e^{z_j - m} \leq 1$, giúp tránh tràn và duy trì độ chính xác số học.

27.3 Softmax với tham số temperature

Để điều khiển độ tập trung của phân bố xác suất suy ra từ các logits cố định, ta đưa vào tham số $\tau > 0$ (temperature), tham số này thực hiện phép co-giãn trên các giá trị logit trước khi chuẩn hoá bằng hàm softmax, từ đó điều chỉnh mức độ phân tán (entropy) của phân bố kết quả. Với logits $\mathbf{z} \in \mathbb{R}^K$ và tham số $\tau > 0$, phân bố

$$p_i(\tau) = \text{softmax}\left(\frac{\mathbf{z}}{\tau}\right)_i = \frac{\exp(z_i/\tau)}{\sum_{j=1}^K \exp(z_j/\tau)}, \quad i = 1, \dots, K.$$

Trường hợp chuẩn là $\tau = 1$.

Khi τ tăng, phép co giãn $z_i \mapsto z_i/\tau$ làm giảm độ chênh lệch giữa các hạng $\exp(z_i/\tau)$, do đó entropy $H(p(\tau)) = -\sum_{i=1}^K p_i(\tau) \log p_i(\tau)$ sẽ tăng theo τ (với \mathbf{z} cố định).

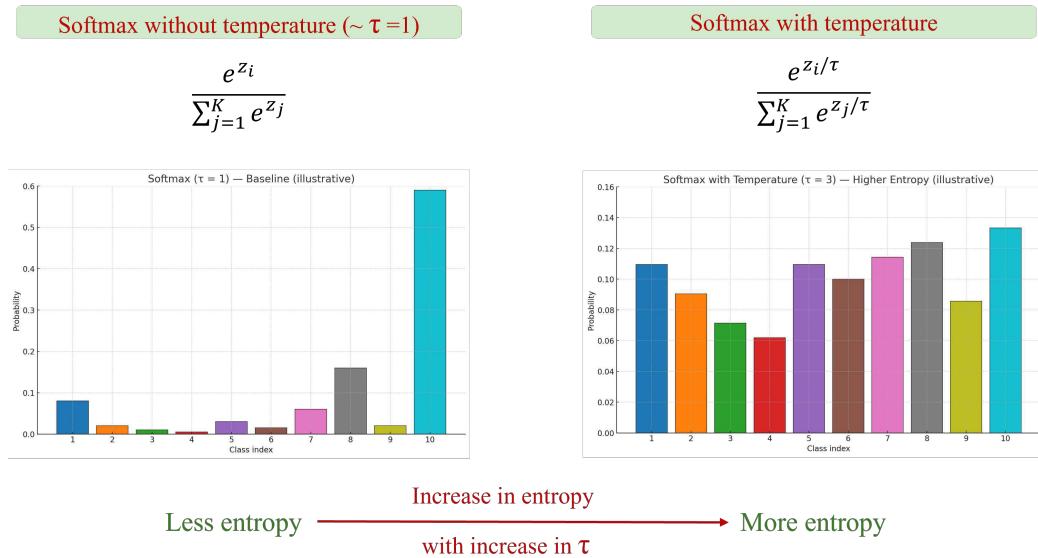
Các giới hạn:

$$\lim_{\tau \rightarrow 0^+} p(\tau) = \mathbf{e}_{\arg \max_i z_i}$$

(phân bố tiến dần đến vector one-hot, tức chỉ giữ lại lớp có logit lớn nhất; entropy cực tiểu).

$$\lim_{\tau \rightarrow \infty} p(\tau) = \left(\frac{1}{K}, \dots, \frac{1}{K} \right)$$

(phân bố tiến đến đều tuyệt đối trên K lớp; entropy cực đại).



Hình 14: Minh họa sự ảnh hưởng của temperature trong softmax.

💡 Mở rộng: Sampling trong LLM 💡

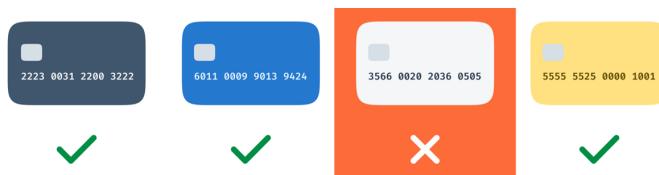
Sau khi thu được phân bố xác suất $p(\tau)$ trên toàn bộ tập từ vựng, mô hình ngôn ngữ lớn (LLM) thực hiện quá trình *lấy mẫu* (sampling) để chọn token kế tiếp dựa trên các quy tắc ngắn gọn sau:

- **Greedy:** Ở mỗi bước chọn token có xác suất cao nhất dựa trên chuỗi đã sinh. Rất nhanh và ổn định nhưng ít đa dạng và không đảm bảo chuỗi tốt nhất (MAP); muốn gần MAP hơn thường dùng beam search.
- **Beam search:** Giữ lại k ứng viên tốt nhất sau mỗi bước, mở rộng rồi chọn tiếp k chuỗi có tổng log-xác suất cao nhất (thường kèm chuẩn hoá độ dài). Thường cho chất lượng cao hơn greedy nhưng tốn thời gian/bộ nhớ hơn và vẫn không tối ưu toàn cục.
- **Top- k sampling:** chỉ giữ lại k phần tử có xác suất cao nhất trong $p(\tau)$ và loại bỏ các phần tử còn lại. Phân bố được chuẩn hoá lại trên tập con này rồi thực hiện lấy mẫu ngẫu nhiên. Cách này giới hạn không gian chọn mẫu, đảm bảo token được sinh ra nằm trong vùng xác suất cao nhất, giảm khả năng xuất hiện từ hiềm.
- **Nucleus sampling (Top- p):** xác định tập con nhỏ nhất của các token có tổng xác suất tích luỹ không nhỏ hơn một ngưỡng p cho trước. Việc chuẩn hoá và lấy mẫu chỉ được thực hiện trong tập con này. Cách chọn theo *khối xác suất tích luỹ* giúp cân bằng tốt giữa độ bao phủ và độ chắc chắn của đầu ra, thay vì cố định kích thước k như phương pháp trước.

Phần trên trình bày 4 quy tắc thường được sử dụng, việc lựa chọn chiến lược nào phụ thuộc vào người dùng muốn ưu tiên tính đa dạng hay độ ổn định của đầu ra.

27.4 Credit Card Fraud Detection

Trong phần này, ta áp dụng mô hình **Softmax Regression** để giải quyết bài toán **Credit Card Fraud Detection** — một bài toán phân loại nhị phân với dữ liệu cực kỳ mất cân bằng. Mục tiêu là phát hiện các giao dịch gian lận hiếm gặp trong hàng loạt giao dịch hợp lệ, đồng thời sử dụng kỹ thuật SMOTE để tái cân bằng dữ liệu huấn luyện. Với phân loại nhị phân, lựa chọn phổ biến là Logistic Regression dùng sigmoid. Tuy nhiên, để làm quen với softmax, trong bài này ta sử dụng Softmax Regression với 2 lớp.



Hình 15: Hình ảnh minh họa cho bài toán Card Fraud Detection.

Bước 1: Import thư viện và tải dữ liệu

```

1 !gdown --id 1npjy1l8BwDe12KQ7MQrb7bo9jzkEWo5s
2 !unzip "card_fraud_detection.zip"
3
4 import pandas as pd
5 import numpy as np
6 import matplotlib.pyplot as plt
7 from imblearn.over_sampling import SMOTE
8
9 from sklearn.model_selection import train_test_split
10 from sklearn.preprocessing import StandardScaler
11
12 # Đọc dữ liệu
13 dataset_path = "creditcard.csv"
14 df = pd.read_csv(dataset_path)
15
16 # Thông tin tổng quan
17 df.info()
18 df.describe()

```

Ở bước này, ta tải và đọc tập dữ liệu creditcard.csv chứa thông tin về các giao dịch thẻ tín dụng. Các lệnh df.info() và df.describe() giúp kiểm tra cấu trúc, kiểu dữ liệu, và thống kê mô tả ban đầu để chuẩn bị cho giai đoạn tiền xử lý.

Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.2777838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.904412	-0.689281	-0.327642	-0.139097	-0.055363	-0.059752	378.66	0
1.0	-0.966272	-0.185228	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647377	-0.221929	0.062723	0.061458	123.50	0
2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.7988278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
...	
786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0.77	0
787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294689	0.584800	...	0.214205	0.924384	0.012463	-0.162226	-0.606628	-0.395255	0.068472	-0.053527	24.79	0
788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031280	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	67.88	0
789.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123305	-0.569159	0.546688	0.108821	0.104533	10.00	0
790.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	217.00	0

Hình 16: Một phần DataFrame của bộ dữ liệu Card Fraud Detection.

Bước 2: Tách đặc trưng và nhãn, thêm thành phần bias

```

1 # Chuyển DataFrame sang mảng NumPy
2 dataset_arr = df.to_numpy()

3

4 # Tách đặc trưng (X) và nhãn (y)
5 X, y = dataset_arr[:, :-1].astype(np.float64), dataset_arr[:, -1].
6                           astype(np.uint8)
7 print(X.shape)

8 # Thêm thành phần bias vào ma trận đặc trưng
9 intercept = np.ones((X.shape[0], 1))
10 X_b = np.concatenate((intercept, X), axis=1)
11 print(X_b.shape)

12

13 # Mã hóa nhãn sang dạng one-hot
14 n_classes = np.unique(y, axis=0).shape[0]
15 n_samples = y.shape[0]

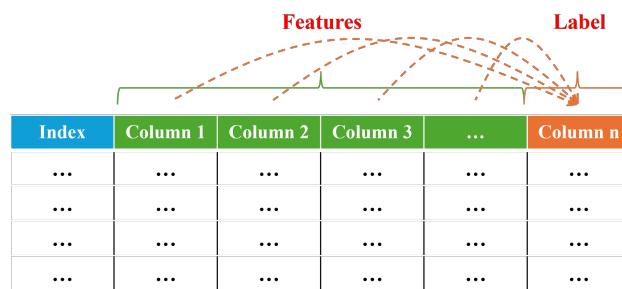
16

17 y_encoded = np.array([np.zeros(n_classes) for _ in range(n_samples)])
18 y_encoded[np.arange(n_samples), y] = 1

```

Ở bước này, ta thực hiện ba thao tác chính:

- Chuyển dữ liệu từ DataFrame sang NumPy array để dễ dàng thao tác dưới dạng ma trận và vector.
- Tách riêng ma trận đặc trưng **X** và nhãn **y**, sau đó thêm thành phần **bias** (cột giá trị 1) vào **X**. Thành phần này giúp mô hình học được cả phần độ lệch thay vì chỉ phụ thuộc vào độ dốc của trọng số.
- Biến đổi nhãn **y** sang dạng *one-hot encoding*, để mỗi mẫu được biểu diễn bằng một vector có giá trị 1 tại vị trí lớp tương ứng — phù hợp với đầu ra xác suất của mô hình **Softmax Regression**.



Hình 17: Mô phỏng việc tách đặc trưng x và nhãn y.

Bước 3: Chia dữ liệu, chuẩn hoá và cân bằng lớp bằng SMOTE

```

1 from imblearn.over_sampling import SMOTE
2
3 val_size = 0.2
4 test_size = 0.125
5 random_state = 2
6 is_shuffle = True
7
8 # Chia tập huấn luyện và validation
9 X_train, X_val, y_train, y_val = train_test_split(
10     X_b, y_encoded,
11     test_size=val_size,
12     random_state=random_state,
13     shuffle=is_shuffle
14 )
15
16 # Chia tiếp một phần từ tập huấn luyện để làm tập kiểm thử (test)
17 X_train, X_test, y_train, y_test = train_test_split(
18     X_train, y_train,
19     test_size=test_size,
20     random_state=random_state,
21     shuffle=is_shuffle
22 )
23
24 # Chuẩn hoá đặc trưng (bỏ qua cột bias)
25 normalizer = StandardScaler()

```

```

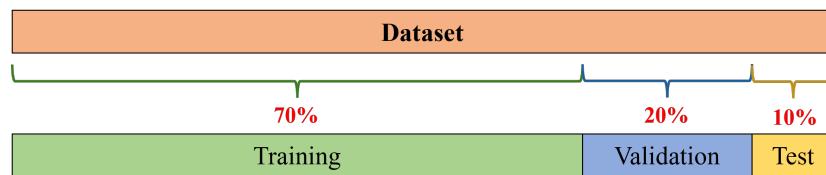
26 X_train[:, 1:] = normalizer.fit_transform(X_train[:, 1:])
27 X_val[:, 1:] = normalizer.transform(X_val[:, 1:])
28 X_test[:, 1:] = normalizer.transform(X_test[:, 1:])
29
30 # Cân bằng dữ liệu huấn luyện bằng SMOTE
31 smote = SMOTE(random_state=random_state, sampling_strategy="minority")
32 X_train_balanced, y_train_balanced = smote.fit_resample(
33     X_train[:, 1:], np.argmax(y_train, axis=1)
34 )
35
36 # Thêm lại thành phần bias sau khi resample
37 intercept_balanced = np.ones((X_train_balanced.shape[0], 1))
38 X_train = np.concatenate((intercept_balanced, X_train_balanced),
39                         axis=1)
40
41 # Mã hóa lại nhãn theo dạng one-hot
42 y_train = np.zeros((y_train_balanced.shape[0], n_classes))
43 y_train[np.arange(y_train_balanced.shape[0]), y_train_balanced] = 1
44
45 # Kiểm tra lại phân phối lớp
46 print(f"Balanced training samples: {X_train.shape[0]}")
47 print(f"Class distribution: {np.bincount(y_train_balanced)}")

```

Ở bước này, ta thực hiện toàn bộ giai đoạn xử lý dữ liệu đầu vào trước khi huấn luyện:

- **Chia dữ liệu:** tách dữ liệu thành ba phần gồm `train`, `validation`, và `test` để huấn luyện, tinh chỉnh và đánh giá mô hình độc lập.
- **Chuẩn hóa đặc trưng:** áp dụng `StandardScaler` cho các đặc trưng đầu vào (ngoại trừ cột `bias`) giúp mô hình hội tụ nhanh hơn và ổn định hơn.
- **Cân bằng dữ liệu bằng SMOTE:** tạo thêm các mẫu nhân tạo cho lớp thiểu số, giảm hiện tượng lệch lớp và giúp mô hình học được ranh giới quyết định chính xác hơn.
- **Tái thêm bias và mã hóa nhãn:** sau khi resample, thêm lại thành phần bias và chuyển nhãn về dạng *one-hot encoding* để đồng nhất với đầu vào/đầu ra của mô hình **Softmax Regression**.

- Cuối cùng, in ra thống kê số mẫu và phân phối lớp để xác nhận dữ liệu đã được xử lý đúng cách.



Hình 18: Hình ảnh minh họa cách chia các tập dữ liệu.

Bước 4: Cài đặt các hàm cơ bản cho Softmax Regression

Code Exercise

```

1 def softmax(z):
2     exp_z = ***Your Code Here***
3
4     return ***Your Code Here***
5
6 def predict(X, theta):
7     z = ***Your Code Here***
8     y_hat = ***Your Code Here***
9
10    return ***Your Code Here***
11
12 def compute_loss(y_hat, y):
13    n = ***Your Code Here***
14
15    return ***Your Code Here***
16
17 def compute_gradient(X, y, y_hat):
18    n = ***Your Code Here***
19
20    return ***Your Code Here***
21
22 def update_theta(theta, gradient, lr):
23    return ***Your Code Here***
24
25 def compute_accuracy(X, y, theta):

```

```

26     y_hat = ***Your Code Here***
27     acc = ***Your Code Here***
28
29     return acc

```

Thực hành điền code vào các vị trí ***Your Code Here*** để hoàn thiện các hàm cốt lõi của mô hình **Softmax Regression**:

- **Hàm softmax(z)**: tính xác suất cho từng lớp bằng công thức

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}},$$

đảm bảo mỗi hàng của đầu ra có tổng bằng 1.

- **Hàm predict(X, theta)**: tính giá trị tuyến tính $z = X\theta$, sau đó áp dụng softmax để thu được ma trận xác suất dự đoán \hat{y} .
- **Hàm compute_loss(y_hat, y)**: triển khai hàm mất mát **cross-entropy** với công thức

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^N y_i \log(\hat{y}_i),$$

trong đó N là số mẫu.

- **Hàm compute_gradient(X, y, y_hat)**: tính gradient theo công thức

$$\nabla_{\theta} = \frac{1}{N} X^T (\hat{y} - y),$$

để cập nhật trọng số.

- **Hàm update_theta(theta, gradient, lr)**: cập nhật tham số mô hình theo quy tắc descent

$$\theta \leftarrow \theta - \eta \nabla_{\theta},$$

với η là tốc độ học (lr).

- **Hàm compute_accuracy(X, y, theta)**: dự đoán đầu ra bằng predict, sau đó so sánh chỉ số lớp lớn nhất (argmax) của \hat{y} và y để tính tỉ lệ chính xác trung bình.

Bước 5: Huấn luyện mô hình Softmax Regression

```
1 lr = 0.01
2 epochs = 30
3 batch_size = 1024
4 n_features = X_train.shape[1]
5
6 # Khởi tạo ngẫu nhiên tham số mô hình
7 np.random.seed(random_state)
8 theta = np.random.uniform(size=(n_features, n_classes))
9
10 # Lưu lịch sử huấn luyện
11 train_accs = []
12 train_losses = []
13 val_accs = []
14 val_losses = []
15
16 for epoch in range(epochs):
17     train_batch_losses = []
18     train_batch_accs = []
19     val_batch_losses = []
20     val_batch_accs = []
21
22     for i in range(0, X_train.shape[0], batch_size):
23         # Lấy mini-batch
24         X_i = X_train[i:i+batch_size]
25         y_i = y_train[i:i+batch_size]
26
27         # Dự đoán và tính mất mát
28         y_hat = predict(X_i, theta)
29         train_loss = compute_loss(y_hat, y_i)
30
31         # Tính gradient và cập nhật tham số
32         gradient = compute_gradient(X_i, y_i, y_hat)
33         theta = update_theta(theta, gradient, lr)
34
35         # Ghi nhận kết quả batch huấn luyện
36         train_batch_losses.append(train_loss)
37         train_acc = compute_accuracy(X_train, y_train, theta)
38         train_batch_accs.append(train_acc)
39
40         # Tính mất mát và độ chính xác trên tập validation
41         y_val_hat = predict(X_val, theta)
```

```

42     val_loss = compute_loss(y_val_hat, y_val)
43     val_batch_losses.append(val_loss)
44
45     val_acc = compute_accuracy(X_val, y_val, theta)
46     val_batch_accs.append(val_acc)
47
48     # Tính trung bình các chỉ số theo batch
49     train_batch_loss = sum(train_batch_losses) / len(
50                               train_batch_losses)
50     val_batch_loss = sum(val_batch_losses) / len(val_batch_losses)
51     train_batch_acc = sum(train_batch_accs) / len(train_batch_accs)
52     val_batch_acc = sum(val_batch_accs) / len(val_batch_accs)
53
54     # Lưu lại lịch sử qua từng epoch
55     train_losses.append(train_batch_loss)
56     val_losses.append(val_batch_loss)
57     train_accs.append(train_batch_acc)
58     val_accs.append(val_batch_acc)
59
60     print(f"\nEPOCH {epoch + 1}:\tTraining loss: {train_batch_loss:.3f}\tValidation loss: {val_batch_loss:.3f}")

```

Đây là giai đoạn huấn luyện chính của mô hình **Softmax Regression**. Các bước được thực hiện như sau:

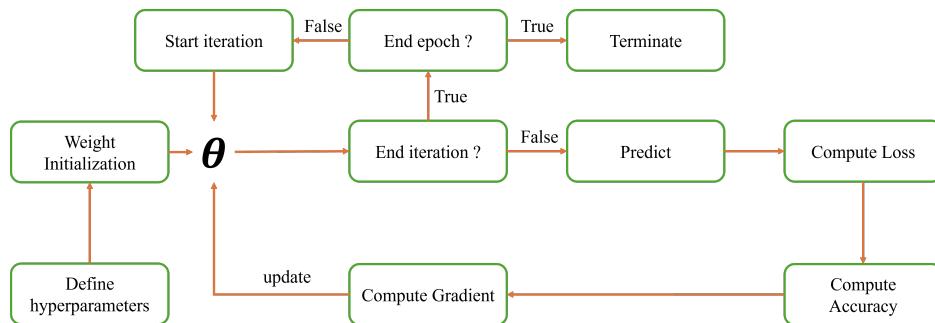
- **Khởi tạo tham số:** sinh ngẫu nhiên ma trận trọng số $\theta \in \mathbb{R}^{(n_features \times n_classes)}$.
- **Huấn luyện theo mini-batch:** chia dữ liệu huấn luyện thành từng lô nhỏ (`batch_size`) để cập nhật tham số tuần tự, giúp giảm dao động gradient và tiết kiệm bộ nhớ.
- **Cập nhật tham số:** với mỗi batch, tính gradient ∇_{θ} và cập nhật theo quy tắc

$$\theta \leftarrow \theta - \eta \nabla_{\theta},$$

trong đó η là tốc độ học (lr).

- **Theo dõi hiệu năng:** sau mỗi epoch, ghi lại *training loss*, *validation loss*, *training accuracy* và *validation accuracy* để đánh giá quá trình hội tụ.

- **In kết quả:** hiển thị thông tin mất mát huấn luyện và validation qua từng epoch để quan sát sự cải thiện của mô hình.



Hình 19: Mô tả quá trình huấn luyện mô hình Softmax Regression sử dụng Gradient Descent.

Sau khi huấn luyện xong, ta trực quan hoá quá trình hội tụ của mô hình qua các chỉ số mất mát và độ chính xác trên cả tập huấn luyện và tập validation.

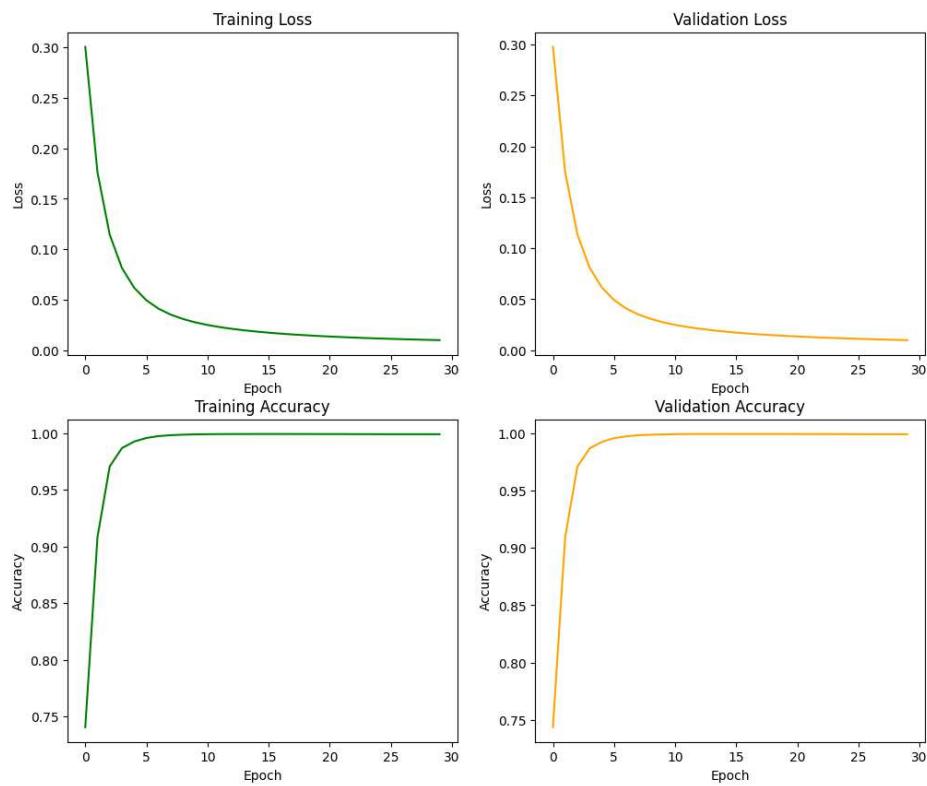
```

1 fig, ax = plt.subplots(2, 2, figsize=(12, 10))
2
3 # Training loss
4 ax[0, 0].plot(train_losses, color="green")
5 ax[0, 0].set(xlabel="Epoch", ylabel="Loss")
6 ax[0, 0].set_title("Training Loss")
7
8 # Validation loss
9 ax[0, 1].plot(val_losses, color="orange")
10 ax[0, 1].set(xlabel="Epoch", ylabel="Loss")
11 ax[0, 1].set_title("Validation Loss")
12
13 # Training accuracy
14 ax[1, 0].plot(train_accs, color="green")
15 ax[1, 0].set(xlabel="Epoch", ylabel="Accuracy")
16 ax[1, 0].set_title("Training Accuracy")
17
18 # Validation accuracy
19 ax[1, 1].plot(val_accs, color="orange")
20 ax[1, 1].set(xlabel="Epoch", ylabel="Accuracy")
21 ax[1, 1].set_title("Validation Accuracy")
  
```

```
22  
23 plt.show()
```

Đoạn mã trên giúp trực quan hóa tiến trình huấn luyện của mô hình:

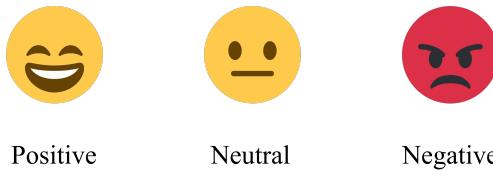
- Hai biểu đồ ở hàng trên thể hiện **giá trị mất mát** (*loss*) trên tập huấn luyện và tập validation qua từng epoch, cho phép quan sát mức độ hội tụ và hiện tượng quá khớp (overfitting) nếu có.
- Hai biểu đồ ở hàng dưới thể hiện **độ chính xác** (*accuracy*) trên cùng hai tập, giúp đánh giá khả năng tổng quát hoá của mô hình.
- Màu xanh biểu diễn kết quả trên tập huấn luyện, trong khi màu cam tương ứng với tập validation.



Hình 20: Hình ảnh trực quan loss, accuracy trên tập train và validation.

27.5 Twitter Sentiment Analysis

Trong phần này, ta áp dụng mô hình **Softmax Regression** cho bài toán **Twitter Sentiment Analysis** — một bài toán phân loại ba lớp nhằm nhận diện cảm xúc của người dùng trên mạng xã hội (*tích cực, tiêu cực và trung lập*). Bài toán này giúp mô hình học cách biểu diễn và phân loại văn bản dựa trên các đặc trưng ngôn ngữ được trích xuất từ dữ liệu tweet.



Hình 21: Hình ảnh minh họa ba nhãn của tập dữ liệu.

Bước 1: Import thư viện và tải dữ liệu

```

1 !gdown --id 1aQ80lUljwEm7RLZz8Qf4xzim0nDewHfU
2 !unzip "../content/twitter_sentiment_analysis_3cls_dataset.zip"
3
4 import numpy as np
5 import pandas as pd
6 import matplotlib.pyplot as plt
7 import re
8 import torch
9 import torch.nn as nn
10 import torch.optim as optim
11 import nltk
12 nltk.download("stopwords")
13
14 from sklearn.model_selection import train_test_split
15 from sklearn.feature_extraction.text import TfidfVectorizer
16 from nltk.corpus import stopwords
17 from nltk.stem import SnowballStemmer
18
19 # Đọc dữ liệu gốc
20 dataset_path = "../content/Twitter_Data.csv"
21 df = pd.read_csv(dataset_path)

```

```
22 df
23
24 # Kiểm tra thông tin tổng quan
25 df.info()
26 df.describe()
27
28 # Kiểm tra và loại bỏ các dòng bị thiếu dữ liệu
29 null_rows = df.isnull().any(axis=1)
30 df[null_rows]
31 df = df.dropna()
```

Ở bước này, ta tiến hành các thao tác chuẩn bị dữ liệu ban đầu:

- Tải và giải nén tập dữ liệu Twitter_Data.csv chứa các tweet và nhãn cảm xúc tương ứng.
- Import đầy đủ các thư viện phục vụ cho tiền xử lý văn bản, bao gồm nltk (xử lý ngôn ngữ tự nhiên), TfidfVectorizer (biểu diễn văn bản dạng vector), và SnowballStemmer (chuẩn hóa từ gốc).
- Kiểm tra thông tin dữ liệu bằng df.info() và df.describe() để nắm cấu trúc và kiểu dữ liệu.
- Loại bỏ các dòng bị thiếu giá trị (NaN) nhằm đảm bảo dữ liệu sạch trước khi tiền xử lý.

	clean_text	category
0	when modi promised "minimum government maximum...	-1.0
1	talk all the nonsense and continue all the dra...	0.0
2	what did just say vote for modi welcome bjp t...	1.0
3	asking his supporters prefix chowkidar their n...	1.0
4	answer who among these the most powerful world...	1.0
...
162975	why these 456 crores paid neerav modi not reco...	-1.0
162976	dear rss terrorist payal gawar what about modi...	-1.0
162977	did you cover her interaction forum where she ...	0.0
162978	there big project came into india modi dream p...	0.0
162979	have you ever listen about like gurukul where ...	1.0

162980 rows × 2 columns

Hình 22: DataFrame của bộ dữ liệu Twitter Sentiment Analysis.

Bước 2: Tiền xử lý và biểu diễn dữ liệu văn bản

```

1 def text_normalize(text):
2     # Chuyển toàn bộ về chữ thường
3     text = text.lower()
4
5     # Xoá tiền tố "RT" của các tweet được retweet
6     text = re.sub(r"^\rt[\s]+", "", text)
7
8     # Loại bỏ đường dẫn (URLs)
9     text = re.sub(r"https?:\/\/.*[\r\n]*", "", text)
10
11    # Loại bỏ dấu câu và ký tự đặc biệt
12    text = re.sub(r"[\w\s]", "", text)
13
14    # Loại bỏ stopwords
15    stop_words = set(stopwords.words("english"))
16    words = text.split()
17    words = [word for word in words if word not in stop_words]
18    text = " ".join(words)
19
20    # Stemming (đưa từ về gốc)
21    stemmer = SnowballStemmer("english")
22    words = text.split()

```

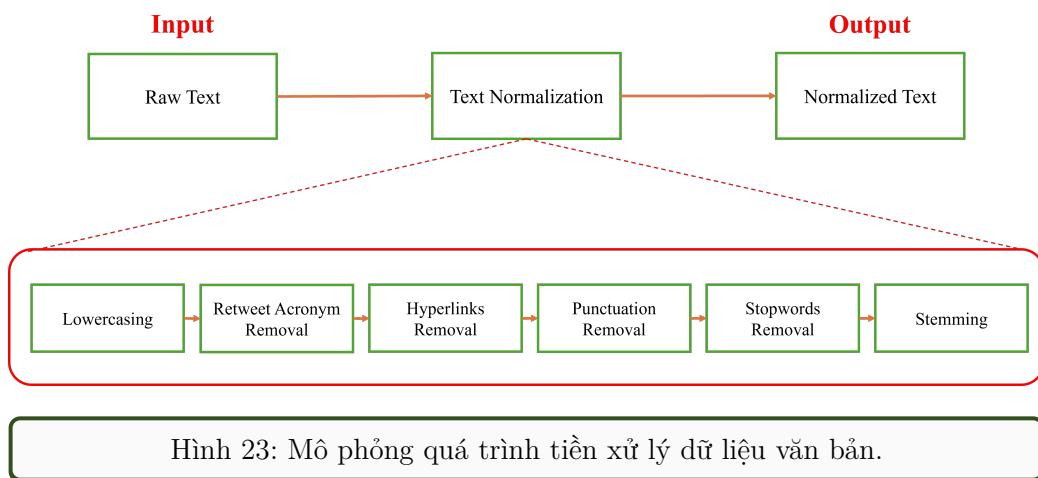
```

23     words = [stemmer.stem(word) for word in words]
24     text = " ".join(words)
25
26     return text
27
28 # Ví dụ minh họa
29 text = """We love this! Would you go?
30 #talk #makememories #unplug
31 #relax #iphone #smartphone #wifi #connect...
32 http://fb.me/6N3LsUpCu
33 """
34 text = text_normalize(text)
35 text
36
37 # Áp dụng chuẩn hoá cho toàn bộ tập dữ liệu
38 df["clean_text"] = df["clean_text"].apply(lambda x: text_normalize(x))
39
40 # Biểu diễn văn bản bằng TF-IDF
41 vectorizer = TfidfVectorizer(max_features=2000)
42 X = vectorizer.fit_transform(df["clean_text"]).toarray()

```

Ở bước này, ta tiến hành tiền xử lý văn bản và biến đổi chúng thành dạng vector số hoá:

- Xây dựng hàm `text_normalize()` gồm các bước: chuyển chữ thường, xoá tiền tố RT, loại bỏ đường dẫn và dấu câu, xoá các từ dừng (*stopwords*) và thực hiện *stemming* để đưa từ về gốc.
- Áp dụng hàm này cho toàn bộ cột `clean_text` trong dữ liệu, nhằm tạo tập văn bản sạch phục vụ cho huấn luyện.
- Sử dụng **TF-IDF Vectorizer** để chuyển đổi văn bản thành ma trận đặc trưng số, trong đó mỗi cột đại diện cho một từ quan trọng (với `max_features=2000`).



Bước 3: Chuẩn bị dữ liệu và xây dựng mô hình Softmax Regression

Code Exercise

```

1 n_classes = ***Your Code Here***
2 n_samples = ***Your Code Here***
3
4 y = ***Your Code Here***
5 y = y.astype(np.uint8)
6 y_encoded = np.array(
    ***Your Code Here***
)
9 y_encoded[np.arange(n_samples), y] = 1
10
11 X = torch.tensor(X, dtype=torch.float32)
12 y = torch.tensor(y_encoded, dtype=torch.float32)
13
14 val_size = 0.2
15 test_size = 0.125
16 random_state = 2
17 is_shuffle = True
18
19 X_train, X_val, y_train, y_val = train_test_split(
    ***Your Code Here***,
    test_size=***Your Code Here***,
)

```

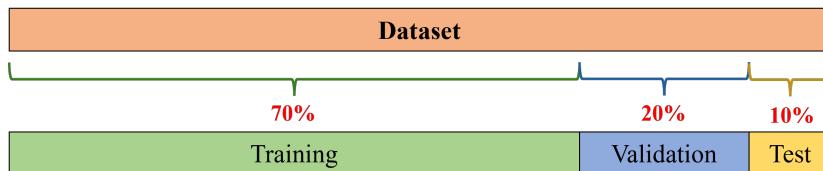
```
22     random_state=random_state,
23     shuffle=is_shuffle
24 )
25
26 X_train, X_test, y_train, y_test = train_test_split(
27     ***Your Code Here***,
28     test_size=***Your Code Here***,
29     random_state=random_state,
30     shuffle=is_shuffle
31 )
32
33 class SoftmaxRegression(nn.Module):
34     def __init__(self, input_dim, output_dim):
35         super(SoftmaxRegression, self).__init__()
36         self.linear = nn.Linear(
37             ***Your Code Here***,
38             ***Your Code Here***,
39             bias=True
40         )
41
42     def forward(self, x):
43         return ***Your Code Here***
44
45     def compute_accuracy(y_hat, y_true):
46         _, y_hat = torch.max(y_hat, dim=1)
47         _, y_true = ***Your Code Here***
48
49         correct = ***Your Code Here***
50         accuracy = ***Your Code Here***
51
52     return accuracy
```

Thực hành điền code vào các vị trí *****Your Code Here***** để hoàn thiện bước chuẩn bị dữ liệu và định nghĩa mô hình **Softmax Regression**:

- Xác định tổng số lớp cảm xúc và tổng số mẫu dữ liệu để thiết lập cấu trúc đầu ra phù hợp cho mô hình.
- Chuyển đổi nhãn sang dạng số nguyên và mã hoá lại theo dạng *one-hot* để mô hình có thể xử lý nhiều lớp cùng lúc.
- Chia dữ liệu thành các tập huấn luyện, kiểm định và kiểm thử, giúp

đánh giá mô hình một cách độc lập và khách quan.

- Trong lớp `SoftmaxRegression`, định nghĩa tầng tuyến tính có đầu vào là vector đặc trưng và đầu ra là vector logit biểu diễn điểm số cho từng lớp.
- Trong hàm `forward`, viết phần truyền thẳng đầu vào qua tầng tuyến tính để thu được giá trị dự đoán thô (logit) của từng lớp.
- Trong hàm `compute_accuracy`, lấy chỉ số lớp có giá trị lớn nhất ở cả đầu ra dự đoán và nhãn thật, so sánh chúng để đếm số lượng dự đoán đúng, sau đó tính tỉ lệ chính xác bằng số đúng chia cho tổng số mẫu.



Hình 24: Hình ảnh minh họa cách chia các tập dữ liệu.

Bước 4: Huấn luyện mô hình Softmax Regression

Code Exercise

```

1 lr = 0.1
2 epochs = 500
3 torch.manual_seed(random_state)
4 if torch.cuda.is_available():
5     torch.cuda.manual_seed(random_state)
6
7 input_dim = X_train.shape[1]
8 output_dim = y_train.shape[1]
9
10 model = SoftmaxRegression(
11     ***Your Code Here***
12 )
13 criterion = ***Your Code Here***
14 optimizer = ***Your Code Here***
15

```

```
16 train_losses = []
17 train_accs = []
18 val_losses = []
19 val_accs = []
20
21 for epoch in range(epochs):
22     model.***Your Code Here***  

23
24     # Zero the gradients
25     optimizer.zero_grad()
26
27     # Forward pass
28     y_hat = model(X_train)
29
30     # Compute loss
31     train_loss = ***Your Code Here***  

32
33     train_losses.append(train_loss.item())
34
35     train_acc = compute_accuracy(y_hat, y_train)
36     train_accs.append(train_acc)
37
38     # Backward pass and optimization
39     train_loss.***Your Code Here***  

40     optimizer.***Your Code Here***  

41
42     model.eval()
43     # Forward pass for validation data
44     with torch.***Your Code Here***:
45         y_val_hat = model(X_val)
46
47         # Compute validation loss
48         val_loss = criterion(y_val_hat, y_val)
49         val_losses.append(val_loss.item())
50
51         val_acc = compute_accuracy(y_val_hat, y_val)
52         val_accs.append(val_acc)
53
54     print(f"\nEPOCH {epoch + 1}: tTraining loss: {train_loss:.3f}\n"
55          "Validation loss: {val_loss:.3f}")
```

Điền vào các vị trí ***Your Code Here*** để hoàn thiện vòng lặp huấn luyện của mô hình **Softmax Regression**:

- Khởi tạo mô hình với kích thước đầu vào và đầu ra tương ứng với số đặc trưng và số lớp cần dự đoán.
- Định nghĩa hàm mất mát dùng để đo sai khác giữa dự đoán và nhãn thật, cùng bộ tối ưu hoá dùng để cập nhật trọng số theo gradient descent.
- Trong mỗi epoch huấn luyện:
 - Kích hoạt chế độ huấn luyện của mô hình, xoá gradient cũ và thực hiện lan truyền xuôi để tính đầu ra dự đoán.
 - Tính giá trị mất mát trên tập huấn luyện, sau đó lan truyền ngược để tính gradient.
 - Cập nhật tham số mô hình thông qua bước tối ưu.
- Sau khi huấn luyện xong một epoch, chuyển mô hình sang chế độ đánh giá, thực hiện dự đoán trên tập kiểm định và tính toán các chỉ số mất mát và độ chính xác để theo dõi quá trình học.
- Lưu lại lịch sử thay đổi của *loss* và *accuracy* qua từng epoch, đồng thời in ra kết quả theo thời gian để quan sát mức độ hội tụ của mô hình.

Sau khi huấn luyện xong, ta tiến hành trực quan hóa quá trình hội tụ của mô hình để quan sát sự thay đổi của các chỉ số mất mát và độ chính xác qua từng epoch.

```

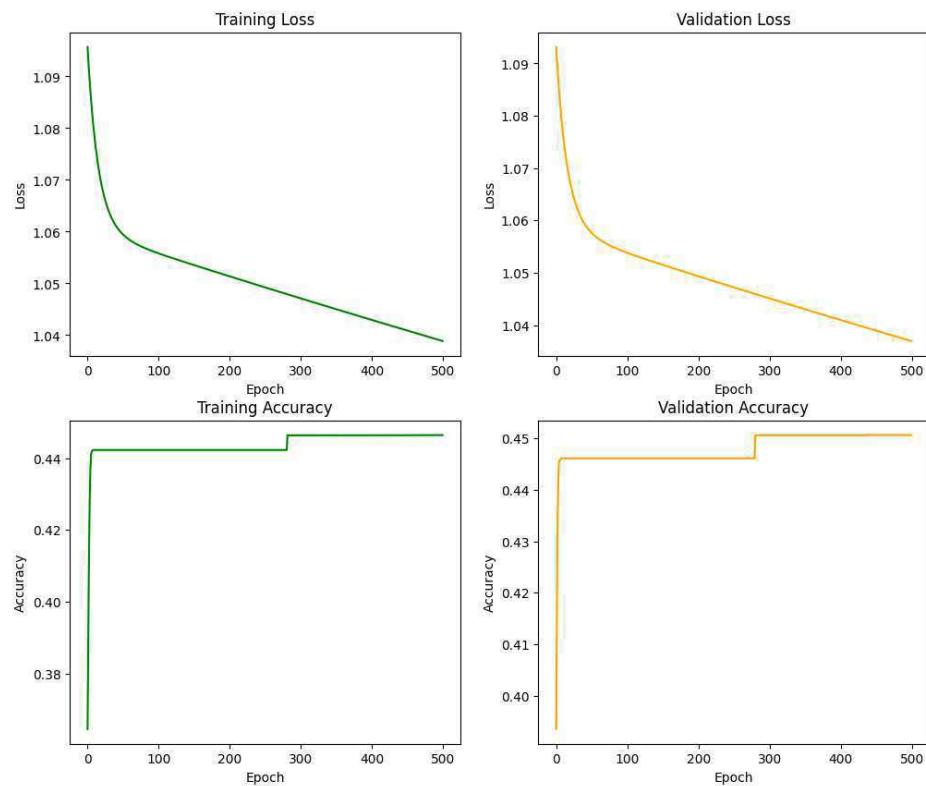
1 fig, ax = plt.subplots(2, 2, figsize=(12, 10))
2
3 # Training loss
4 ax[0, 0].plot(train_losses, color="green")
5 ax[0, 0].set(xlabel="Epoch", ylabel="Loss")
6 ax[0, 0].set_title("Training Loss")
7
8 # Validation loss
9 ax[0, 1].plot(val_losses, color="orange")
10 ax[0, 1].set(xlabel="Epoch", ylabel="Loss")
11 ax[0, 1].set_title("Validation Loss")
12
13 # Training accuracy
14 ax[1, 0].plot(train_accs, color="green")
15 ax[1, 0].set(xlabel="Epoch", ylabel="Accuracy")

```

```
16 ax[1, 0].set_title("Training Accuracy")
17
18 # Validation accuracy
19 ax[1, 1].plot(val_accs, color="orange")
20 ax[1, 1].set(xlabel="Epoch", ylabel="Accuracy")
21 ax[1, 1].set_title("Validation Accuracy")
22
23 plt.show()
```

Đoạn mã trên trực quan hóa toàn bộ quá trình huấn luyện của mô hình:

- Hai biểu đồ ở hàng trên thể hiện sự thay đổi của **loss** trên tập huấn luyện và tập kiểm định, giúp quan sát xu hướng hội tụ và phát hiện hiện tượng quá khớp nếu xảy ra.
- Hai biểu đồ ở hàng dưới mô tả **độ chính xác** trên hai tập dữ liệu tương ứng, phản ánh mức độ học và khả năng tổng quát của mô hình.
- Màu xanh đại diện cho các chỉ số trên tập huấn luyện, trong khi màu cam biểu thị cho tập kiểm định.
- Việc theo dõi song song hai cặp biểu đồ này giúp đánh giá sự ổn định và khả năng hội tụ của mô hình trong suốt quá trình huấn luyện.



Hình 25: Hình ảnh trực quan loss, accuracy trên tập train và validation.

27.6 Câu hỏi trắc nghiệm

1. (Lý thuyết) Mục đích chính của hàm **softmax** trong phân loại đa lớp là gì?
 - (a) Tăng biên độ của logit để mô hình học nhanh hơn.
 - (b) Làm giảm số chiều của dữ liệu đầu vào.
 - (c) Giảm độ chênh lệch giữa các lớp bằng cách chuẩn hoá giá trị tuyệt đối.
 - (d) Biến các logit thành phân bố xác suất chuẩn hoá trên các lớp.
2. (Lý thuyết) Khi thêm **temperature** τ vào softmax: $p_i(\tau) = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)}$, tác động của τ lên phân bố xác suất là gì?
 - (a) Khi τ nhỏ, phân bố phẳng hơn; khi τ lớn, phân bố tập trung hơn.
 - (b) Thay đổi τ không ảnh hưởng đến phân bố xác suất.
 - (c) Khi τ nhỏ, phân bố tập trung hơn (entropy giảm); khi τ lớn, phân bố phẳng hơn (entropy tăng).
 - (d) τ chỉ ảnh hưởng đến nhãn có logit nhỏ nhất.
3. (Lý thuyết) Phát biểu nào dưới đây là đúng về **tính chất chuẩn hoá** của softmax?
 - (a) Các xác suất có thể âm nếu logits chênh lệch nhiều.
 - (b) Tổng tất cả xác suất đều ra luôn bằng 1.
 - (c) Tổng xác suất có thể lớn hơn 1 nếu logits âm.
 - (d) Không có ràng buộc tổng xác suất trong softmax.
4. (Tính toán) Cho $\mathbf{z} = [2.0, 1.0, 0.1]$.
Tính softmax(\mathbf{z}) (làm tròn đến 3 chữ số thập phân).
 - (a) [0.721, 0.178, 0.101]
 - (b) [0.610, 0.300, 0.090]
 - (c) [0.659, 0.242, 0.099]
 - (d) [0.500, 0.300, 0.200]

5. (Tính toán) Với cùng $\mathbf{z} = [2.0, 1.0, 0.1]$ và $\tau = 2$, tính $p(\tau) = \text{softmax}(\mathbf{z}/2)$. Giá trị $p_1(\tau)$ xấp xỉ bằng (làm tròn 3 chữ số thập phân):
- (a) 0.333
 - (b) 0.242
 - (c) 0.659
 - (d) 0.502
6. (Tính toán) Cho $\mathbf{z} = [2, 0]$.
Tính $\text{softmax}(\mathbf{z})$ và $\text{softmax}(\mathbf{z}/2)$, sau đó so sánh độ “phẳng” của phân bố.
- (a) $\tau = 1$ cho phân bố phẳng hơn; $\tau = 2$ cho phân bố tập trung hơn.
 - (b) $\tau = 1$ cho phân bố tập trung hơn; $\tau = 2$ cho phân bố phẳng hơn.
 - (c) Hai phân bố giống hệt nhau.
 - (d) $\tau = 2$ làm thay đổi tổng xác suất.
7. (Tính toán) Cho ba lớp với logits $\mathbf{z} = [3, 1, 0]$.
Tính $\exp(z_i)$ và mẫu số $\sum_j \exp(z_j)$ (làm tròn 3 chữ số thập phân).
- (a) $\exp(z) = [3.000, 1.000, 0.333]$, $S = 4.333$
 - (b) $\exp(z) = [9.000, 3.000, 1.000]$, $S = 13.000$
 - (c) $\exp(z) = [2.000, 1.000, 0.500]$, $S = 3.500$
 - (d) $\exp(z) = [20.086, 2.718, 1.000]$, $S = 23.804$
8. (Tính toán) Với logits $\mathbf{z} = [3, 1, 0]$ và kết quả trên, tính xác suất lớp đầu tiên p_1 .
- (a) $p_1 = 0.650$
 - (b) $p_1 = 0.844$
 - (c) $p_1 = 0.500$
 - (d) $p_1 = 0.707$
9. (Tính toán) Cho $\mathbf{z} = [1.0, 1.0, 1.0]$.
Kết quả $\text{softmax}(\mathbf{z})$ là:
- (a) $[0.6, 0.2, 0.2]$

(b) [0.7, 0.2, 0.1]

(c) [1/3, 1/3, 1/3]

(d) [0.5, 0.3, 0.2]

10. (Tính toán) Cho $\mathbf{z} = [2, 1, 0]$ và $\tau = 10$.

Giá trị của phân bố $p(\tau)$ gần nhất với (làm tròn 3 chữ số thập phân):

(a) [0.660, 0.240, 0.100]

(b) [0.367, 0.332, 0.301]

(c) [0.990, 0.000, 0.010]

(d) [0.800, 0.150, 0.050]

1. **Hint:** Các file code gợi ý và dữ liệu nếu có được lưu trong thư mục có thể được tải [tại đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại [đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 giờ.

AIO_QAs-Verified

■ Nhóm Riêng tư · 1,4K thành viên



Hình 26: Hình ảnh group facebook AIO Q&A

4. Rubric:

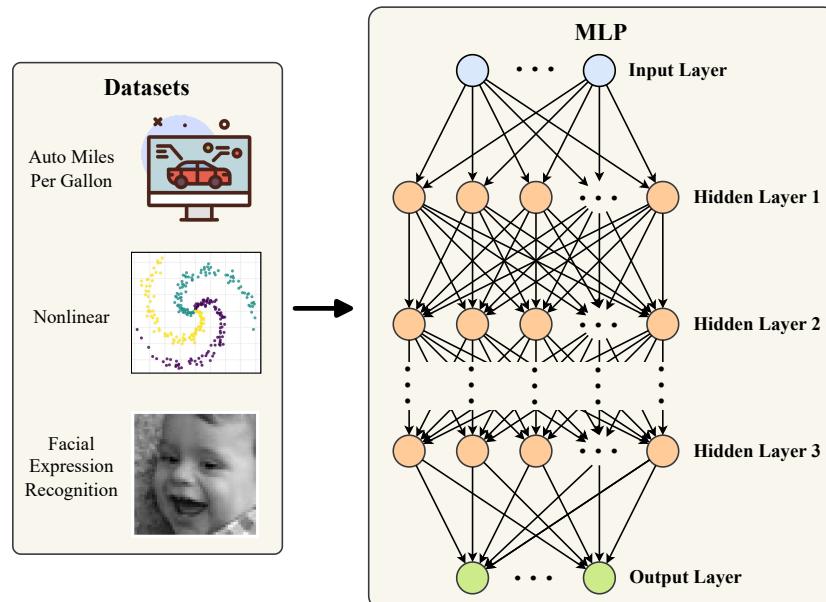
Softmax Regression Project - Rubric		
Phần	Kiến Thức	Đánh Giá
II.A	<ul style="list-style-type: none"> - Kiến thức về việc cài đặt chương trình huấn luyện mô hình Softmax Regression từ đầu. - Kiến thức về các hàm thành phần trong Softmax Regression. - Kiến thức về thuật toán Gradient Descent. - Kiến thức về một số phương pháp tiền xử lý dữ liệu bảng và dữ liệu văn bản. 	<ul style="list-style-type: none"> - Nắm được cách cài đặt mô hình Softmax Regression từ đầu. - Nắm được các hàm thành phần liên quan đến Softmax Regression. - Hiểu được cách vận hành của thuật toán Gradient Descent.
II.B	<ul style="list-style-type: none"> - Các kiến thức và nội dung cơ bản về Softmax Regression. 	<ul style="list-style-type: none"> - Nắm được kiến thức cơ bản về Softmax Regression.

Chương 28

Multi-layer Perceptron

28.1 Giới thiệu

Multi-layer Perceptron (MLP) là kiến trúc neural network (tạm dịch: mạng nơ-ron) cơ bản và quan trọng trong lĩnh vực deep learning (tạm dịch: học sâu). MLP được cấu tạo từ ba thành phần chính, bao gồm input layer (tạm dịch: lớp đầu vào) nhận dữ liệu đặc trưng, hidden layers (tạm dịch: các lớp ẩn) thực hiện biến đổi phi tuyến thông qua activation functions (tạm dịch: các hàm kích hoạt), và output layer (tạm dịch: lớp đầu ra) tạo ra kết quả dự đoán cuối cùng. Khả năng học các mối quan hệ phi tuyến phức tạp giữa đầu vào và đầu ra khiến MLP trở thành công cụ mạnh mẽ cho đa dạng bài toán như classification (tạm dịch: phân loại), regression (tạm dịch: hồi quy) và pattern recognition (tạm dịch: nhận dạng mẫu).



Hình 28.1: Kiến trúc tổng quát của Multi-layer Perceptron (MLP) với các lớp fully connected. Mỗi neuron ở lớp hiện tại kết nối với tất cả neuron ở lớp trước đó.

Trong bài tập này, chúng ta sẽ thực hành cách triển khai và huấn luyện mạng MLP bằng thư viện PyTorch thông qua ba bài tập thực hành với độ phức tạp tăng dần. Nhiệm vụ của các bạn là sẽ hoàn thiện các file code Hint trong phần ?? để triển khai thành công chương trình. Nội dung chính của các câu bài tập như sau:

Tổng quan các bài tập lập trình			
Bài	Mục tiêu	Biểu diễn/Đầu vào	Hàm mục tiêu/Đánh giá
1	Xây dựng MLP với PyTorch cho bài toán hồi quy Auto MPG, huấn luyện và đánh giá.	Dữ liệu CSV với 9 đặc trưng xe hơi đã chuẩn hóa, $X \in \mathbb{R}^{n \times 9}$, $y \in \mathbb{R}^n$ (MPG liên tục).	Mean Squared Error (MSE), theo dõi loss và R^2 score, đánh giá trên validation và test.
2	Xây dựng MLP cho phân loại dữ liệu phi tuyến 3 lớp trong không gian 2D.	Dữ liệu .npy với 2 đặc trưng đã chuẩn hóa, $X \in \mathbb{R}^{300 \times 2}$, $y \in \{0, 1, 2\}^{300}$.	Cross Entropy Loss, accuracy đa lớp, trực quan hóa đường hội tụ train/validation loss và accuracy.
3	Xây dựng MLP sâu với PyTorch cho phân loại cảm xúc khuôn mặt FER-2013.	Ảnh grayscale 128×128 flatten $\rightarrow 16384$ chiều, chuẩn hóa về $[-1, 1]$; $X \in \mathbb{R}^{n \times 16384}$, $y \in \{0, \dots, 6\}^n$ (7 cảm xúc).	Cross Entropy Loss, accuracy đa lớp trên validation/test, kiến trúc pyramid 3 lớp ẩn ($256 \rightarrow 128 \rightarrow 64$), trực quan hóa hội tụ.

Mặc dù có sự khác biệt về bối cảnh ứng dụng (hồi quy, phân loại phi tuyến, phân loại ảnh), việc cài đặt và huấn luyện mô hình MLP với PyTorch đều tuân theo quy trình chung như sau:

- Nhập thư viện và thiết lập môi trường:** Import PyTorch, NumPy, pandas, scikit-learn, matplotlib; cài đặt seed cố định cho reproducibility; thiết lập device (GPU/CPU).

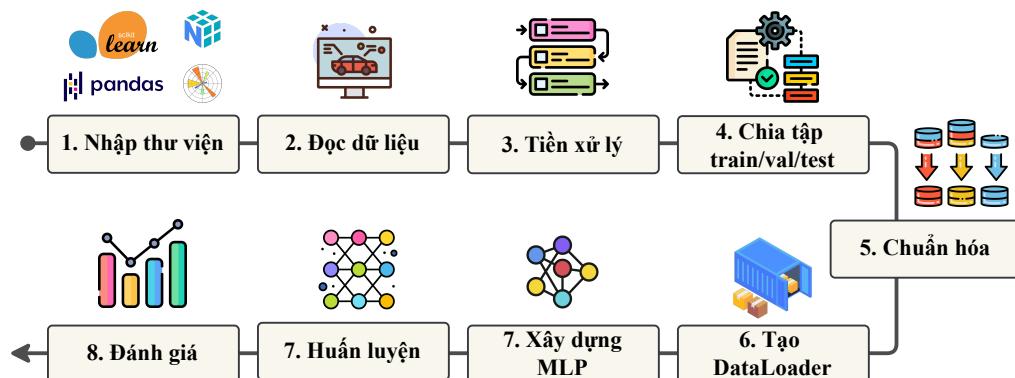
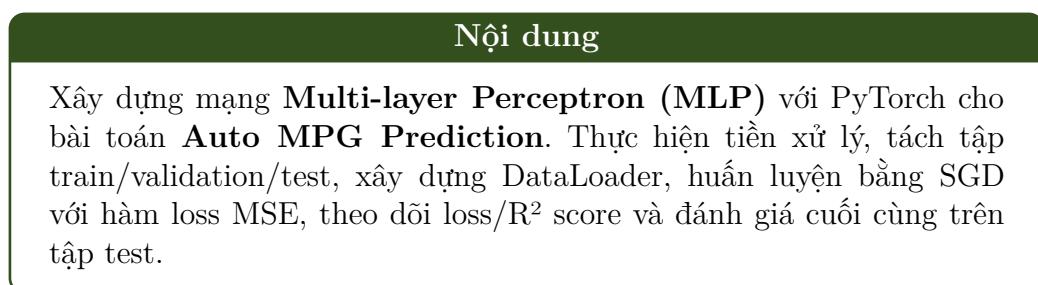
2. **Chuẩn bị dữ liệu:** Tải dữ liệu (CSV/NPY/ZIP), tách X và y, chia train/validation/test theo tỉ lệ 7:2:1, chuẩn hóa đặc trưng bằng StandardScaler (fit trên train, transform trên validation/test), chuyển đổi sang PyTorch tensor với dtype phù hợp.
3. **Xây dựng DataLoader:** Tạo class CustomDataset kế thừa `torch.utils.data.Dataset`, triển khai `__len__` và `__getitem__`, khởi tạo DataLoader với batch size phù hợp (train có `shuffle=True`, validation/test có `shuffle=False`).
4. **Định nghĩa kiến trúc MLP:** Xây dựng class kế thừa `nn.Module`, định nghĩa các lớp Linear và activation (ReLU) trong `__init__`, triển khai forward pass với số lớp ẩn và hidden dimensions phù hợp với độ phức tạp bài toán.
5. **Khai báo loss và optimizer:** Chọn hàm loss phù hợp (`nn.MSELoss` cho regression, và classification là `nn.CrossEntropyLoss`), khởi tạo optimizer (`torch.optim.SGD`) với learning rate thích hợp.
6. **Huấn luyện:** Chuyển sang `model.train()`, duyệt qua `train_loader`, thực hiện forward pass, tính loss, backward propagation (`loss.backward()`), cập nhật weights (`optimizer.step()`); chuyển sang `model.eval()` với `torch.no_grad()` để đánh giá validation, ghi lại loss/metrics cho cả train và validation.
7. **Đánh giá và trực quan:** Tính metrics cuối cùng (R^2 score cho regression, accuracy cho classification) trên test set, vẽ biểu đồ loss/metrics theo epoch để phân tích hội tụ và phát hiện overfitting.

Với quy trình chung đã trình bày, phần tiếp theo sẽ hướng dẫn chi tiết cách triển khai ba bài tập thực hành với PyTorch.

28.2 Thực hành

28.2.1 Bài tập 1: Dự đoán hiệu suất xe hơi

Phát biểu bài toán



Hình 28.2: Quy trình xây dựng MLP cho bài tập dự đoán hiệu suất xe hơi.

Trong bài tập này, ta làm quen với việc xây dựng mạng MLP sử dụng PyTorch thông qua bài toán hồi quy (regression) dự đoán mức tiêu thụ nhiên liệu (MPG - Miles Per Gallon) của xe hơi dựa trên các đặc trưng kỹ thuật. Mô hình MLP gồm 2 lớp ẩn với hàm kích hoạt ReLU, được tối ưu hóa bằng thuật toán Stochastic Gradient Descent với hàm loss Mean Squared Error. Quá trình huấn luyện được giám sát qua hai chỉ số: MSE loss và R² score (Coefficient of determination).

Mục tiêu của bài tập

- Thành thạo việc sử dụng PyTorch để xây dựng mạng MLP với các thành phần: `nn.Module`, `nn.Linear`, `F.relu`.
- Thiết lập pipeline hoàn chỉnh: đọc dữ liệu CSV, tách train/validation/test theo tỉ lệ 7:2:1, chuẩn hóa đặc trưng bằng StandardScaler.
- Xây dựng PyTorch Dataset và DataLoader để quản lý dữ liệu hiệu quả với mini-batch.
- Cài đặt seed cố định cho tính tái lập và tự động chọn thiết bị GPU/CPU.
- Huấn luyện mô hình với vòng lặp epoch, theo dõi đường hối tụ loss và R^2 score trên cả train và validation.
- Đánh giá khả năng tổng quát hóa của mô hình thông qua R^2 score trên tập test.

Yêu cầu

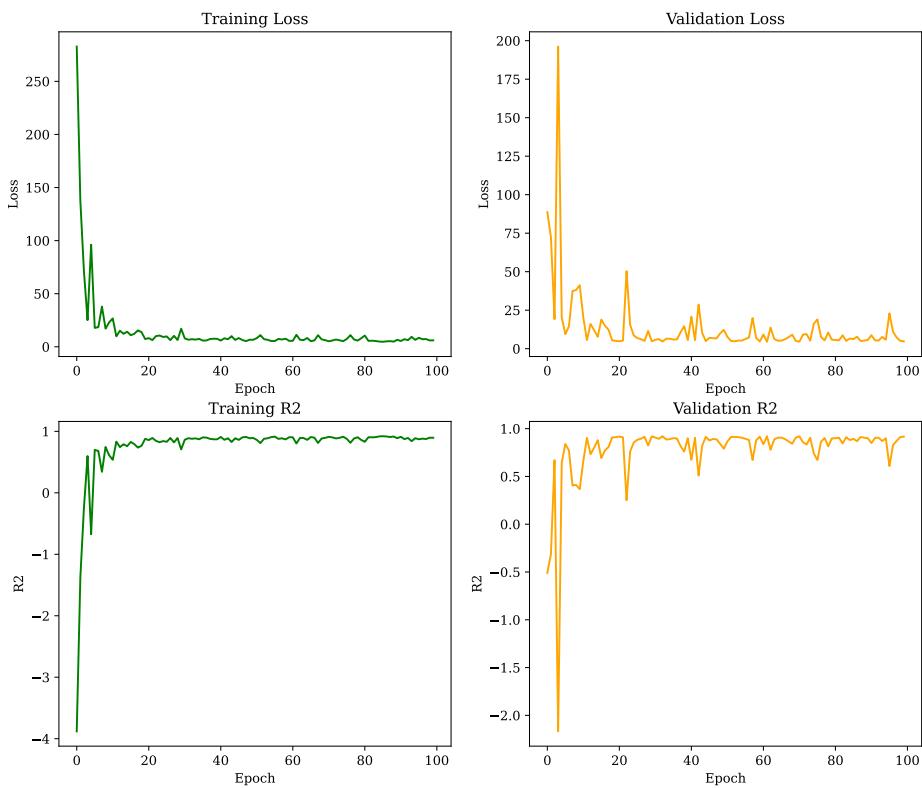
Dựa vào hướng dẫn chi tiết dưới đây, hãy hoàn thiện pipeline huấn luyện mô hình MLP cho bài toán Auto MPG:

- **Chuẩn bị môi trường:** Tải bộ dữ liệu Auto MPG data.csv, import các thư viện cần thiết (PyTorch, NumPy, pandas, scikit-learn), cài đặt seed cố định cho NumPy và PyTorch, thiết lập device (GPU/CPU).
- **Tiền xử lý dữ liệu:** Đọc file CSV, tách đặc trưng X và nhãn y (MPG), chia các tập train/validation/test theo tỉ lệ 7:2:1, chuẩn hóa đặc trưng bằng StandardScaler (fit trên train, transform trên cả ba tập), chuyển đổi sang PyTorch tensor.
- **Xây dựng DataLoader:** Tạo class CustomDataset kế thừa `torch.utils.data.Dataset` với các phương thức `__len__` và `__getitem__`, khởi tạo DataLoader cho train (shuffle=True) và validation (shuffle=False).
- **Kiến trúc mô hình:** Xây dựng class MLP kế thừa `nn.Module` với 2 lớp ẩn, mỗi lớp sau bởi ReLU activation, lớp output không có activation (regression), sử dụng `squeeze(1)` để đảm bảo output shape phù hợp.

- Loss và Optimizer:** Khai báo `nn.MSELoss()` và `torch.optim.SGD()`, cài đặt learning rate phù hợp.
- Hàm đánh giá:** Cài đặt hàm `r_squared` để tính R^2 score: $R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$ với $SS_{res} = \sum(y_{true} - y_{pred})^2$ và $SS_{tot} = \sum(y_{true} - \bar{y})^2$.
- Vòng lặp huấn luyện:** Chuyển đổi giữa `model.train()` và `model.eval()`, thực hiện forward-backward-update cho mỗi batch, tính và lưu train/validation loss và R^2 score mỗi epoch, sử dụng `torch.no_grad()` khi đánh giá.
- Đánh giá cuối cùng:** Tính R^2 score trên tập test sau khi huấn luyện hoàn tất.

Bảng cấu hình siêu tham số – Bài 1 (Auto MPG)

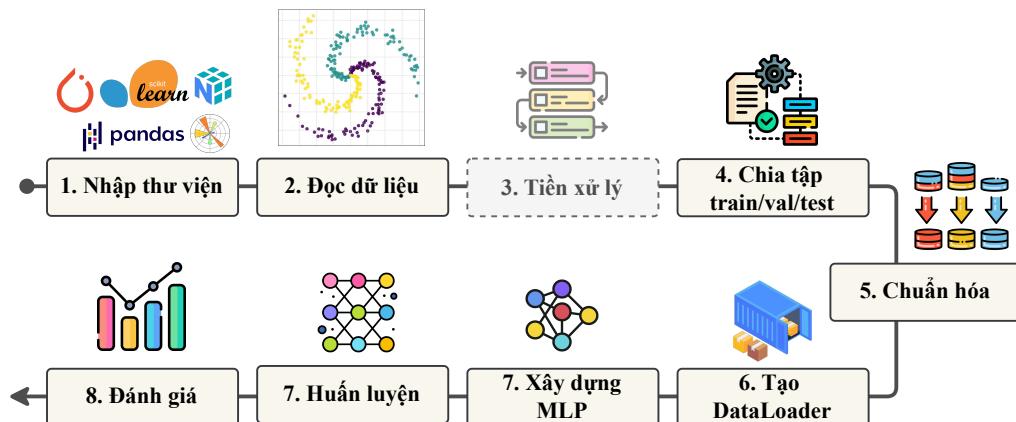
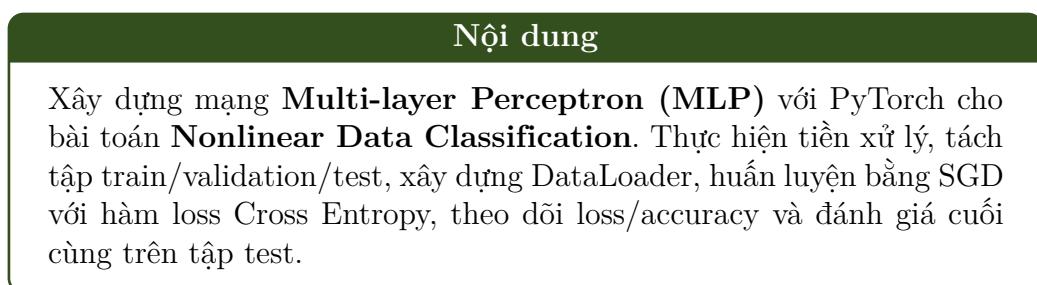
Tham số	Giá trị
Learning rate (lr)	0.01
Số epoch (epochs)	100
Batch size (batch_size)	32
Hidden dimensions (hidden_dims)	64
Val/Test split	0.2 / 0.125
Seed cố định (random_state)	59
Loss function	MSELoss
Optimizer	SGD
Evaluation metric	R^2 score



Hình 28.3: Trực quan hóa kết quả huấn luyện/training/validation loss và accuracy qua các epoch, cho thấy sự hội tụ ổn định của mô hình.

28.2.2 Bài tập 2: Phân loại với dữ liệu phi tuyến

Phát biểu bài toán

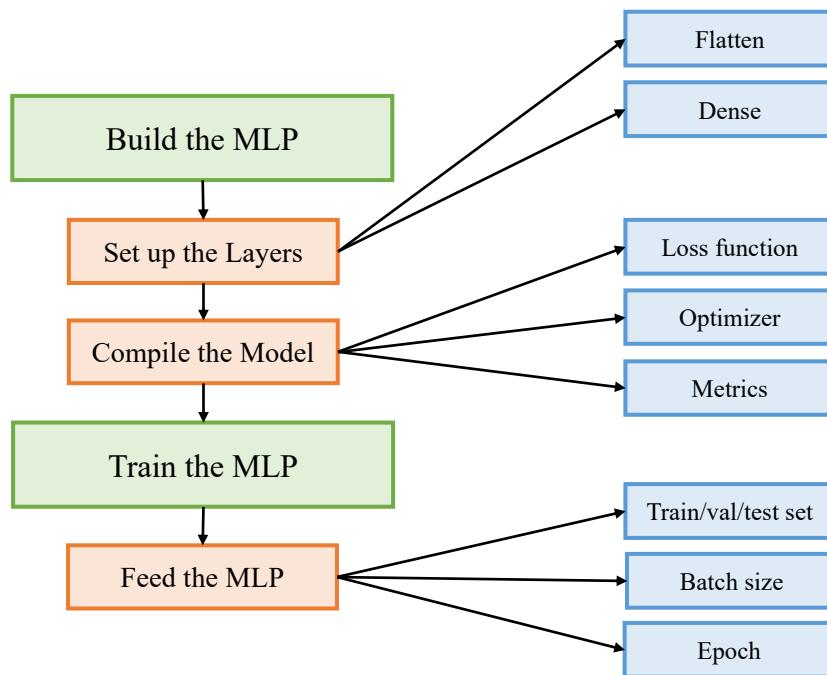


Hình 28.4: Quy trình từng bước cho bài tập 2.

Trong bài tập này, ta làm quen với khả năng của mạng neural network trong việc học các ranh giới quyết định phi tuyến (nonlinear decision boundaries) thông qua bài toán phân loại đa lớp. Bộ dữ liệu gồm 300 mẫu trong không gian 2D được chia thành 3 lớp với phân bố phi tuyến phức tạp. Mô hình MLP gồm 1 lớp ẩn với hàm kích hoạt ReLU, được tối ưu hóa bằng thuật toán Stochastic Gradient Descent với hàm loss Cross Entropy. Quá trình huấn luyện được giám sát qua hai chỉ số: Cross Entropy loss và classification accuracy.

Mục tiêu của bài tập

- Hiểu rõ khả năng của MLP trong việc học các đặc trưng phi tuyến và ranh giới quyết định phức tạp.
- Thành thạo xây dựng mạng MLP cho bài toán phân loại đa lớp với PyTorch.
- Làm việc với dữ liệu dạng .npy (NumPy Array File), xử lý dictionary chứa đặc trưng và nhãn.
- Hiểu và áp dụng hàm loss Cross Entropy cho bài toán phân loại đa lớp, nắm vững công thức tính loss trên batch.
- Cài đặt hàm `compute_accuracy()` để đánh giá hiệu suất phân loại.
- Tự động xác định số lớp đầu ra (`output_dims`) bằng `torch.unique` để mô hình thích ứng với các bài toán khác nhau.
- Thu thập và trực quan hóa đường hội tụ loss/accuracy trên train và validation qua các epoch.
- Hiểu sự khác biệt trong cấu hình mô hình giữa bài regression và classification (số lớp ẩn, activation cuối, hàm loss).



Hình 28.5: Minh họa các bước xây dựng và huấn luyện mô hình trong PyTorch.

Yêu cầu

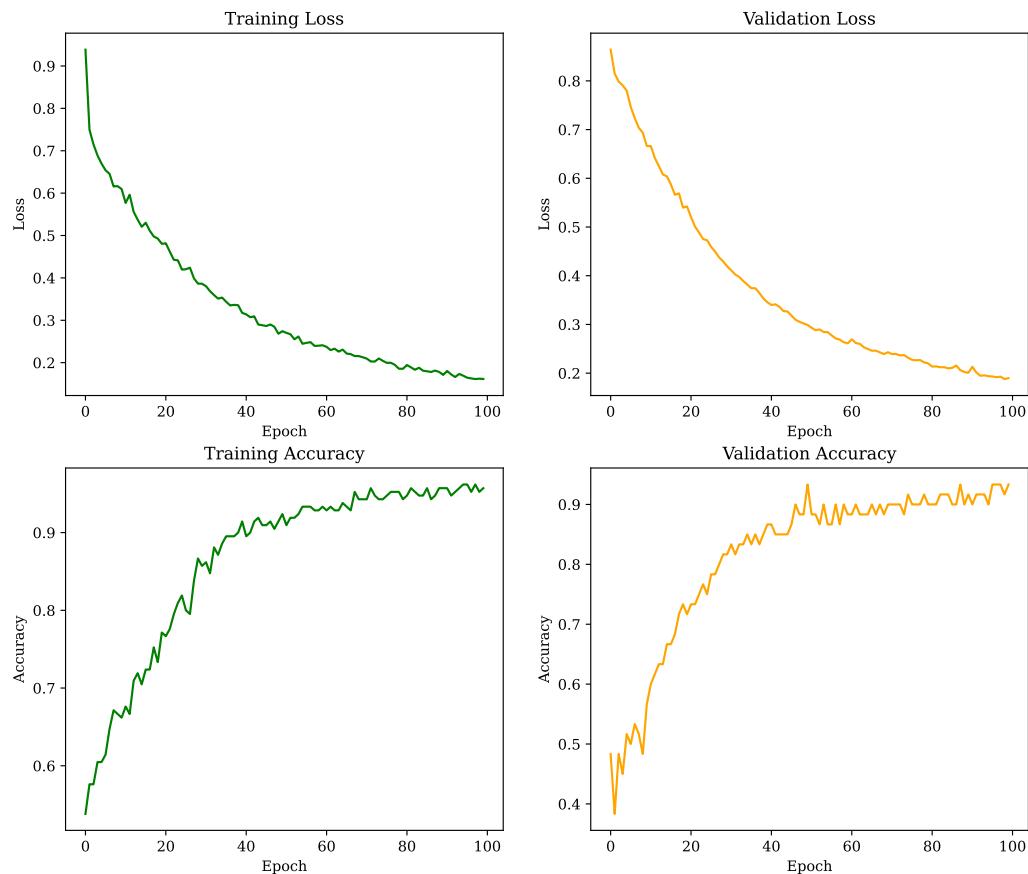
Dựa vào hướng dẫn chi tiết dưới đây, hãy hoàn thiện pipeline huấn luyện mô hình MLP cho bài toán phân loại phi tuyến:

- Chuẩn bị môi trường:** Tải bộ dữ liệu NonLinear_data.npy bằng gdown hoặc thủ công, import các thư viện PyTorch, NumPy, matplotlib, scikit-learn, cài đặt seed cố định và thiết bị GPU/CPU.
- Đọc và khám phá dữ liệu:** Sử dụng `np.load()` với `allow_pickle=True` để đọc file .npy, trích xuất `X` (ma trận 300×2) và `y` (vector 300 nhãn) từ dictionary, kiểm tra shape của dữ liệu.
- Tiền xử lý dữ liệu:** Chia train/validation/test theo tỉ lệ 7:2:1, chuẩn hóa đặc trưng bằng StandardScaler (fit trên train, transform trên cả ba tập), chuyển đổi sang PyTorch tensor với `dtype=torch.float32` cho `X` và `dtype=torch.long` cho `y` (quan trọng cho CrossEntropyLoss).

- **Xây dựng DataLoader:** Tạo class CustomDataset cho cả train, validation và test, khởi tạo DataLoader với batch_size=32, train có shuffle=True, còn val và test có shuffle=False.
- **Kiến trúc mô hình:** Xây dựng class MLP với 1 lớp ẩn (không dùng `squeeze(1)` ở output vì là bài phân loại đa lớp), tự động xác định `output_dims` bằng `torch.unique(y_train).shape[0]`, đặt `hidden_dims=128` để có đủ capacity học đặc trưng phi tuyến phức tạp.
- **Loss và Optimizer:** Khai báo `nn.CrossEntropyLoss()` (đã tích hợp softmax, không cần thêm activation ở output layer) và `torch.optim.SGD()`, hiểu rõ công thức Cross Entropy trên batch: $CE = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^C y_{i,c} \log(\hat{y}_{i,c})$.
- **Hàm đánh giá:** Cài đặt `compute_accuracy` sử dụng `torch.max()` để lấy class có xác suất cao nhất, so sánh với nhãn thực tế và tính tỉ lệ dự đoán đúng.
- **Vòng lặp huấn luyện:** Thu thập predictions và targets từ tất cả batches (sử dụng `.detach().cpu()`, `torch.cat()`), tính accuracy sau khi hoàn tất mỗi epoch trên toàn bộ tập train và validation, lưu lịch sử train/val loss và accuracy.
- **Trực quan hóa:** Vẽ 4 subplot (2×2) hiển thị: Training Loss, Validation Loss, Training Accuracy, Validation Accuracy theo epoch, sử dụng màu xanh cho train và cam cho validation.
- **Đánh giá cuối cùng:** Tính accuracy trên tập test bằng cách thu thập predictions từ `test_loader`, sử dụng `torch.no_grad()` và `model.eval()`.

Bảng cấu hình siêu tham số – Bài 2 (Nonlinear Classification)

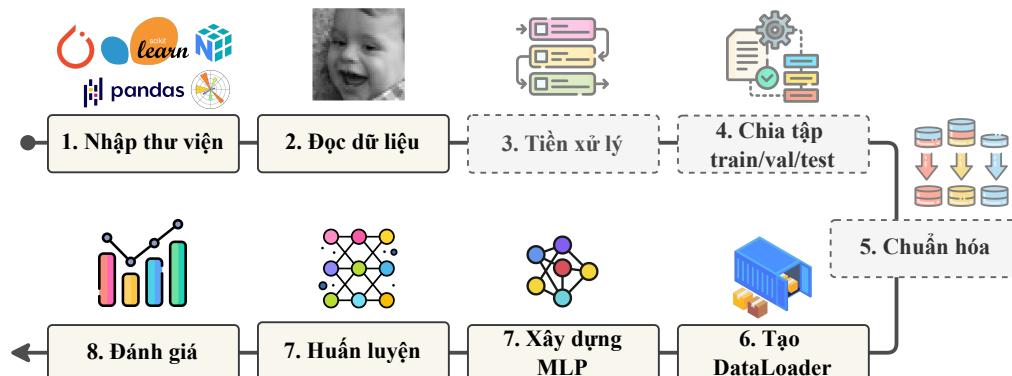
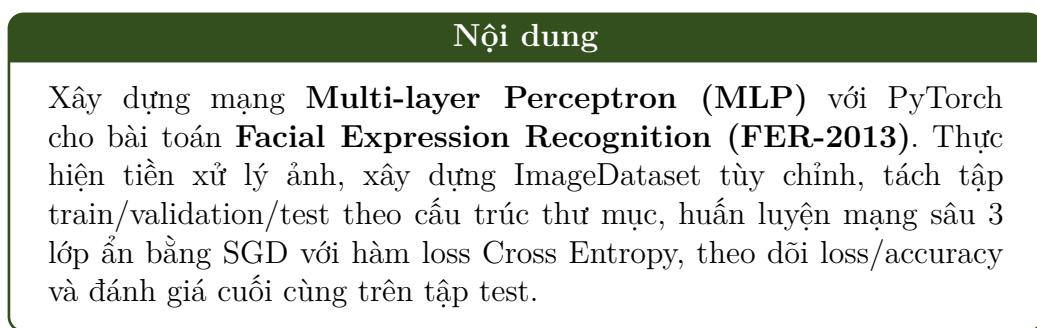
Tham số	Giá trị
Learning rate (<code>lr</code>)	0.1
Số epoch (<code>epochs</code>)	100
Batch size (<code>batch_size</code>)	32
Hidden dimensions (<code>hidden_dims</code>)	128
Input dimensions (<code>input_dims</code>)	2
Output dimensions (<code>output_dims</code>)	3 (auto-detect)
Val/Test split	0.2 / 0.125
Seed cố định (<code>random_state</code>)	59
Loss function	CrossEntropyLoss
Optimizer	SGD
Evaluation metric	Accuracy



Hình 28.6: Trực quan hóa kết quả huấn luyện training/validation loss và accuracy qua các epoch, cho thấy sự hội tụ ổn định của mô hình.

28.2.3 Bài tập 3: Phân loại cảm xúc trên ảnh

Phát biểu bài toán



Hình 28.7: Quy trình từng bước cho bài tập 3.

Trong bài tập này, ta mở rộng khả năng của MLP sang lĩnh vực Computer Vision thông qua bài toán phân loại cảm xúc khuôn mặt từ ảnh grayscale. Bộ dữ liệu FER-2013 chứa hàng nghìn ảnh khuôn mặt được tổ chức theo cấu trúc thư mục, mỗi thư mục con tương ứng với một loại cảm xúc. Mô hình MLP được thiết kế sâu hơn với 3 lớp ẩn (kiến trúc pyramid: 256, 128, 64 nodes) và hàm kích hoạt ReLU, nhằm trích xuất các đặc trưng phức tạp từ không gian pixel. Dữ liệu ảnh được chuẩn hóa về khoảng $[-1, 1]$ và flatten trước khi đưa vào mạng. Bài tập này giúp hiểu rõ cách xử lý dữ liệu ảnh trong PyTorch và xây dựng pipeline hoàn chỉnh cho bài toán phân loại đa lớp trên dữ liệu thực tế.



Hình 28.8: Minh họa một số hình ảnh từ tập huấn luyện với các nhãn cảm xúc tương ứng (angry, happy, sad, surprise, etc.).

Mục tiêu của bài tập

- Làm quen với việc xử lý dữ liệu ảnh trong PyTorch: đọc file .zip, giải nén, quản lý cấu trúc thư mục.
- Xây dựng class `ImageDataset` tùy chỉnh kế thừa `torch.utils.data.Dataset` cho dữ liệu ảnh được tổ chức theo thư mục.
- Sử dụng `torchvision.io.read_image` và `torchvision.transforms.Resize` để đọc và resize ảnh.
- Hiểu và áp dụng kỹ thuật chuẩn hóa ảnh: $(\text{img}/127.5) - 1$ để đưa pixel về khoảng $[-1, 1]$.

- Tự động tạo mapping giữa tên lớp và index (`label2idx`, `idx2label`) từ cấu trúc thư mục.
- Tách train/validation theo tỉ lệ với `stratify` để đảm bảo phân bố cân bằng các lớp.
- Xây dựng mạng MLP sâu với kiến trúc pyramid (3 lớp ẩn) để xử lý dữ liệu ảnh phức tạp.
- Sử dụng `nn.Flatten()` để chuyển tensor ảnh 2D thành vector 1D trước khi đưa vào fully connected layers.
- Trực quan hóa batch dữ liệu ảnh với `matplotlib` để kiểm tra tính đúng đắn của pipeline.
- Huấn luyện mô hình với batch size lớn hơn (256) và theo dõi hội tụ trên dữ liệu thực tế quy mô lớn.

Yêu cầu

Dựa vào hướng dẫn chi tiết dưới đây, hãy hoàn thiện pipeline huấn luyện mô hình MLP cho bài toán phân loại cảm xúc trên ảnh:

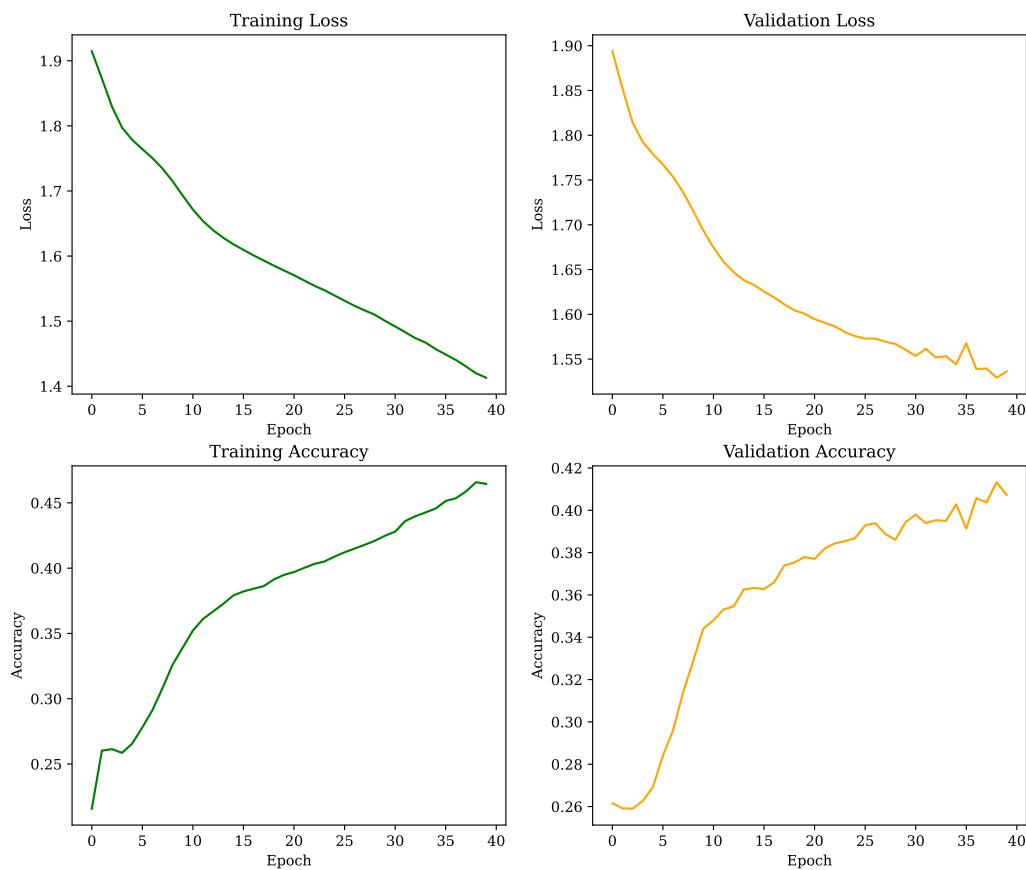
- **Chuẩn bị dữ liệu:** Tải file FER-2013.zip bằng gdown, giải nén bằng lệnh `!unzip -q`, kiểm tra cấu trúc thư mục train/test và các thư mục con (angry, happy, sad, etc.).
- **Import thư viện:** Ngoài PyTorch, NumPy, pandas, cần thêm `cv2`, `torchvision` để xử lý ảnh.
- **Thiết lập môi trường:** Cài đặt seed, device, và kích thước ảnh chuẩn (`img_height=128`, `img_width=128`).
- **Tạo label mapping:** Sử dụng `os.listdir()` để lấy danh sách các lớp cảm xúc, tạo dictionary `label2idx` (mapping tên sang index) và `idx2label` (mapping index sang tên) cho việc encoding/decoding nhãn.
- **Xây dựng ImageDataset:** Tạo class kế thừa Dataset với các chức năng:

- `read_img_files()`: Duyệt qua cấu trúc thư mục, thu thập đường dẫn ảnh và nhãn.
 - `__init__()`: Thực hiện `train_test_split` với `stratify=self-img_labels` để chia train/val cân bằng (80/20).
 - `__getitem__()`: Đọc ảnh bằng `read_image()`, resize về 128×128 , chuyển sang `float32`, áp dụng normalization $(\text{img}/127.5) - 1$ nếu `norm=True`, trả về tuple (img, label).
 - `__len__()`: Trả về số lượng ảnh.
- **Tạo DataLoader:** Khởi tạo DataLoader cho train (`shuffle=True`), validation (`shuffle=False`), và test (`shuffle=False`) với `batch_size=-256`.
 - **Trực quan hóa dữ liệu:** Sử dụng `next(iter(train_loader))` để lấy một batch, vẽ 9 ảnh đầu tiên (subplot 3×3) với `plt.imshow()` và `cmap="gray"`, hiển thị nhãn cảm xúc tương ứng.
 - **Kiến trúc MLP sâu:** Xây dựng mạng với 3 lớp ẩn theo kiến trúc pyramid:
 - Layer 1: `input_dims` sang `hidden_dims*4` (128×128 sang 256)
 - Layer 2: `hidden_dims*4` sang `hidden_dims*2` (256 sang 128)
 - Layer 3: `hidden_dims*2` sang `hidden_dims` (128 sang 64)
 - Output: `hidden_dims` sang `output_dims` (64 sang `num_classes`)
 - Sử dụng `nn.Flatten()` ở đầu forward pass để chuyển ảnh 2D thành vector 1D.
 - ReLU activation sau mỗi lớp ẩn, không có activation ở output (CrossEntropyLoss đã tích hợp softmax).
 - **Loss và Optimizer:** Khai báo `nn.CrossEntropyLoss()` và `torch.optim.SGD()` với `lr=0.01`.
 - **Hàm đánh giá:** Sử dụng lại hàm `compute_accuracy` từ bài 2.
 - **Vòng lặp huấn luyện:** Huấn luyện 40 epochs, thu thập predictions từ tất cả batches bằng `.detach().cpu()` và `torch.cat()`, tính loss/accuracy cho cả tập train và validation mỗi epoch.

- **Trực quan hóa kết quả:** Vẽ 4 subplot (2×2) cho Training/Validation Loss và Accuracy theo epoch.
- **Đánh giá test set:** Tính accuracy trên tập test sau khi huấn luyện hoàn tất.

Bảng cấu hình siêu tham số – Bài 3 (FER-2013)

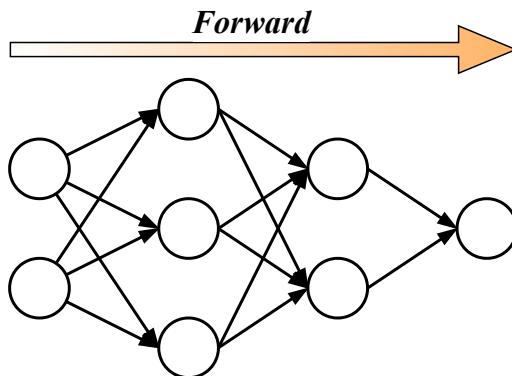
Tham số	Giá trị
Learning rate (lr)	0.01
Số epoch (epochs)	40
Batch size (batch_size)	256
Hidden dimensions (hidden_dims)	64
Image size (img_height \times img_width)	128×128
Input dimensions (input_dims)	16384 (128×128)
Output dimensions (output_dims)	7 (auto-detect)
Train/Val split	0.8 / 0.2
Normalization	(img/127.5) - 1
Seed cố định (random_state)	59
Loss function	CrossEntropyLoss
Optimizer	SGD
Evaluation metric	Accuracy
MLP architecture	3 hidden layers (256, 128, 64)



Hình 28.9: Trực quan hóa kết quả huấn luyện trainin/validation loss và accuracy qua 40 epochs, cho thấy khả năng của MLP trong việc học các đặc trưng cảm xúc từ ảnh khuôn mặt.

28.3 Câu hỏi trắc nghiệm

1. Mạng nơ-ron trong hình có cấu trúc như thế nào?:

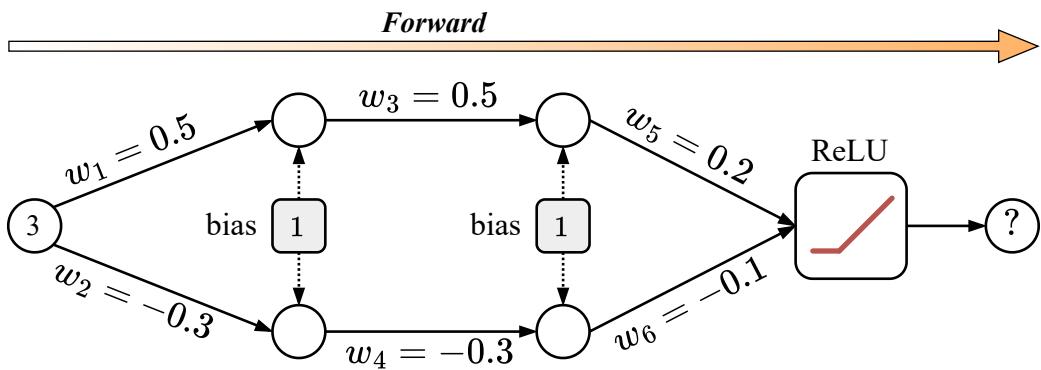


- (a) Lớp input: 5 neuron, các lớp hidden: 5 neuron, lớp output: 1 neuron.
 - (b) Lớp input: 2 neuron, các lớp hidden: 3 neuron, lớp output: 5 neuron.
 - (c) Lớp input: 2 neuron, các lớp hidden: 5 neuron, lớp output: 1 neuron.
 - (d) Lớp input: 1 neuron, các lớp hidden: 5 neuron, lớp output: 1 neuron.
2. Khi một nơ-ron trong mạng MLP có đầu vào \mathbf{x} là $[2, 3]$ và trọng số \mathbf{w} là $[0.5, -0.5]$, bias \mathbf{b} bằng 1 và hàm activation là hàm ReLU, đầu ra là:
- (a) 0.5.
 - (b) 1.0.
 - (c) 1.5.
 - (d) 0.0.
3. Trong ngữ cảnh của neural networks, back-propagation được sử dụng để:
- (a) Tính toán đầu vào cho mạng.
 - (b) Tính toán loss output layer.
 - (c) Cập nhật trọng số dựa trên loss.
 - (d) Tăng số lượng hidden layers.

4. Nếu bạn có một MLP model với một hidden layer chứa 5 neurons và output layer có 2 neurons, số lượng trọng số của mạng là:

- (a) 5.
- (b) 7.
- (c) 10.
- (d) 12.

5. Cho mạng nơ-ron nhiều lớp (MLP) như hình, với đầu vào $x = 3$. Giá trị đầu ra cuối cùng của mạng là bao nhiêu?



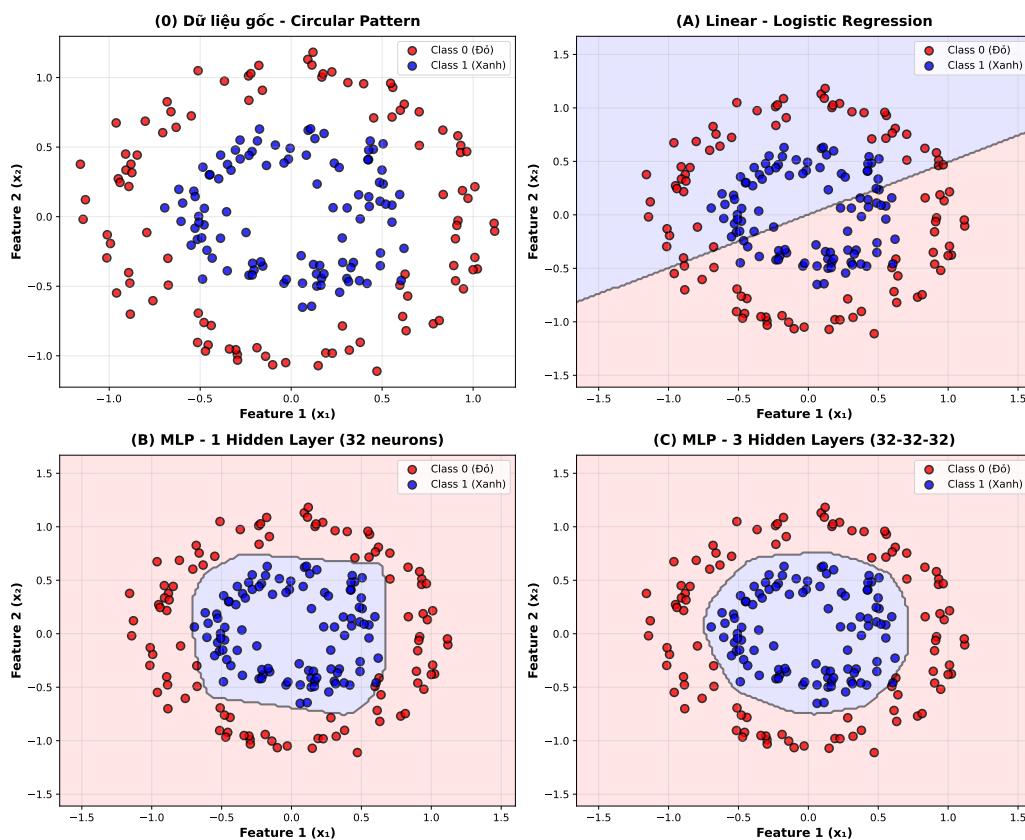
- (a) 0.100.
- (b) 0.353.
- (c) 1.250.
- (d) 2.250.

6. Phân loại hoa dựa trên chiều dài cánh bằng cách sử dụng MLP model có 1 hidden layer chứa 2 neuron (trọng số $w_{hidden} = [0.5, -0.5]$, và bias $b_{hidden} = [1, -1]$) đi kèm với hàm ReLU và output layer chứa 1 neuron (trọng số $w_{output} = [0.3, -0.2]$, và bias $b_{output} = 0.5$) đi kèm với hàm Sigmoid. Nếu cho input $x = 5.1$ thì model phân loại thuộc loại nào (> 0.5 được xem như loại 1):

Chiều dài cánh	Loại cánh
5.1	0
6.0	1
5.7	0

Bảng 28.1: Bảng dữ liệu phân loại cánh hoa theo chiều dài cánh.

- (a) Loại 0.
 (b) Loại 1.
 7. Cho dữ liệu gốc như hình bên dưới, Decision boundary của mô hình nào là phù hợp nhất cho dữ liệu này mà không bị overfitting?



- (a) Linear.
- (b) MLP 1 hidden layer.
- (c) MLP 3 layer.
- (d) Cả 3 mô hình trên.
8. Tính kết quả dự đoán giá nhà dựa vào diện tích bằng cách sử dụng MLP model có 1 hidden layer chứa 2 neuron (trọng số $w_{hidden} = [0.5, -0.5]$, và bias $b_{hidden} = [1, -1]$) đi kèm với hàm ReLU và output layer chứa 1 neuron (trọng số $w_{output} = [1.5, 2]$, và bias $b_{output} = 0$). Nếu cho input $x = 50$ thì model dự đoán giá nhà là bao nhiêu:
- | Diện tích | Giá nhà |
|-----------|---------|
| 50 | 100 |
| 60 | 120 |
| 70 | 140 |
- Bảng 28.2: Bảng dữ liệu mô phỏng giá nhà theo diện tích.
- (a) 39.
- (b) 390.
- (c) 93.
- (d) 930.
9. Dựa vào đoạn code dưới để chọn đáp án đúng. Shape của output là?

```

1 import torch
2 import torch.nn as nn
3
4 model = nn.Sequential(
5     nn.Linear(784, 128),
6     nn.ReLU(),
7     nn.Linear(128, 64),
8     nn.ReLU(),
9     nn.Linear(64, 10)
10 )

```

```
11  
12 # Lấy weight của layer thứ 3  
13 W3 = list(model.parameters())[4]  
14 print(W3.shape)
```

- (a) torch.Size([128, 64]).
(b) torch.Size([64, 128]).
(c) torch.Size([10, 64]).
(d) torch.Size([64, 10]).
10. Dựa vào đoạn code dưới để chọn đáp án đúng. Giá trị loss xấp xỉ bằng bao nhiêu?

```
1 import torch  
2 import torch.nn as nn  
3  
4 y_pred = torch.tensor([[2.0, 1.0, 0.1]])  
5 y_true = torch.tensor([0])  
6  
7 criterion = nn.CrossEntropyLoss()  
8 loss = criterion(y_pred, y_true)
```

- (a) 0.417.
(b) 0.659.
(c) 1.0.
(d) 2.0.

1. **Dataset:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
2. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
3. **Solution:** Các file code đáp án có thể được tải tại [đây](#).
4. **Rubric:**

Mục	Kiến thức	Đánh giá
I.	<ul style="list-style-type: none"> - Kiến thức về việc cài đặt và huấn luyện mô hình Multi-layer Perceptron với PyTorch. - Kiến thức về các thành phần trong MLP (Input layer, Hidden layers, Output layer, Activation functions). - Kiến thức về Forward Propagation và Backpropagation. - Kiến thức về các phương pháp tiền xử lý dữ liệu dạng bảng, dữ liệu phi tuyến và dữ liệu ảnh. - Kiến thức về PyTorch Dataset, DataLoader và cách xây dựng pipeline huấn luyện hoàn chỉnh. 	<ul style="list-style-type: none"> - Nắm được cách cài đặt mô hình MLP với PyTorch từ đầu. - Nắm được các thành phần và cơ chế hoạt động của MLP. - Hiểu được cách vận hành của Forward và Backward Propagation trong neural networks.
II.	<ul style="list-style-type: none"> - Các kiến thức và nội dung cơ bản về Multi-layer Perceptron. 	<ul style="list-style-type: none"> - Nắm được kiến thức cơ bản về MLP và ứng dụng cho các bài toán regression, classification.

5. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại [đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 tiếng.

AIO_QAs-Verified

🔒 Private group · 1.4K members



Hình 28.10: Hình ảnh group Facebook AIO Q&A.

Chương 29

Project 1: Dự đoán chính xác cho dữ liệu time-series với mô hình DLinear và NLinear

29.1 Giới thiệu

Dự báo chuỗi thời gian, đặc biệt là trong lĩnh vực tài chính, là một trong những bài toán kinh điển và cũng đầy thách thức của ngành AI. Việc thành công trong dự đoán giá cổ phiếu không chỉ đem về cho ta lợi nhuận, mà còn là một công cụ cốt lõi để quản trị rủi ro. Tuy nhiên, thử thách của bài toán này thực sự không nằm ở việc đoán giá của ngày mai, mà là việc “nhìn xa” (Long-term Time Series Forecasting - LTSF). Dữ liệu tài chính nổi tiếng là cực kỳ “nhiều” (noisy), phi tuyến tính (non-linear) và không có tính dừng (non-stationary). Dự đoán vài ngày tới đã khó, dự đoán hàng tuần hay hàng tháng là một bài toán ở cấp độ hoàn toàn khác.



Hình 27: Sự biến động phức tạp và “nhiều” của thị trường chứng khoán, một thách thức lớn cho các mô hình dự báo dài hạn.

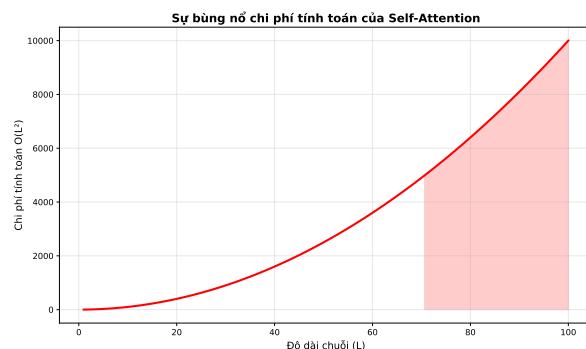
Để giải quyết bài toán LTSF một cách hiệu quả, mô hình không chỉ đơn thuần là mở rộng cửa sổ dự báo. Thách thức cốt lõi nằm ở việc nắm bắt các phụ thuộc dài hạn (long-term dependencies). Trong dữ liệu tài chính, một sự kiện ở thời điểm t (ví dụ: giá hôm nay) có thể không chỉ phụ thuộc vào

$t - 1$ (hôm qua), mà còn bị ảnh hưởng mạnh mẽ bởi các mô hình (patterns) đã xảy ra từ nhiều tháng, hoặc thậm chí nhiều năm trước. Ví dụ, tính mùa vụ (seasonality) có thể tạo ra các liên hệ cách nhau 12 tháng, hoặc các chu kỳ kinh tế (market cycles) có thể tạo ra các phụ thuộc kéo dài 3-5 năm. Các phương pháp dự báo truyền thống, thường thất bại trong việc “ghi nhớ” và liên kết các thông tin ở quá khứ xa xôi này. Do đó, yêu cầu cấp thiết là phải có một kiến trúc có khả năng “nhìn” và kết nối hiệu quả các điểm dữ liệu bất kể chúng cách xa nhau bao nhiêu, cho phép mô hình phân biệt được tín hiệu dài hạn thực sự khỏi nhiều cục bộ.

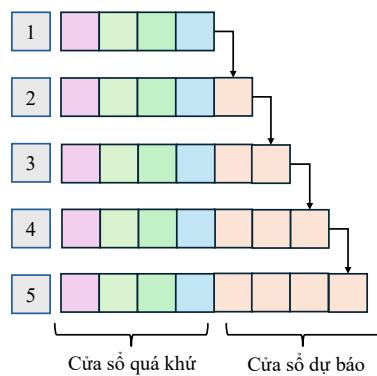
Khi nghe đến vấn đề phụ thuộc dài hạn, chúng ta thường nghĩ ngay đến lớp mô hình Transformer. Với cơ chế self-attention, các mô hình như Informer hay Autoformer hứa hẹn khả năng nhìn toàn bộ dòng lịch sử để tìm ra các mối quan hệ, phức tạp.

Nhưng có một vấn đề lớn với mô hình Transformer chính là độ phức tạp $O(L^2)$ của thuật toán. Khi L là độ dài chuỗi đầu vào (ví dụ $L = 720$, tương đương dữ liệu giao dịch 3 năm theo ngày), chi phí tính toán và bộ nhớ sẽ tăng vọt. Điều này khiến việc huấn luyện các mô hình Transformer cho chuỗi rất dài trở nên cực kỳ tốn kém, thậm chí là không khả thi trong thời gian thật.

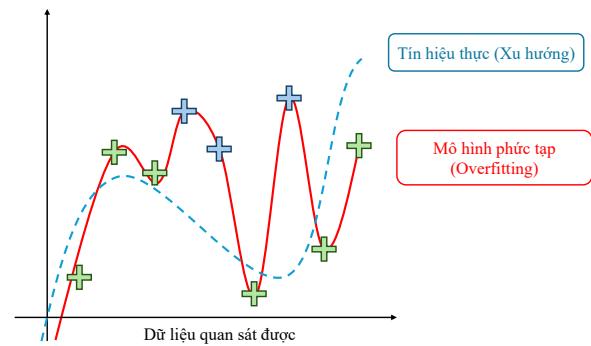
Điều này đặt ra một câu hỏi then chốt, đặc biệt là với dữ liệu nhiều như tài chính: Liệu một kiến trúc siêu phức tạp có đang vô tình ‘học vẹt’ (overfit) chính cái nhiễu đó, thay vì nắm bắt tín hiệu thực sự?



Hình 28: Sự bùng nổ về chi phí tính toán của cơ chế self-attention. Khi độ dài chuỗi L tăng lên, chi phí $O(L^2)$ khiến mô hình trở nên không thực tế cho các tác vụ dự báo rất dài. Vùng màu đỏ là khu vực mà VRAM sử dụng lớn hơn 8GB.

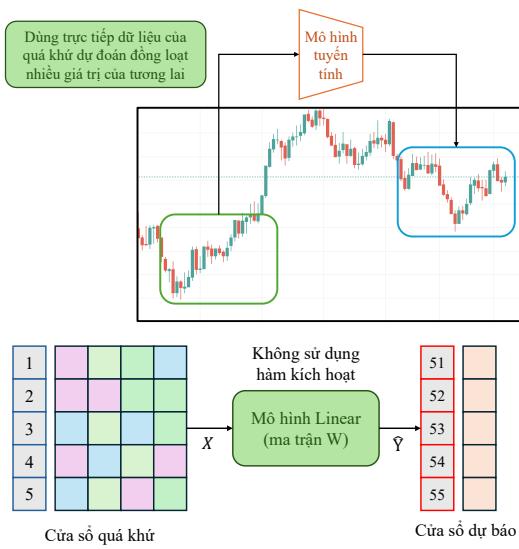


Hình 29: Các mô hình chuỗi thời gian thường dự đoán tương lai bằng cách dự đoán tuần tự $t, t+1, \dots, t+n$.



Hình 30: Minh họa hiện tượng Overfitting trong dữ liệu chuỗi thời gian. Các mô hình phức tạp thường ‘học vẹt’ một cách hoàn hảo các dao động ngẫu nhiên (nhiều) có trong dữ liệu.

Giữa bối cảnh đó, một nhóm mô hình ‘quay về cơ bản’ (back-to-basics) mang tên LTSF-Linear xuất hiện. Ý tưởng của chúng đơn giản đến bất ngờ: hãy mô hình hóa toàn bộ trực thời gian chỉ bằng một phép ánh xạ tuyến tính.



Hình 31: Kiến trúc của mô hình Linear.
Thay vì dự đoán tuần tự các ngày trong tương lai, chúng chỉ cần chạy mô hình một lần.

- Linear: Mô hình cơ sở, ‘kéo’ trực tiếp cửa sổ quá khứ sang tương lai bằng một ma trận trọng số duy nhất.
- DLinear: Cải tiến bằng cách phân rã (decompose) tín hiệu thành xu thế (trend) và dao động (seasonality), sau đó dùng hai mô hình Linear riêng biệt.
- NLinear: Bổ sung bước chuẩn hóa (normalization) để giảm lệch phân phối (distribution shift) trước khi đưa vào mô hình Linear.

Đặc biệt, trong các benchmark, những mô hình tuyến tính đơn giản này không chỉ nhẹ và ổn định, mà còn cho ra kết quả cạnh tranh, thậm chí vượt trội so với các Transformer phức tạp. Điều này dấy lên một giả thuyết mạnh mẽ: trong một miền ‘nhiều’ như tài chính, việc tách lọc tín hiệu (như DLinear làm) có thể còn quan trọng hơn là cố gắng mô hình hóa mọi mối quan hệ phức tạp có thể có trong không gian bài toán.

Mục tiêu project: Project này không chỉ là một bài hướng dẫn để ‘chạy lại’ code mà còn là một project cơ sở để bạn hiểu việc xử lý dữ liệu chuỗi thời gian và đồng thời cải tiến chúng. Nhiệm vụ của bạn là:

- Hiểu rõ cách triển khai và so sánh Linear, NLinear, và DLinear trên dữ liệu tài chính thực tế.
- Phân tích (và phản biện) kết quả: Tại sao một mô hình đơn giản lại có thể hoạt động tốt? Giới hạn của nó ở đâu?
- Thử nghiệm và cải tiến: Bạn sẽ làm gì tiếp theo để cải thiện hiệu suất dự báo từ cái nền tảng này? (Ví dụ: thay đổi cửa sổ look-back, thử nghiệm với các nhóm cổ phiếu khác nhau, hay kết hợp các ý tưởng?)

Project này được thiết kế cho các bạn:

- Học viên AI/data science: Muốn áp dụng và ‘thách thức’ các mô hình time series trên dữ liệu tài chính phức tạp.
- Nhà phân tích định lượng (quants): Cần hiểu rõ các mô hình baseline mạnh, nhẹ để làm nền tảng cho các hệ thống phức tạp hơn.
- Bất kỳ ai tò mò: Giữa các lớp mô hình ‘đơn giản’ và ‘phức tạp’ trong machine learning, có nhất thiết càng phức tạp sẽ càng tốt hơn không?

Để việc đọc thuận tiện hơn, bảng sau liệt kê và chuẩn hoá các ký hiệu sẽ xuất hiện xuyên suốt nội dung bài.

Bảng Ký hiệu Toán học (LTSF)

Ký hiệu	Ý nghĩa
L	Độ dài cửa sổ quá khứ.
T	Độ dài dự báo tương lai.
B	Kích thước một batch.
$\mathbf{x} \in \mathbb{R}^L$	Vector input, chứa L giá trị quá khứ.
$\mathbf{y} \in \mathbb{R}^T$	Vector mục tiêu, chứa T giá trị thực tương lai.
$\hat{\mathbf{y}} \in \mathbb{R}^T$	Vector dự đoán, chứa T giá trị dự đoán tương lai.
x_t	Giá trị của chuỗi thời gian tại thời điểm t .
y_t	Giá trị thực tại bước tương lai thứ t .
\hat{y}_t	Giá trị dự đoán tại bước tương lai thứ t .
$W \in \mathbb{R}^{T \times L}$	Ma trận trọng số của mô hình Linear.
$b \in \mathbb{R}^T$	Vector bias của mô hình Linear.
ℓ	Mốc cục bộ (giá trị cuối x_t của \mathbf{x}) để tính tiền.
$\mathbf{1}_L, \mathbf{1}_T$	Vector chỉ chứa số 1, với độ dài L và T .
\mathbf{x}'	Vector input \mathbf{x} đã được tính tiền (trừ ℓ).
$\hat{\mathbf{y}}'$	Vector dự đoán $\hat{\mathbf{y}}$ trước khi hoàn nguyên (cộng ℓ).
m	Kích thước cửa sổ của Moving Average.
$\text{MA}_m(\cdot)$	Toán tử moving average với cửa sổ m .
x_t	Thành phần xu hướng của \mathbf{x} .
x_s	Thành phần mùa vụ của \mathbf{x} .
W_t, b_t	Ma trận trọng số và bias cho nhánh xu hướng.
W_s, b_s	Ma trận trọng số và bias cho nhánh mùa vụ.
\hat{y}_t, \hat{y}_s	Vector dự đoán cho nhánh Trend và Seasonal.

29.2 Kiến thức cơ bản về dự báo chuỗi thời gian

Trước khi đi vào mô hình, chúng ta đi qua cách đóng gói dữ liệu chuỗi thời gian để đưa vào mô hình học máy, kèm các khái niệm nền tảng về loại bài toán, đầu vào/đầu ra, và cách đánh giá các mô hình trên bài toán này.

29.2.1 Bài toán dự báo chuỗi thời gian là gì?

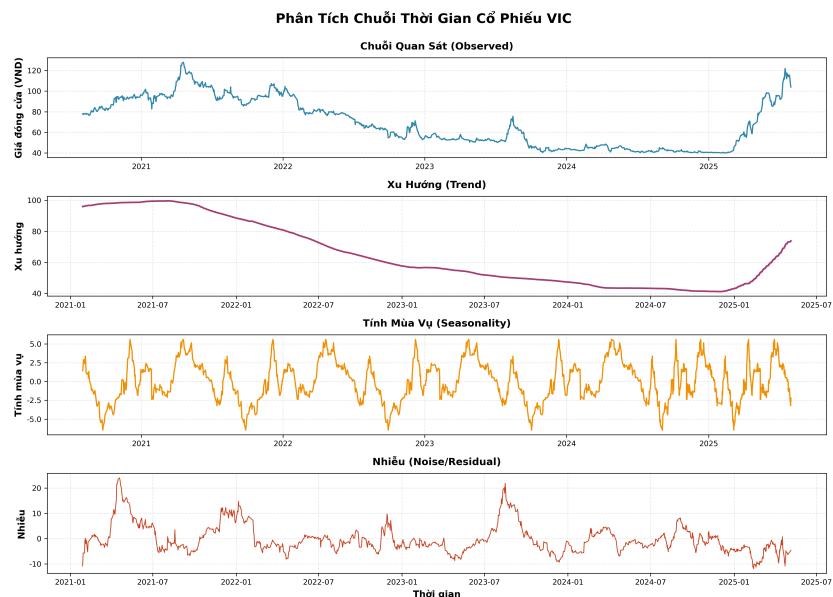
Một chuỗi thời gian đơn giản là một chuỗi các điểm dữ liệu được ghi lại theo thứ tự thời gian, ví dụ như giá cổ phiếu mỗi ngày, hoặc nhiệt độ mỗi giờ. Mục tiêu của bài toán dự báo là: dùng dữ liệu quá khứ để dự đoán dữ liệu tương lai. Chúng ta gọi độ dài của quá khứ mà mô hình nhìn vào là cửa sổ quá khứ (Look-back, L), và số bước tương lai cần dự đoán là chân trời dự đoán (Horizon, H).

Một cách tổng quát, chúng ta đang cố gắng tìm một hàm dự báo để học được quy luật này:

$$\underbrace{y_{t+1:t+H}}_{\text{đầu ra}} = \text{hàm dự báo} \left(\underbrace{y_{t-L+1:t}}_{\text{quá khứ}}, \underbrace{x_{t-L+1:t}^{\text{đã quan sát}}}_{\text{biến phụ trợ đã quan sát}}, \underbrace{x_{t+1:t+H}^{\text{biết trước}}}_{\text{biến phụ trợ biết trước}}, \underbrace{s}_{\text{đặc trưng tĩnh}} \right) + \underbrace{\varepsilon}_{\text{nhiều}}$$

Công thức này trông phức tạp, nhưng nó chỉ mô tả những gì chúng ta đã nói. Để dự đoán chính xác hơn (đầu ra màu xanh), chúng ta không chỉ dùng giá trị trong quá khứ (màu đỏ). Chúng ta cần cung cấp thêm thông tin, gọi là **biến phụ trợ** (covariates). Các loại này bao gồm **biến phụ trợ đã quan sát** (màu xanh lá), như lượng mưa hôm qua. Chúng cũng bao gồm các **biến phụ trợ biết trước** (màu tím), như ngày mai là chủ nhật. Và cuối cùng là các **đặc trưng tĩnh** (màu cam), như id của cửa hàng.

Khi nhìn vào một chuỗi thời gian, chúng ta thường có thể phân tách nó thành các thành phần. Các thành phần chính bao gồm **xu hướng** (trend), là hướng đi chung của dữ liệu; **mùa vụ** (seasonality), là các mẫu lặp lại có chu kỳ; và **nhiều** (noise), là các biến động ngẫu nhiên. Các mô hình khác nhau sẽ có cách xử lý các thành phần này khác nhau.

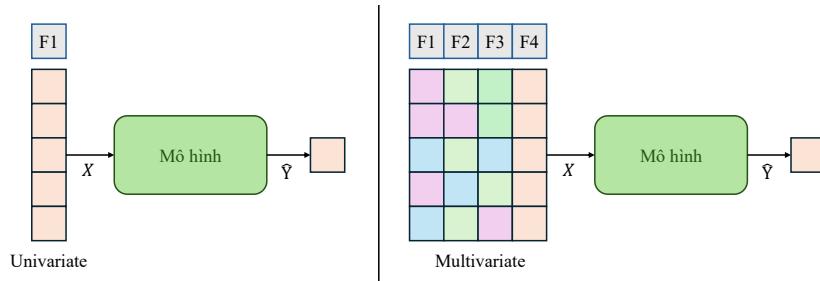


Hình 32: Phân tích chuỗi thời gian chúng ta sẽ sử dụng trong project này. Ta thấy rằng bài toán chúng ta có rất nhiều nhiễu, như ở plot cuối cùng có nhiều đường lún xuồng.

29.2.2 Dữ liệu trông như thế nào?

Về cấu trúc, dữ liệu đầu vào có thể là đơn biến (univariate), nghĩa là chúng ta chỉ có một đặc trưng duy nhất (ví dụ: chỉ dùng chuỗi giá đóng cửa của cổ phiếu A). Hoặc, dữ liệu có thể là đa biến (multivariate), khi chúng ta sử dụng nhiều đặc trưng đầu vào cùng một lúc (ví dụ: dùng giá đóng cửa, khối lượng giao dịch của cổ phiếu A, và cả chỉ số VN-Index) để đưa ra dự đoán.

Một khái niệm liên quan là cấu trúc đầu ra (output). Chúng ta có thể dự đoán một chuỗi (ví dụ: chỉ dự đoán giá cổ phiếu A), hoặc dự đoán nhiều chuỗi cùng lúc (ví dụ: dự đoán giá của A, B, và C). Hình dưới minh họa sự khác biệt về cấu trúc đầu vào.

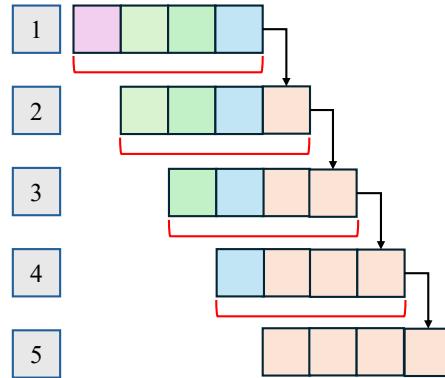


Hình 33: Hai cách biểu diễn đầu vào của bài toán chuỗi thời gian. Trong đó, cột cam là cột dữ liệu chứa các biến cần dự đoán.

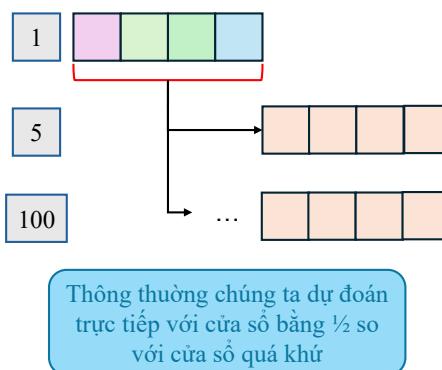
29.2.3 Chúng ta muốn dự đoán cái gì?

Khi nói về kiểu dự báo, chúng ta có thể dự đoán 1 bước (one-step) là giá trị ngay tiếp theo, hoặc **dự đoán nhiều bước (multi-step)** là một chuỗi giá trị trong tương lai ($H > 1$). Đây là mục tiêu chính của chúng ta ở trong bài toán này.

Chiến lược phổ biến thứ nhất cho bài toán multi-step là **đệ quy (recursive)**. Theo cách này, mô hình đầu tiên dự đoán bước thời gian $t + 1$. Sau đó, kết quả dự đoán (giả) này được sử dụng làm đầu vào mới để mô hình tiếp tục dự đoán bước $t + 2$. Quá trình này lặp đi lặp lại cho đến khi đủ H bước. Mặc dù trực quan, phương pháp này có một rủi ro rất lớn là tích lũy lỗi (compounding errors): một lỗi nhỏ ở bước $t + 1$ có thể bị khuếch đại và làm sai lệch nghiêm trọng các dự đoán ở $t + H$.



Hình 34: Chiến lược dự đoán đệ quy: Mô hình dự đoán từng bước một, sử dụng kết quả dự đoán trước đó làm đầu vào cho bước tiếp theo.



Hình 35: Chiến lược dự đoán trực tiếp: Mô hình ánh xạ toàn bộ quá khứ đến toàn bộ tương lai trong một lần, tránh tích lũy lỗi.

Chiến lược thứ hai, hiệu quả hơn là **trực tiếp (direct)**. Với chiến lược này, mô hình được huấn luyện để học cách ánh xạ thẳng từ toàn bộ cửa sổ dữ liệu quá khứ (X) để dự đoán đồng thời *toàn bộ* H bước tương lai (\hat{Y}) chỉ trong một lần duy nhất. Phương pháp này hoàn toàn tránh được vấn đề tích lũy lỗi, vì mỗi dự đoán đều dựa trên dữ liệu thực tế. Đây chính là chiến lược cốt lõi được các mô hình Linear, NLinear, và DLinear sử dụng.

Cuối cùng, kết quả dự báo có thể là một **dự đoán điểm (point forecast)**, là một con số duy nhất như dự đoán ngày mai giá là 100. Hoặc, nó có thể là một **dự đoán xác suất (probabilistic forecast)**, trả về một khoảng giá trị, như dự đoán ngày mai giá nằm trong khoảng 95-105 với độ chắc chắn 90%.

29.2.4 Liên hệ tới các mô hình trong project này

Các mô hình trong project này đều áp dụng chiến lược dự báo trực tiếp. Mô hình **NLinear** tập trung vào việc chuẩn hóa (normalization) dữ liệu đầu vào để xử lý vấn đề khi dữ liệu trong tương lai trông khác nhiều so với quá khứ. Mô hình **DLinear** thì thực hiện theo một cách khác: nó tự động tách chuỗi thời gian thành hai phần là xu hướng và mùa vụ. Sau đó, nó dùng hai mô hình linear riêng biệt để học hai phần này, hoạt động rất tốt cho dữ liệu có xu hướng và mùa vụ rõ rệt.

29.3 Mô hình LTSF-Linear: Linear

Cơ chế: Linear mô hình hoá dự đoán dài hạn như một phép biến đổi tuyến tính dọc trực thời gian cho một biến (univariate). Với mỗi mẫu trong batch, ta có input $\mathbf{x} \in \mathbb{R}^L$ (cửa sổ quá khứ L bước) và output $\mathbf{y} \in \mathbb{R}^T$ (dự đoán T

bước). Mô hình học ma trận $W \in \mathbb{R}^{T \times L}$ và bias $b \in \mathbb{R}^T$:

$$\hat{\mathbf{y}} = W \mathbf{x} + b \in \mathbb{R}^T.$$

Huấn luyện (DMS - Direct Multi-step). Tối ưu đồng thời cả T bước bằng MSE:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{BT} \sum_{n=1}^B \sum_{t=1}^T (\hat{y}_t - y_t)^2.$$

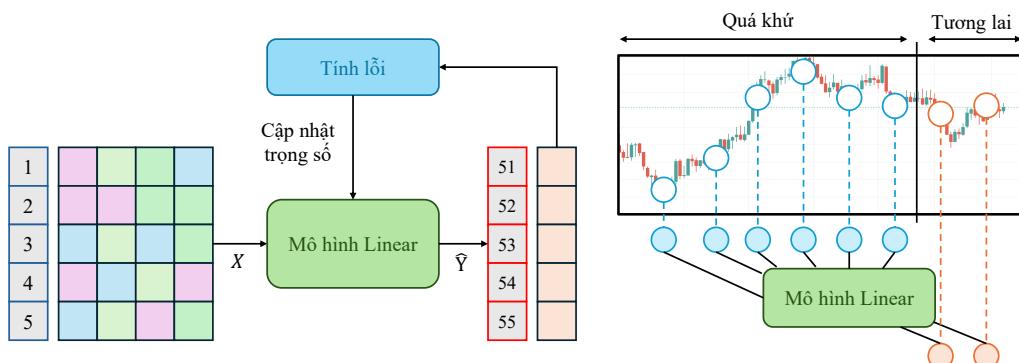
Trong đó:

- T : số bước dự đoán (horizon); B : kích thước batch.
- \hat{y}_t : giá trị **dự đoán** tại bước tương lai t ; y_t : giá trị **thực**.
- MSE là **trung bình** sai số bình phương trên toàn bộ T bước và B mẫu.

Độ phức tạp. Tham số $\approx T \times L$ (cộng T bias); chi phí tính toán tỉ lệ $B \times T \times L \Rightarrow$ huấn luyện/suy luận nhanh, ổn định.

M

ô hình Linear trực tiếp biến đổi tuyến tính từ L bước gần nhất để dự đoán T bước kế tiếp, phù hợp khi chuỗi chúng ta đang xét ổn định. Nếu chuỗi có thuộc tính phi tuyến mạnh, có nhiều thay đổi đột ngột, hoặc cần mô hình hoá quan hệ đa biến, cân nhắc NLinear/DLinear hoặc các mô hình phi tuyến khác.



Hình 36: Quy trình chạy và huấn luyện của mô hình LTTSF-Linear.

29.4 Mô hình LTSF-Linear: NLinear

Cơ chế: Mô hình NLinear mở rộng lớp Linear cho dự đoán đơn biến bằng bước tịnh tiến dữ liệu về mốc cục bộ (**re-centering**) theo giá trị quan sát cuối của cửa sổ đang dự đoán, nhằm làm mô hình tập trung học sự biến thiên tương đối và giảm nhẹ cảm trước dịch mức cộng tính (**additive level shift**). Hai thuật ngữ khó hiểu nói trên sẽ được diễn giải dưới đây.

Thuật ngữ:

- **Additive level shift:** toàn bộ các giá trị trong một đoạn thời gian cùng tăng hoặc cùng giảm một lượng như nhau. Nói cách khác, “mặt bằng” của chuỗi được đẩy lên hoặc hạ xuống nhưng hình dáng dao động (nhịp lên xuống, độ chênh giữa các điểm liền kề) hầu như không đổi.
- **Re-centering:** chọn một mốc tham chiếu gần nhất (thường là giá trị cuối cửa sổ), trừ mốc này khỏi mọi giá trị trong cửa sổ để đưa chuỗi về quanh mốc đó, huấn luyện mô hình trên phần độ lệch tương đối, rồi cộng lại mốc tham chiếu vào đầu ra. Cách này giúp mô hình ít bị ảnh hưởng khi mặt bằng dữ liệu dịch lên/xuống đồng đều.

Thiết lập dữ liệu. Với chuỗi $\{x_t\}$, tại thời điểm t tạo

$$\mathbf{x} = [x_{t-L+1}, \dots, x_t] \in \mathbb{R}^L, \quad \mathbf{y} = [x_{t+1}, \dots, x_{t+T}] \in \mathbb{R}^T.$$

Trong đó (thiết lập):

- \mathbf{x} : input quá khứ dài L bước; \mathbf{y} : mục tiêu dự đoán T bước tương lai.
- $\mathbf{1}_L, \mathbf{1}_T$: vectơ với mọi phần tử đều bằng 1, kích thước L và T .

Công thức. Đặt mốc cục bộ $\ell = x_t$, thực hiện ba bước: *tịnh tiến* \rightarrow *chiếu tuyến tính* \rightarrow *hoàn nguyên*:

$$\begin{aligned} \mathbf{x}' &= \mathbf{x} - \ell \mathbf{1}_L && (\text{tịnh tiến: đưa toàn bộ giá trị trong cửa sổ về quanh mốc cuối } \ell) \\ \hat{\mathbf{y}}' &= W \mathbf{x}' + b \\ \hat{\mathbf{y}} &= \hat{\mathbf{y}}' + \ell \mathbf{1}_T \in \mathbb{R}^T && (\text{hoàn nguyên: cộng lại mốc } \ell \text{ để trả dự đoán về mức gốc}) \end{aligned}$$

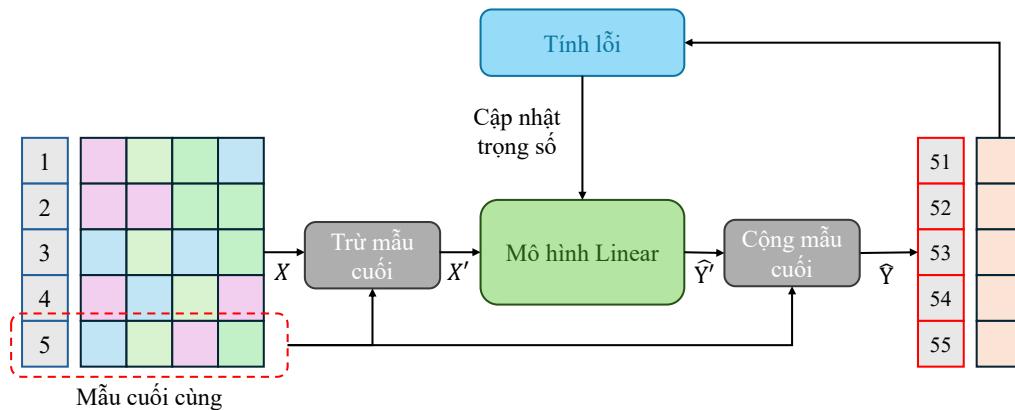
Huấn luyện (DMS). Tối ưu đồng thời cả T bước bằng MSE trên batch B :

$$\mathcal{L}_{\text{MSE}} = \frac{1}{BT} \sum_{n=1}^B \sum_{t=1}^T (\hat{y}_t - y_t)^2.$$

Trong đó (huấn luyện):

- T : số bước dự đoán (horizon); B : kích thước batch.
- \hat{y}_t, y_t : giá trị dự đoán và giá trị thực tại bước tương lai t .
- MSE: trung bình sai số bình phương trên toàn bộ T bước và B mẫu (tối ưu trực tiếp, không cuộn từng bước).

Độ phức tạp. Như Linear: tham số $\approx T \times L$ (cộng T bias); chi phí $\propto B \times T \times L$.



Hình 37: Quy trình chạy và huấn luyện của mô hình LTSF-NLinear.



Hình 38: Khi chúng ta biến đổi mốc cục bộ giá trị cuối cùng, mô hình Linear sẽ cần phải học ít thông tin hơn để đưa ra kết quả.

29.5 Mô hình LTSF-Linear: DLinear

Cơ chế: DLinear (đơn biến) phân rã chuỗi thành **trend** và **seasonal/residual**, sau đó *chiều tuyến* riêng cho từng phần và *cộng lại*. Cách làm này phù hợp khi dữ liệu có trend và/hoặc seasonal rõ rệt.

Ví dụ về MA_m(·) (moving average-cách tạo trend):

Giả sử $m=3$ với xử lý rìa kiểu **replicate** (tạm dịch là: lặp biến). Với $\mathbf{x} = [10, 12, 11, 13]$:

$$\text{MA}_3(\mathbf{x}) = \left[\frac{10+10+12}{3}, \frac{10+12+11}{3}, \frac{12+11+13}{3}, \frac{11+13+13}{3} \right] = [10.67, 11.00, 12.00, 12.33].$$

Chuỗi trên đóng vai trò *trend*; phần *seasonal/residual* là $\mathbf{x} - \text{MA}_3(\mathbf{x})$.

Thuật ngữ:

- **trend:** thành phần thay đổi chậm theo thời gian, phản ánh xu hướng dài hạn của chuỗi.
- **seasonal/residual:** phần còn lại sau khi bỏ trend; thường chứa nhịp lặp theo chu kỳ và dao động nhanh.

Thiết lập dữ liệu. Với chuỗi $\{x_t\}$, tại thời điểm t tạo

$$\mathbf{x} = [x_{t-L+1}, \dots, x_t] \in \mathbb{R}^L, \quad \mathbf{y} = [x_{t+1}, \dots, x_{t+T}] \in \mathbb{R}^T.$$

Trong đó (thiết lập):

- \mathbf{x} : input quá khứ dài L ; \mathbf{y} : mục tiêu T bước tương lai.
- $\mathbf{1}_L, \mathbf{1}_T$: vectơ toàn 1 kích thước L và T .

Công thức. Chọn bậc làm mượt m (ví dụ $m=5$). Phân rã bằng moving average, dự đoán tuyến tính theo từng phần, rồi cộng:

$$\begin{aligned} x_t &= \text{MA}_m(x) && \text{trend (moving average của } x) \\ x_s &= x - x_t && \text{seasonal/residual (phần còn lại)} \\ \hat{y}_t &= W_t x_t + b_t && \text{linear head cho trend } (W_t \in \mathbb{R}^{T \times L}, b_t \in \mathbb{R}^T) \\ \hat{y}_s &= W_s x_s + b_s && \text{linear head cho seasonal/residual } (W_s \in \mathbb{R}^{T \times L}, b_s \in \mathbb{R}^T) \\ \hat{y} &= \hat{y}_{\text{trend}} + \hat{y}_{\text{seasonal}} \in \mathbb{R}^T && \text{tổng hợp (forecast cuối cùng)} \end{aligned}$$

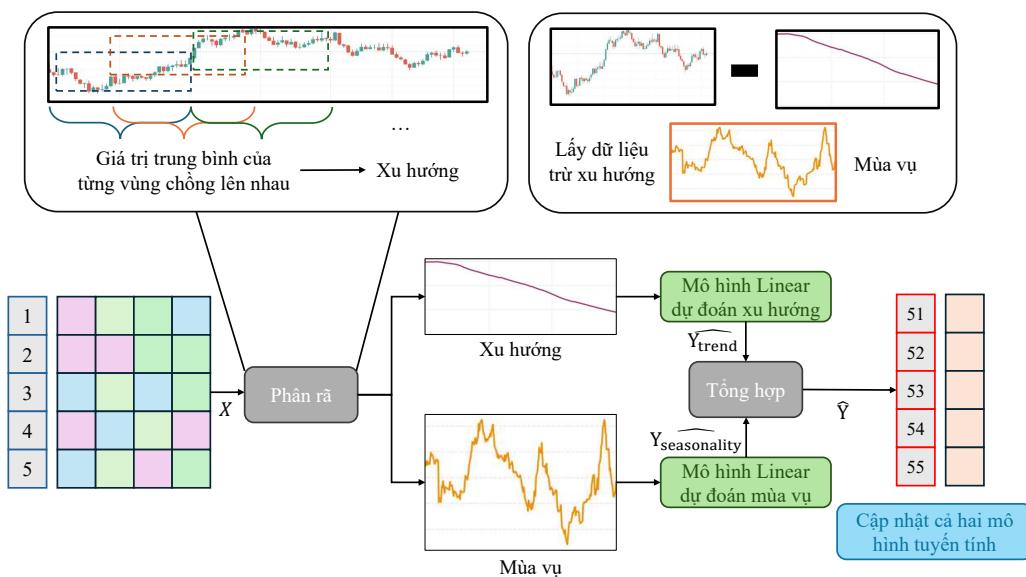
Huấn luyện (DMS). Tối ưu đồng thời cả T bước bằng MSE trên batch B :

$$\mathcal{L}_{\text{MSE}} = \frac{1}{BT} \sum_{n=1}^B \sum_{t=1}^T (\hat{y}_t - y_t)^2.$$

Trong đó (huấn luyện):

- T : số bước dự đoán (horizon); B : kích thước batch.
- \hat{y}_t, y_t : giá trị dự đoán và thực tại bước t .
- MSE: trung bình sai số bình phương trên toàn bộ T bước và B mẫu (tối ưu trực tiếp, không cuộn từng bước).

Độ phức tạp. Hai đầu tuyến tính nên tham số $\approx 2(T \times L)$ (cộng $2T$ bias); chi phí $\propto B \times T \times L$ (nhân đôi hằng số do hai nhánh), vẫn rất nhẹ.



Hình 39: Quy trình chạy và huấn luyện của mô hình LTSF-DLinear.

i T

Trong DLinear, tham số m biểu thị độ dài cửa sổ trung bình trượt (moving average window length), tức số điểm dữ liệu được lấy trung bình để ước lượng xu hướng $\text{Trend}(x_t)$. Tham số này xác định mức độ làm mượt của *low-pass filter* — bộ lọc cho phép thành phần tần số thấp (xu hướng dài hạn) đi qua và loại bỏ thành phần tần số cao (đao động ngắn hạn). Khi m lớn, xu hướng trở nên trơn hơn nhưng làm giảm chi tiết chu kỳ; ngược lại, m nhỏ giúp phản ứng nhanh hơn với biến động ngắn hạn. DLinear phù hợp với chuỗi có xu hướng hoặc chu kỳ rõ, trong khi NLinear hiệu quả hơn với dữ liệu có *level shift*, tức là các điểm thời gian mà giá trị kỳ vọng (hoặc trung bình cục bộ) của chuỗi thay đổi đột ngột.

29.6 Xây dựng các mô hình LTSF-Linear từ đầu (from scratch)

Để minh họa rõ cơ chế hoạt động, ta xây dựng lại ba mô hình **Linear**, **NLinear** và **DLinear** với phiên bản đơn giản, dễ hiểu hơn từ đầu.

```

1 import torch
2 import torch.nn as nn
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import pandas as pd
6
7 data = torch.tensor([[100.0, 102.0, 98.0, 105.0, 103.0]]) # [1, 5]
8 print(f"Input: {data.squeeze().numpy()}")
9
10 target = torch.tensor([[101.0, 104.0, 106.0]]) # [1, 3]
11 print(f"Target: {target.squeeze().numpy()}")

```

Ở bước này, bạn đã có dữ liệu đầu vào $\mathbf{x} \in \mathbb{R}^{1 \times 5}$ (5 bước quá khứ) và mục tiêu $\mathbf{y} \in \mathbb{R}^{1 \times 3}$ (3 bước cần dự báo). Ta sẽ dùng chúng để kiểm tra mô hình ở các bước sau.

Bước 2: Xây dựng mô hình Linear với trọng số cố định

Code Exercise

```

1 class Linear(nn.Module):
2     def __init__(self):
3         super().__init__()
4         self.linear = nn.Linear(5, 3)
5         # Set weights cụ thể
6         with torch.no_grad():
7             self.linear.weight.data = torch.tensor([
8                 [0.2, 0.3, -0.1, 0.4, 0.2], # Output 1
9                 [0.1, 0.2, 0.3, 0.3, 0.1], # Output 2
10                [0.0, 0.1, 0.2, 0.4, 0.3]   # Output 3
11            ])
12         self.linear.bias.data = torch.tensor([1.0, 2.0, 3.0])

```

```
13
14     def forward(self, x):
15         return ***Your Code Here***
16
17 # Test Linear
18 linear_model = Linear()
19 linear_out = ***Your Code Here***
20
21 print("LINEAR MODEL:")
22 print(f"    Weights: {linear_model.linear.weight.data.numpy()}")
23 print(f"    Bias: {linear_model.linear.bias.data.numpy()}")
24 print(f"    Input: {data.squeeze().numpy()}")
25 print(f"    Output: {linear_out.squeeze().detach().numpy()}")
26
27 # Manual calculation để verify
28 print("    Manual calc:")
29 for i in range(3):
30     w = linear_model.linear.weight.data[i]
31     b = linear_model.linear.bias.data[i]
32     result = torch.sum(w * data.squeeze()) + b
33     print(f"        Out{i+1}: {result:.3f}")
```

Thực hành điền code vào các vị trí *****Your Code Here***** để hoàn thành các yêu cầu sau:

- Hoàn thiện hàm `forward` để đưa đầu vào qua lớp `nn.Linear` và nhận đầu ra có kích thước phù hợp.
- Gọi mô hình với `data` để lấy `linear_out`.
- Đổi chiều Output với phần tính tay (Manual calculation) để kiểm tra công thức tuyến tính $\hat{y}_i = \sum_j W_{ij}x_j + b_i$.

Bước 3: Xây dựng mô hình DLinear với trọng số cố định

Code Exercise

```
1 class DLinear(nn.Module):
2     def __init__(self):
3         super().__init__()
4         self.linear_trend = nn.Linear(5, 3)
5         self.linear_seasonal = nn.Linear(5, 3)
6
7         # Set weights cụ thể cho trend
8         with torch.no_grad():
9             self.linear_trend.weight.data = torch.tensor([
10                 [0.3, 0.3, 0.2, 0.1, 0.1],
11                 [0.2, 0.3, 0.3, 0.1, 0.1],
12                 [0.1, 0.2, 0.3, 0.3, 0.1]
13             ])
14             self.linear_trend.bias.data = torch.tensor([0.5, 1.0, 1.
15                                             5])
16
17         # Set weights cho seasonal
18         self.linear_seasonal.weight.data = torch.tensor([
19             [0.1, -0.2, 0.3, -0.1, 0.0],
20             [0.0, 0.1, -0.2, 0.2, 0.1],
21             [-0.1, 0.0, 0.1, 0.1, 0.2]
22         ])
23         self.linear_seasonal.bias.data = torch.tensor([0.2, 0.3,
24                                             0.1])
25
26     def decompose(self, x):
27         # Simple moving average (window=3) - replicate padding
28         x_np = x.squeeze().numpy()
29         trend = []
30         for i in range(len(x_np)):
31             if i == 0:
32                 ***Your Code Here***
33             elif i == len(x_np) - 1:
34                 ***Your Code Here***
35             else:
36                 ***Your Code Here***
37             trend.append(avg)
38
39         trend_tensor = torch.tensor(trend).unsqueeze(0)
40         seasonal = ***Your Code Here***
```

```

39         return trend_tensor, seasonal
40
41     def forward(self, x):
42         trend, seasonal = ***Your Code Here***
43         trend_pred = self.linear_trend(trend)
44         seasonal_pred = self.linear_seasonal(seasonal)
45         return ***Your Code Here***
46 # Test DLinear
47 dlinear_model = DLinear()
48 dlinear_out = dlinear_model(data)
49
50 print("DLINEAR MODEL:")
51 trend, seasonal = dlinear_model.decompose(data)
52 print(f"    Input: {data.squeeze().numpy()}")
53 print(f"    Trend: {trend.squeeze().numpy()}")
54 print(f"    Seasonal: {seasonal.squeeze().numpy()}")
55 print(f"    Output: {dlinear_out.squeeze().detach().numpy()}")
56
57 # Show component predictions
58 trend_pred = dlinear_model.linear_trend(trend)
59 seasonal_pred = dlinear_model.linear_seasonal(seasonal)
60 print(f"    Trend pred: {trend_pred.squeeze().detach().numpy()}")
61 print(f"    Seasonal pred: {seasonal_pred.squeeze().detach().numpy()}")
62

```

Thực hành điền code vào các vị trí ***Your Code Here*** để hoàn thành các yêu cầu sau:

- Trong `decompose`: áp dụng trung bình trượt cửa sổ 3 điểm với replicate padding ở hai đầu để tạo dãy trend; sau đó lấy tín hiệu gốc trừ trend tại từng vị trí để thu được seasonal.
- Trong `forward`: gọi `decompose` để lấy trend và seasonal; đưa trend qua `linear_trend`, đưa seasonal qua `linear_seasonal`; cộng hai kết quả để tạo đầu ra cuối cùng.
- Chạy khôi kiểm thử để quan sát lần lượt: Input, trend, seasonal, dự đoán từ nhánh trend, dự đoán từ nhánh seasonal và Output tổng hợp.

Mục tiêu học tập:

- Hiểu quy trình tách chuỗi bằng thao tác làm mượt rồi trừ: trước hết lấy trend bằng trung bình mượt, sau đó trừ trend khỏi tín hiệu gốc để có phần còn lại.
- Nắm cách hai nhánh tuyến tính xử lý riêng hai phần và được cộng lại để tạo dự báo, qua đó thấy được tính tách biệt và khả năng diễn giải của DLinear.

Bước 4: Xây dựng mô hình NLinear với trọng số cố định

Code Exercise

```

1 class NLinear(nn.Module):
2     def __init__(self):
3         super().__init__()
4         self.linear = nn.Linear(5, 3)
5         # Set weights cụ thể
6         with torch.no_grad():
7             self.linear.weight.data = torch.tensor([
8                 [0.1, 0.2, 0.1, 0.3, 0.3],
9                 [0.2, 0.1, 0.3, 0.2, 0.2],
10                [0.3, 0.2, 0.2, 0.2, 0.1]
11            ])
12         self.linear.bias.data = torch.tensor([0.5, 1.0, 2.0])
13
14     def forward(self, x):
15         # Normalize bằng last value
16         last_value = ***Your Code Here***
17         x_norm = ***Your Code Here***
18         pred_norm = ***Your Code Here***
19         pred = ***Your Code Here***
20         return pred
21
22 # Test NLinear
23 nlinear_model = NLinear()
24 nlinear_out = nlinear_model(data)
25
26 print("NLINEAR MODEL:")
27 last_val = data[:, -1:]
28 x_norm = data - last_val
29 pred_norm = nlinear_model.linear(x_norm)
30

```

```

31 print(f" Input: {data.squeeze().numpy()}")
32 print(f" Last value: {last_val.squeeze().item()}")
33 print(f" Normalized: {x_norm.squeeze().numpy()}")
34 print(f" Pred normalized: {pred_norm.squeeze().detach().numpy()}")
35 print(f" Final output: {nlinear_out.squeeze().detach().numpy()}")
36
37 # Manual verification
38 print(" Manual calc:")
39 for i in range(3):
40     w = nlinear_model.linear.weight.data[i]
41     b = nlinear_model.linear.bias.data[i]
42     norm_pred = torch.sum(w * x_norm.squeeze()) + b
43     final = norm_pred + last_val.squeeze()
44     print(f" Out{i+1}: {norm_pred:.3f} + {last_val.squeeze().item():.1f} = {final:.3f}")

```

Thực hành điền code vào các vị trí ***Your Code Here*** để hoàn thành các yêu cầu sau:

- Điền last_value là cột cuối của cửa sổ đầu vào theo trực thời gian.
- Chuẩn hoá đầu vào bằng cách trừ last_value khỏi toàn bộ dãy để thu được x_norm.
- Tính pred_norm bằng cách đưa x_norm qua lớp tuyến tính đã khai báo.
- Cộng lại mốc last_value vào pred_norm để có pred và trả về pred.

Mục tiêu học tập:

- Hiểu cơ chế tái căn tâm theo giá trị cuối cùng: mô hình học trên độ lệch so với mốc hiện tại, sau đó cộng mốc để quay về thang gốc.
- Quan sát chuỗi bước in ra: giá trị cuối, đầu vào đã chuẩn hoá, dự đoán trên không gian chuẩn hoá, và đầu ra cuối cùng.

29.7 Bài toán dự đoán giá cổ phiếu sử dụng LTSF-Linear Models

Trong phần này, ta áp dụng họ mô hình **LTSF-Linear** (Linear, NLinear, DLinear) cho bài toán **dự đoán giá đóng cửa cổ phiếu VIC trong 7 ngày tới**. Mỗi mô hình được huấn luyện với các độ dài cửa sổ quá khứ khác nhau — 7, 30, 120 và 480 ngày — nhằm đánh giá khả năng học và khai quát trên nhiều quy mô thời gian. Kết quả dự đoán sẽ được so sánh với nhau để kiểm chứng hiệu quả và độ ổn định của các phương pháp tuyến tính trong dự báo chuỗi tài chính.

Bước 1: Import thư viện và tải dữ liệu

```
1 !gdown 18J_Z8b-qMMj9wm5eGyQ-1nPS16PfRePK
2
3 import pandas as pd
4 import numpy as np
5 import matplotlib.pyplot as plt
6 from matplotlib.ticker import PercentFormatter
7 import matplotlib.dates as mdates
8
9 import seaborn as sns
10 import warnings
11 warnings.filterwarnings("ignore")
12
13 import torch
14 import torch.nn as nn
15 from torch.utils.data import Dataset, DataLoader
16 from sklearn.metrics import mean_squared_error,
17 mean_absolute_error, r2_score
18 from sklearn.preprocessing import StandardScaler
19
20 plt.style.use("seaborn-v0_8")
21 sns.set_palette("husl")
22 print(f"Device available: {'CUDA' if torch.cuda.is_available() else
23                                     'CPU'}")
24
25 # Đọc dữ liệu
26 df = pd.read_csv("VIC.csv")
27 print(f"Dataset shape: {df.shape}")
```

```

27 print(f"Date range: {df['time'].min()} to {df['time'].max()}")
28
29 # Xử lý và sắp xếp theo thời gian
30 df["time"] = pd.to_datetime(df["time"])
31 df = df.sort_values("time").reset_index(drop=True)
32
33 # Thông tin cơ bản
34 print("\nFirst few rows:")
35 display(df.head())
36
37 print("\nData types:")
38 print(df.dtypes)
39
40 print("\nBasic statistics:")
41 display(df.describe())

```

Trong bước đầu tiên, các thư viện cần thiết được import, bao gồm: `pandas`, `numpy`, `matplotlib`, `seaborn` cho xử lý và trực quan hóa dữ liệu; `torch` và `sklearn` cho xây dựng mô hình và đánh giá hiệu suất. Tập dữ liệu `VIC.csv` được tải về và đọc bằng `pandas.read_csv()`, sau đó cột `time` được chuyển sang kiểu `datetime` và sắp xếp theo thứ tự thời gian.

Cuối cùng, dữ liệu được kiểm tra sơ bộ qua kích thước tập, khoảng thời gian, kiểu dữ liệu và thống kê mô tả cơ bản — giúp xác nhận cấu trúc dữ liệu đầu vào trước khi xử lý và huấn luyện mô hình.

	time	open	high	low	close	volume	symbol
0	2020-08-03	77.33	78.31	75.56	77.87	164310	VIC
1	2020-08-04	78.84	78.84	77.69	78.22	229230	VIC
2	2020-08-05	78.22	78.67	75.38	77.33	434490	VIC
3	2020-08-06	78.22	78.40	77.42	77.78	332340	VIC
4	2020-08-07	77.87	78.40	77.60	77.78	182500	VIC

Hình 40: Vài samples của tập dữ liệu VIC

Bước 2: Khám phá và tiền xử lý dữ liệu

```

1 # Tạo essential features
2 print("1. Tạo essential features:")
3 df["daily_return"] = df["close"].pct_change()
4 df["close_log"] = np.log(df["close"])
5 print(" - daily_return: phần trăm thay đổi hàng ngày")
6 print(" - close_log: log giá đóng cửa")
7
8 # Kiểm tra missing values cho TẤT CẢ các cột
9 missing_info = df.isnull().sum()
10
11 # Chỉ xử lý giá trị khuyết cần thiết
12 if df["daily_return"].isnull().sum() > 0:
13     df["daily_return"].fillna(0, inplace=True)
14
15 # Kiểm tra lại sau khi xử lý
16 missing_after = df.isnull().sum()
17 missing_after

```

Trong bước này, hai đặc trưng cơ bản được tạo ra từ giá đóng cửa:

- `daily_return`: phần trăm thay đổi giá giữa hai ngày liên tiếp, phản ánh biến động ngắn hạn.
- `close_log`: logarit tự nhiên của giá đóng cửa, giúp giảm độ lệch và ổn định phân phối dữ liệu.

Sau đó, tập dữ liệu được kiểm tra để phát hiện giá trị khuyết (*missing values*) trên toàn bộ các cột. Nếu xuất hiện giá trị NaN ở cột `daily_return` (do phép tính phần trăm thay đổi tại dòng đầu tiên), ta thay thế bằng 0 để đảm bảo tính toàn vẹn dữ liệu trước khi đưa vào huấn luyện.

Việc kiểm tra lại sau khi xử lý giúp xác nhận rằng toàn bộ dữ liệu hiện đã hoàn chỉnh, sẵn sàng cho các bước tiền xử lý và tạo tập huấn luyện tiếp theo.

```

1 fig, axes = plt.subplots(2, 2, figsize=(16, 12))
2 fig.suptitle("VIC Stock Data Analysis - After Feature Creation",
              fontsize=16, fontweight="bold")

```

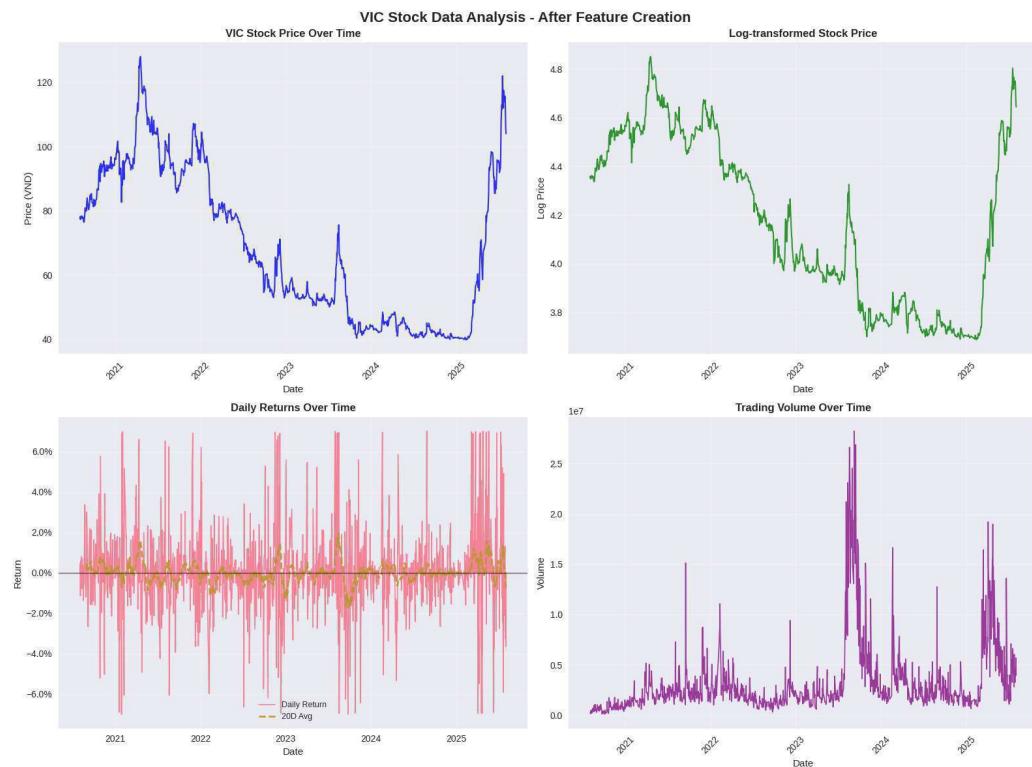
```
3 # 1) Stock Price Over Time
4 axes[0, 0].plot(df["time"], df["close"], linewidth=1.5, color="blue"
5 , alpha=0.8)
6 axes[0, 0].set_title("VIC Stock Price Over Time", fontsize=12,
7 fontweight="bold")
8 axes[0, 0].set_xlabel("Date")
9 axes[0, 0].set_ylabel("Price (VND)")
10 axes[0, 0].grid(True, alpha=0.3)
11 axes[0, 0].tick_params(axis="x", rotation=45)
12 # 2) Log-transformed Price
13 axes[0, 1].plot(df["time"], df["close_log"], linewidth=1.5, color="green",
14 alpha=0.8)
15 axes[0, 1].set_title("Log-transformed Stock Price", fontsize=12,
16 fontweight="bold")
17 axes[0, 1].set_xlabel("Date")
18 axes[0, 1].set_ylabel("Log Price")
19 axes[0, 1].grid(True, alpha=0.3)
20 axes[0, 1].tick_params(axis="x", rotation=45)
21 # 3) Daily Returns Over Time
22 ret_ts = df[["time", "daily_return"]].dropna().copy()
23 ret_ts["ret_ma20"] = ret_ts["daily_return"].rolling(20).mean()
24 axes[1, 0].plot(ret_ts["time"], ret_ts["daily_return"], linewidth=1.
25 2, alpha=0.85, label="Daily Return
26 ")
27 axes[1, 0].plot(ret_ts["time"], ret_ts["ret_ma20"], linewidth=2.0,
28 linestyle="--", alpha=0.95, label=
29 "20D Avg")
30 axes[1, 0].axhline(0, color="black", linewidth=1, alpha=0.6)
31 axes[1, 0].set_title("Daily Returns Over Time", fontsize=12,
32 fontweight="bold")
33 axes[1, 0].set_xlabel("Date")
34 axes[1, 0].set_ylabel("Return")
35 axes[1, 0].yaxis.set_major_formatter(PercentFormatter(1.0))
36 axes[1, 0].xaxis.set_major_locator(mdates.AutoDateLocator())
37 axes[1, 0].xaxis.set_major_formatter(mdates.ConciseDateFormatter(
38 mdates.AutoDateLocator()))
39 axes[1, 0].tick_params(axis="x", rotation=0)
40 axes[1, 0].grid(True, alpha=0.3)
41 axes[1, 0].legend(fontsize=9)
```

```
37 # 4) Volume Over Time
38 axes[1, 1].plot(df["time"], df["volume"], linewidth=1.2, color="purple", alpha=0.75)
39 axes[1, 1].set_title("Trading Volume Over Time", fontsize=12, fontweight="bold")
40 axes[1, 1].set_xlabel("Date")
41 axes[1, 1].set_ylabel("Volume")
42 axes[1, 1].grid(True, alpha=0.3)
43 axes[1, 1].tick_params(axis="x", rotation=45)
44
45 plt.tight_layout()
46 plt.show()
```

Sau khi xử lý và bổ sung đặc trưng, dữ liệu được trực quan hóa để quan sát xu hướng và cấu trúc biến động theo thời gian:

- (1) **Stock Price Over Time:** biểu diễn giá đóng cửa theo thời gian, thể hiện biến động tổng thể của cổ phiếu VIC.
- (2) **Log-transformed Price:** đồ thị giá đã logarit hóa, giúp làm phẳng các dao động lớn, dễ quan sát xu hướng dài hạn.
- (3) **Daily Returns Over Time:** hiển thị phần trăm thay đổi giá hằng ngày cùng đường trung bình trượt 20 ngày, giúp nhận diện giai đoạn biến động mạnh hay ổn định.
- (4) **Trading Volume Over Time:** biểu đồ khối lượng giao dịch, phản ánh mức độ thanh khoản và sự quan tâm của thị trường.

Những biểu đồ này cung cấp cái nhìn trực quan đầu tiên về hành vi của cổ phiếu VIC — bao gồm xu hướng giá, độ biến động và mối liên hệ giữa giá và khối lượng giao dịch — là nền tảng quan trọng trước khi trích xuất đặc trưng đầu vào cho mô hình dự báo.



Hình 41: Các biểu đồ cung cấp cái nhìn trực quan về hành vi của cổ phiếu VIC

Bước 3: Chuẩn bị dữ liệu chuỗi thời gian đơn biến cho mô hình dự báo

```

1 class UnivariateTimeSeriesDataset(Dataset):
2     """Dataset for univariate time series forecasting - 7 days ahead
3
4     def __init__(self, data, seq_len, pred_len=7, target_col="close",
5                  normalize=False):
6         self.data = data.dropna().reset_index(drop=True)
7         self.seq_len = seq_len
8         self.pred_len = pred_len
9         self.target_col = target_col
10        self.normalize = normalize
11
12        self.series = self.data[target_col].values

```

```
11
12     if normalize:
13         self.mean = np.mean(self.series)
14         self.std = np.std(self.series)
15
16     def __len__(self):
17         return len(self.series) - self.seq_len - self.pred_len + 1
18
19     def __getitem__(self, idx):
20         x = self.series[idx:idx+self.seq_len].copy()
21         y = self.series[idx+self.seq_len:idx+self.seq_len+self.
22                         pred_len].copy()
23
24         if self.normalize:
25             x = (x - self.mean) / self.std
26             y = (y - self.mean) / self.std
27
28         return torch.FloatTensor(x), torch.FloatTensor(y)
29
30     def denormalize(self, normalized_values):
31         if not self.normalize:
32             return normalized_values
33         return normalized_values * self.std + self.mean
34
35     def create_univariate_datasets(df, seq_lengths, pred_len=7,
36                                     target_col="close"):
37         """Create univariate datasets for different sequence lengths"""
38         datasets = {}
39
40         for seq_len in seq_lengths:
41             dataset = UnivariateTimeSeriesDataset(
42                 data=df, seq_len=seq_len, pred_len=pred_len,
43                 target_col=target_col, normalize=False
44             )
45             datasets[f"{seq_len}d"] = dataset
46
47
48     # Create datasets for 7-day prediction with new input lengths
49     seq_lengths = [7, 30, 120, 480] # Input sequence lengths
50     pred_len = 7 # Predict 7 days ahead
51
52
```

```

53 # Create datasets using log prices for better stability
54 datasets = create_univariate_datasets(df, seq_lengths, pred_len, "
55                                     close_log")
56 print(f"Created datasets for 7-day prediction with input lengths: {"
57           seq_lengths}")
58 for name, dataset in datasets.items():
59     print(f"- {name}: {len(dataset)} samples")
60 print("Using log-transformed prices for stability.")

```

Lớp `UnivariateTimeSeriesDataset` được định nghĩa để tạo tập dữ liệu dự báo đơn biến. Mỗi mẫu gồm hai phần: $\mathbf{x} \in \mathbb{R}^L$ (chuỗi đầu vào gồm L ngày trước) và $\mathbf{y} \in \mathbb{R}^T$ (giá trị mục tiêu cho T ngày tiếp theo). Ở đây, dữ liệu sử dụng giá đóng cửa logarit (`close_log`) để tăng tính ổn định khi huấn luyện mô hình.

Các tập dữ liệu được tạo cho bốn độ dài của sổ đầu vào: **7, 30, 120** và **480** ngày, với mục tiêu dự báo **7 ngày tiếp theo**.

```

1 def create_time_based_splits(dataset, train_ratio=0.7, val_ratio=0.
15):
2     """Create time-based splits for time series data"""
3     total_len = len(dataset)
4     train_len = int(total_len * train_ratio)
5     val_len = int(total_len * val_ratio)
6
7     # Time-based splits (no shuffling to preserve temporal order)
8     train_indices = list(range(0, train_len))
9     val_indices = list(range(train_len, train_len + val_len))
10    test_indices = list(range(train_len + val_len, total_len))
11
12    train_dataset = torch.utils.data.Subset(dataset, train_indices)
13    val_dataset = torch.utils.data.Subset(dataset, val_indices)
14    test_dataset = torch.utils.data.Subset(dataset, test_indices)
15
16    return train_dataset, val_dataset, test_dataset
17
18
19 # Create splits for all sequence lengths

```

```
20 data_splits = []
21
22 print("Creating time-based data splits:")
23 print("- Train: 70%, Validation: 15%, Test: 15%")
24 print("- Temporal order preserved (no shuffling)")
25
26 for seq_name, dataset in datasets.items():
27     train, val, test = create_time_based_splits(dataset)
28     data_splits[seq_name] = {
29         "train": train,
30         "val": val,
31         "test": test
32     }
33
34 print(f"\nTotal datasets with splits: {len(data_splits)}")
35
36 # Verify split integrity
37 print(f"\n==== SPLIT VERIFICATION ====")
38 for seq_name, splits in data_splits.items():
39     total_samples = len(splits["train"]) + len(splits["val"]) + len(
40                                     splits["test"])
41     original_samples = len(datasets[seq_name])
42
43     print(f"{seq_name}: {total_samples}/{original_samples} samples"
44           if total_samples == original_samples
45           else f"{seq_name}: ERROR - {total_samples}/{{
46                                     original_samples}}")
```

Sau khi tạo tập dữ liệu, dữ liệu được chia theo tỷ lệ thời gian cố định: **70%** cho huấn luyện, **15%** cho xác thực và **15%** cho kiểm thử. Không áp dụng xáo trộn (*no shuffling*) để giữ nguyên thứ tự thời gian. Cách chia này đảm bảo tính liên tục của chuỗi và phản ánh đúng bối cảnh dự báo thực tế, đồng thời giúp đánh giá khả năng mô hình tổng quát hoá qua các giai đoạn khác nhau.

```

Creating time-based data splits:
- Train: 70%, Validation: 15%, Test: 15%
- Temporal order preserved (no shuffling)

Total datasets with splits: 4

==== SPLIT VERIFICATION ====
7d: 1236/1236 samples √
30d: 1213/1213 samples √
120d: 1123/1123 samples √
480d: 763/763 samples √

Data splits created successfully!

```

Hình 42: Tổng quan dữ liệu sau khi được chia thành các tập

Bước 4: Xây dựng và khởi tạo các mô hình LTSF-Linear

Code Exercise

```

1 # LTSF-Linear Models Implementation for Univariate Time Series (7-
   day prediction)
2 class Linear(nn.Module):
3     """Simple Linear model for univariate time series forecasting"""
4     def __init__(self, seq_len, pred_len=7):
5         super(Linear, self).__init__()
6         self.seq_len = seq_len
7         self.pred_len = pred_len
8         self.linear = ***Your Code Here***
9
10    def forward(self, x):
11        # x: [batch_size, seq_len] - historical prices
12        # Output: [batch_size, pred_len] - 7-day future predictions
13        return ***Your Code Here***
14
15 class DLinear(nn.Module):
16     """Decomposition Linear for univariate time series - handles
       trend and seasonality"""
17     def __init__(self, seq_len, pred_len=7, moving_avg=5):
18         super(DLinear, self).__init__()
19         self.seq_len = seq_len
20         self.pred_len = pred_len
21         self.moving_avg = min(moving_avg, seq_len - 1)
22
23         # Linear layers for trend and seasonal components
24         self.linear_trend = ***Your Code Here***

```

```
25     self.linear_seasonal = ***Your Code Here***  
26  
27     # Create moving average kernel for trend extraction  
28     self.register_buffer('avg_kernel', torch.ones(1, 1, self.  
29                           moving_avg) / self.moving_avg)  
30  
31     def decompose(self, x):  
32         """Decompose series into trend and seasonal components"""  
33         batch_size, seq_len = x.shape  
34         x_reshaped = x.unsqueeze(1)  
35  
36         # Apply moving average for trend  
37         padding = self.moving_avg // 2  
38         x_padded = torch.nn.functional.pad(x_reshaped, (padding,  
39                                              padding), mode='replicate')  
40         trend = torch.nn.functional.conv1d(x_padded, self.avg_kernel  
41                                           , padding=0)  
42         trend = trend.squeeze(1)  
43  
44         # Ensure trend has same length as input  
45         if trend.shape[1] != seq_len:  
46             trend = torch.nn.functional.interpolate(  
47                 trend.unsqueeze(1), size=seq_len, mode='linear',  
48                               align_corners=False  
49             ).squeeze(1)  
50  
51         seasonal = ***Your Code Here***  
52         return ***Your Code Here***  
53  
54     def forward(self, x):  
55         trend, seasonal = ***Your Code Here***  
56         trend_pred = ***Your Code Here***  
57         seasonal_pred = ***Your Code Here***  
58         return ***Your Code Here***  
59  
60 class NLinear(nn.Module):  
61     """Normalized Linear for univariate time series - handles  
62         distribution shift"""  
63     def __init__(self, seq_len, pred_len=7):  
64         super(NLinear, self).__init__()  
65         self.seq_len = seq_len  
66         self.pred_len = pred_len  
67         self.linear = ***Your Code Here***
```

```

64     def forward(self, x):
65         # Normalize by subtracting last value
66         last_value = ***Your Code Here***
67         x_normalized = ***Your Code Here***
68         pred_normalized = ***Your Code Here***
69         pred = ***Your Code Here***
70         return ***Your Code Here***
71
72 print("LTSF-Linear models implemented for 7-day univariate
      forecasting.")

```

Thực hành điền code vào các vị trí ***Your Code Here*** để hoàn thành các yêu cầu sau:

- Linear: khởi tạo lớp $[seq_len] \rightarrow [pred_len]$; trong forward, đưa x qua lớp đó để nhận đầu ra $[batch, pred_len]$.
- DLinear: tạo hai lớp cho trend và seasonal; trong decompose, tính seasonal bằng $x - trend$ và trả về (trend, seasonal); trong forward, dự đoán từng nhánh rồi cộng.
- NLinear: lấy $last_value = x[:, -1 :]$, chuẩn hoá $x - last_value$, dự đoán trên không gian chuẩn hoá, cộng lại mốc để ra kết quả.

Mục tiêu học tập:

- Nắm đường đi dữ liệu và kích thước đầu vào–đầu ra ở cả ba biến thể.
- Hiểu cơ chế tách–cộng của DLinear và tái căn tâm theo giá trị cuối của NLinear.
- Giữ nguyên kích thước batch và trực thời gian; tránh dùng `squeeze()` gây mất chiều.

Sau đó ta tạo cấu hình và khởi tạo mô hình:

```

1 # Model Configuration and Creation for 7-day Prediction
2 horizon_configs = {

```

```

3     '7d': {'seq_len': 7, 'pred_len': 7},
4     '30d': {'seq_len': 30, 'pred_len': 7},
5     '120d': {'seq_len': 120, 'pred_len': 7},
6     '480d': {'seq_len': 480, 'pred_len': 7}
7 }
8
9 # Create model instances for each combination
10 model_configs = {
11     'Linear': {},
12     'DLinear': {},
13     'NLinear': {}
14 }
15
16 print("Creating model instances for 7-day prediction:")
17 for horizon, config in horizon_configs.items():
18     seq_len = config['seq_len']
19     pred_len = config['pred_len']
20
21     # Create model instances
22     model_configs['Linear'][horizon] = {
23         'model': Linear(seq_len, pred_len),
24         'seq_len': seq_len,
25         'pred_len': pred_len
26     }
27
28     model_configs['DLinear'][horizon] = {
29         'model': DLinear(seq_len, pred_len),
30         'seq_len': seq_len,
31         'pred_len': pred_len
32     }
33
34     model_configs['NLinear'][horizon] = {
35         'model': NLinear(seq_len, pred_len),
36         'seq_len': seq_len,
37         'pred_len': pred_len
38     }
39
40     print(f" {horizon}: seq_len={seq_len}, pred_len={pred_len}")

```

Đoạn mã trên định nghĩa bốn cấu hình độ dài của sổ đầu vào (7, 30, 120, 480 ngày) cùng chung đích dự báo 7 ngày, rồi khởi tạo ba mô hình Linear, DLinear và NLinear cho từng cấu hình và lưu vào `model_configs` để phục

vụ huấn luyện và so sánh hiệu suất.

Bước 5: Huấn luyện và trực quan hóa trực loss

Đầu tiên ta tạo dataset chuẩn hoá dùng chung một scaler, chia tập theo thời gian và đánh giá trên thang giá gốc.

```
1 # Training Configuration with Data Normalization for 7-day
2 # Prediction
3 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
4
5 # Create normalized dataset class
6 class NormalizedDataset(Dataset):
7     def __init__(self, original_dataset, scaler=None):
8         self.original_dataset = original_dataset
9
10    # Fit scaler if not provided
11    if scaler is None:
12        all_data = []
13        for i in range(len(original_dataset)):
14            x, y = original_dataset[i]
15            all_data.extend(x.numpy())
16            all_data.extend(y.numpy())
17
18        self.scaler = StandardScaler()
19        self.scaler.fit(np.array(all_data).reshape(-1, 1))
20    else:
21        self.scaler = scaler
22
23    def __len__(self):
24        return len(self.original_dataset)
25
26    def __getitem__(self, idx):
27        x, y = self.original_dataset[idx]
28
29        # Normalize features and targets
30        x_norm = self.scaler.transform(x.numpy().reshape(-1, 1)).
31                           flatten()
```

```
32         return torch.FloatTensor(x_norm), torch.FloatTensor(y_norm)
33
34     def denormalize(self, normalized_values):
35         """Convert normalized values back to original scale"""
36         return self.scaler.inverse_transform(normalized_values.
37                                             reshape(-1, 1)).flatten()
38
39 # Create normalized datasets
40 normalized_datasets = {}
41 scaler = None
42
43 for horizon, dataset in datasets.items():
44     normalized_dataset = NormalizedDataset(dataset, scaler)
45     if scaler is None:
46         scaler = normalized_dataset.scaler # Use same scaler for
47                                         all
48     normalized_datasets[horizon] = normalized_dataset
49
50 # Create normalized splits
51 normalized_data_splits = {}
52 for horizon in datasets.keys():
53     total_len = len(normalized_datasets[horizon])
54     train_len = int(total_len * 0.7)
55     val_len = int(total_len * 0.15)
56
57     train_indices = list(range(0, train_len))
58     val_indices = list(range(train_len, train_len + val_len))
59     test_indices = list(range(train_len + val_len, total_len))
60
61     normalized_data_splits[horizon] = {
62         'train': torch.utils.data.Subset(normalized_datasets[horizon],
63                                         train_indices),
64         'val': torch.utils.data.Subset(normalized_datasets[horizon],
65                                         val_indices),
66         'test': torch.utils.data.Subset(normalized_datasets[horizon],
67                                         test_indices)
68     }
69
70 # Updated evaluation function for normalized data
71 def evaluate_model_normalized(model, test_loader, scaler, device='
72                               cpu'):
73     """Evaluate model performance with denormalization"""
74     model.eval()
75     predictions = []
```

```

70     actuals = []
71
72     with torch.no_grad():
73         for batch_x, batch_y in test_loader:
74             batch_x, batch_y = batch_x.to(device), batch_y.to(device
75                                         )
76             outputs = model(batch_x)
77
78             # Denormalize predictions and actuals
79             for i in range(outputs.shape[0]):
80                 pred_denorm = scaler.inverse_transform(outputs[i].
81                                             cpu().numpy().reshape(-1, 1)).
82                                             flatten()
83                 actual_denorm = scaler.inverse_transform(batch_y[i].
84                                             cpu().numpy().reshape(-1, 1)).
85                                             flatten()
86                 predictions.extend(pred_denorm)
87                 actuals.extend(actual_denorm)
88
89             predictions = np.array(predictions)
90             actuals = np.array(actuals)
91
92             # Calculate metrics on original scale
93             mse = mean_squared_error(actuals, predictions)
94             mae = mean_absolute_error(actuals, predictions)
95             rmse = np.sqrt(mse)
96
97             try:
98                 r2 = r2_score(actuals, predictions)
99             except:
100                 r2 = -999
101
102             return {
103                 'mse': mse,
104                 'mae': mae,
105                 'rmse': rmse,
106                 'r2': r2,
107                 'predictions': predictions,
108                 'actuals': actuals
109             }
110

```

Lớp NormalizedDataset chuẩn hoá cả đầu vào và mục tiêu bằng cùng một StandardScaler dùng chung cho mọi horizon; tạo các tập train/val/test theo

thứ tự thời gian; hàm

`evaluate_model_normalized` suy luận trên dữ liệu chuẩn hoá, rồi đảo chuẩn hoá dự đoán và nhân để tính các chỉ số (MSE, MAE, RMSE, R²) trên thang giá gốc.

```
1 def train_model(model, train_loader, val_loader, num_epochs=50, lr=0
2                 .001, device="cpu"):
3     """Train one model and return epoch-wise train/val MSE (no
4         printing)."""
5     criterion = nn.MSELoss()
6     optimizer = torch.optim.Adam(model.parameters(), lr=lr)
7     scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size
8                      =20, gamma=0.5)
9
10    model.to(device)
11    train_losses, val_losses = [], []
12
13    for _ in range(num_epochs):
14        # Train
15        model.train()
16        epoch_train = 0.0
17        for batch_x, batch_y in train_loader:
18            batch_x, batch_y = batch_x.to(device), batch_y.to(device
19                                )
20            optimizer.zero_grad()
21            preds = model(batch_x)
22            loss = criterion(preds, batch_y)
23            loss.backward()
24            optimizer.step()
25            epoch_train += loss.item()
26            train_losses.append(epoch_train / max(1, len(train_loader)))
27
28        # Validate
29        model.eval()
30        epoch_val = 0.0
31        with torch.no_grad():
32            for batch_x, batch_y in val_loader:
33                batch_x, batch_y = batch_x.to(device), batch_y.to(
34                                device)
35                preds = model(batch_x)
36                epoch_val += criterion(preds, batch_y).item()
37                val_losses.append(epoch_val / max(1, len(val_loader)))
```

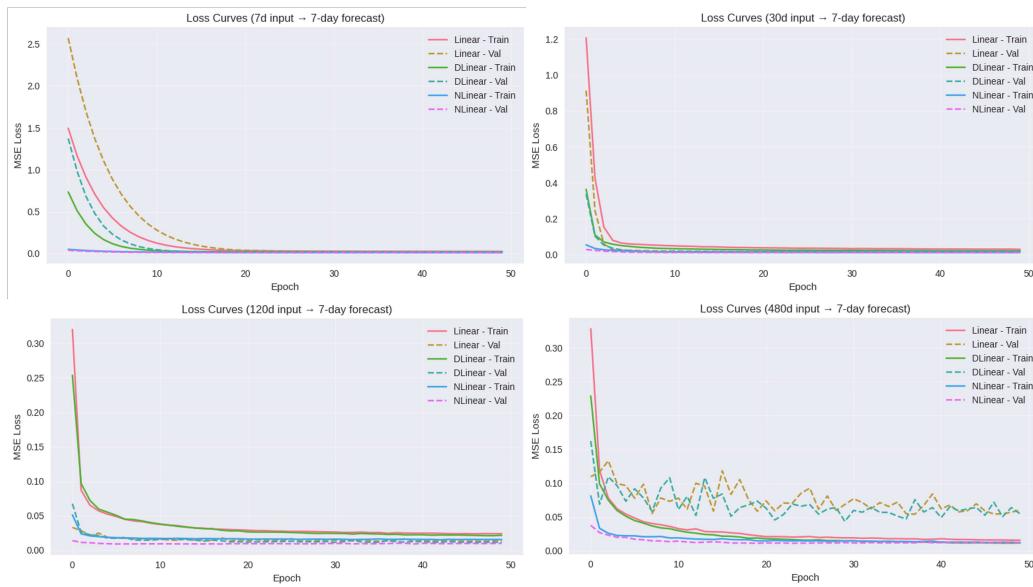
```
34     scheduler.step()
35
36     return model, train_losses, val_losses
```

Hàm `train_model` huấn luyện mô hình và trả về lịch sử `train_losses` / `val_losses` cho việc vẽ đồ thị, *không* in log theo từng epoch.

```
1 import matplotlib.pyplot as plt
2
3 def plot_loss_curves(loss_history, horizons):
4     """Plot train/val losses for all models across horizons."""
5     for hz in horizons:
6         plt.figure(figsize=(8, 4.5))
7         for model_name in ["Linear", "DLinear", "NLinear"]:
8             if hz in loss_history.get(model_name, {}):
9                 tl = loss_history[model_name][hz]["train"]
10                vl = loss_history[model_name][hz]["val"]
11                plt.plot(tl, label=f"{model_name} - Train")
12                plt.plot(vl, linestyle="--", label=f"{model_name} - Val")
13            plt.title(f"Loss Curves ({hz} input --> 7-day forecast)")
14            plt.xlabel("Epoch")
15            plt.ylabel("MSE Loss")
16            plt.grid(True, alpha=0.3)
17            plt.legend()
18            plt.tight_layout()
19            plt.show()
20
21 batch_size = 32
22 num_epochs = 50
23 learning_rate = 0.001
24
25 results = {"Linear": {}, "DLinear": {}, "NLinear": {}}
26 trained_models = {"Linear": {}, "DLinear": {}, "NLinear": {}}
27 loss_history = {"Linear": {}, "DLinear": {}, "NLinear": {}}
28
29 for horizon in ["7d", "30d", "120d", "480d"]:
30     train_loader = DataLoader(
31         normalized_data_splits[horizon]["train"],
32         batch_size=batch_size, shuffle=True, drop_last=True
```

```
33     )
34     val_loader = DataLoader(
35         normalized_data_splits[horizon] ["val"],
36         batch_size=batch_size, shuffle=False, drop_last=True
37     )
38     test_loader = DataLoader(
39         normalized_data_splits[horizon] ["test"],
40         batch_size=batch_size, shuffle=False, drop_last=True
41     )
42
43     for model_name in ["Linear", "DLinear", "NLinear"]:
44         model = model_configs[model_name] [horizon] ["model"]
45         trained_model, tr_losses, va_losses = train_model(
46             model, train_loader, val_loader,
47             num_epochs=num_epochs, lr=learning_rate, device=device
48         )
49         loss_history[model_name] [horizon] = {"train": tr_losses, "
50                                         val": va_losses}
51         trained_models[model_name] [horizon] = trained_model
52
53         # (Tuỳ chọn) lưu kết quả test để tổng hợp cuối cùng, không
54         # in
55         test_results = evaluate_model_normalized(trained_model,
56                                                 test_loader, scaler, device)
57         results[model_name] [horizon] = test_results
58
59     # Chỉ trực quan hóa (không in log)
60     plot_loss_curves(loss_history, horizons=["7d", "30d", "120d", "480d"
61                                         ])
```

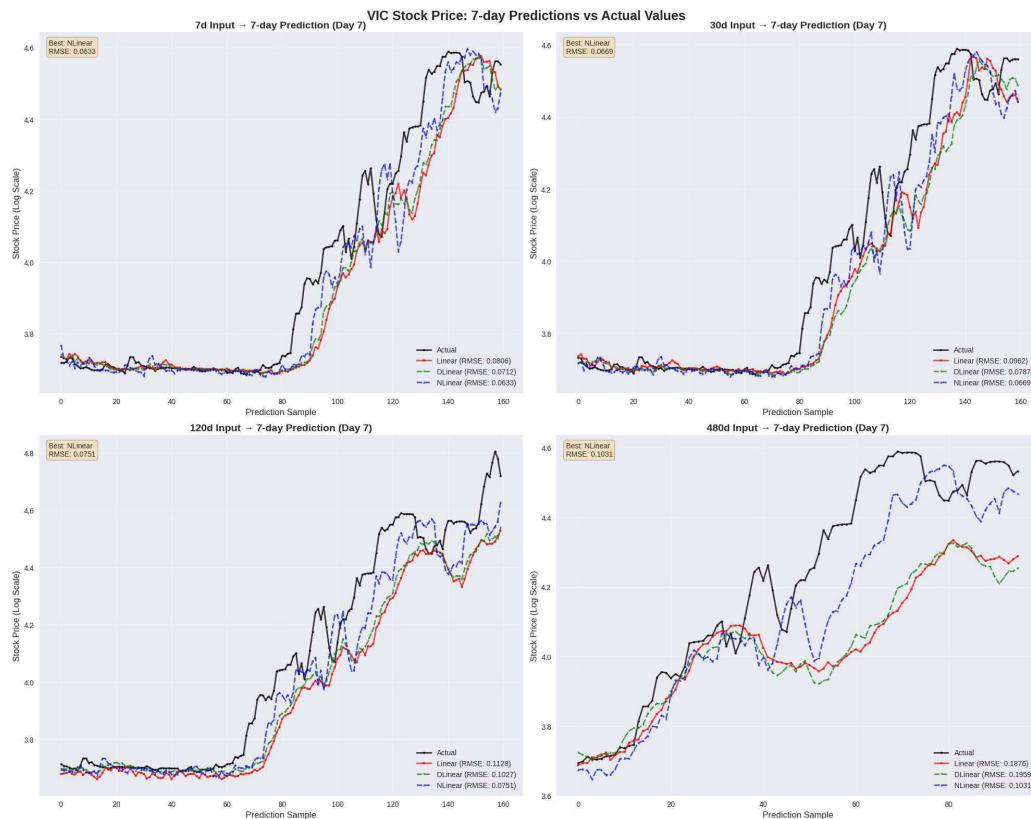
Khôi lệnh trên huấn luyện toàn bộ các mô hình **Linear**, **DLinear**, **NLinear** trên bốn độ dài cửa sổ đầu vào và chỉ vẽ đường học (train/val) cho từng cấu hình, giúp theo dõi hội tụ mà không cần in log chi tiết.



Hình 43: Trực quan hóa loss của 4 độ dài input khác nhau

Sau khi quá trình huấn luyện hoàn tất, các mô hình Linear, DLinear và NLinear được sử dụng để dự đoán giá đóng cửa cổ phiếu VIC trong 7 ngày kế tiếp, dựa trên dữ liệu đầu vào có độ dài lần lượt là 7, 30, 120 và 480 ngày.

Các biểu đồ dưới đây trực quan hóa kết quả dự đoán, thể hiện sự so khớp giữa giá *thực tế* và *giá dự đoán* của từng mô hình. Mỗi subplot tương ứng với một độ dài cửa sổ đầu vào, giúp quan sát rõ mức độ ảnh hưởng của kích thước lịch sử đến độ chính xác của dự báo.



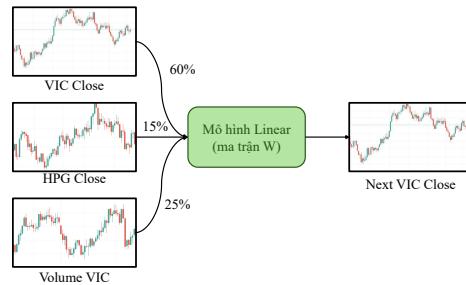
Hình 44: So sánh giữa giá thực tế và giá dự đoán từ các mô hình cho từng cửa sổ đầu vào.

29.8 Hướng cải tiến và mở rộng

Chúng ta đã thấy được sức mạnh của các mô hình LTSF-Linear. Với sự đơn giản, tốc độ và hiệu quả trong việc xử lý nhiều thông qua các cơ chế như chuẩn hóa (NLinear) hay phân rã (DLinear), chúng đóng vai trò là một baseline cực kỳ mạnh mẽ và dễ sử dụng. Tuy nhiên, chính sự đơn giản dựa trên nền tảng tuyến tính cũng là giới hạn của lớp mô hình này. Sau khi đã có một nền tảng vững chắc, đây là hai hướng cải tiến và mở rộng tự nhiên nhất mà bạn có thể khám phá.

29.8.1 Mở rộng sang bài toán đa biến

Hạn chế hiện tại: Toàn bộ project này đang tập trung vào bài toán đơn biến, tức là chỉ dùng lịch sử giá của chính cổ phiếu VIC để dự đoán tương lai của nó. Điều này bỏ qua một thực tế quan trọng trên thị trường tài chính: giá của một cổ phiếu hiếm khi di chuyển một mình. Nó chịu ảnh hưởng mạnh mẽ từ các yếu tố bên ngoài như chỉ số thị trường chung (VN-Index), giá của các cổ phiếu đầu ngành khác (ví dụ: HPG, FPT), hoặc thậm chí là các chỉ số kinh tế vĩ mô.



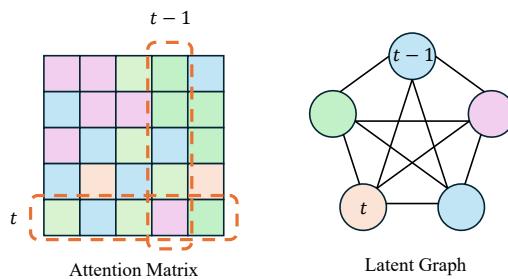
Hình 45: Với bài toán đa biến, mô hình sẽ quyết định được bao nhiêu phần trăm đóng góp mà mỗi giá trị đầu vào ảnh hưởng tới kết quả cuối cùng.

Hướng cải tiến: Hướng đi đầu tiên là **thu thập dữ liệu đa biến**, bổ sung thêm các chuỗi thời gian của các mã cổ phiếu liên quan, chỉ số VN-Index, hoặc các dữ liệu kinh tế khác. Tiếp theo, cần **điều chỉnh mô hình** bằng cách mở rộng các mô hình Linear, NLinear, và DLinear để chấp nhận đầu vào đa biến (ví dụ: input $\mathbf{x} \in \mathbb{R}^{L \times C}$ với C là số lượng biến), điều này đòi hỏi phải thay đổi kiến trúc của lớp `nn.Linear` và cách xử lý dữ liệu. Một bài toán mà chúng ta phải giải quyết là làm thế nào để áp dụng cơ chế phân rã

của DLinear hay chuẩn hóa của NLinear một cách hiệu quả trên nhiều biến cùng lúc. Đây là một câu hỏi mở và là một hướng nghiên cứu thú vị.

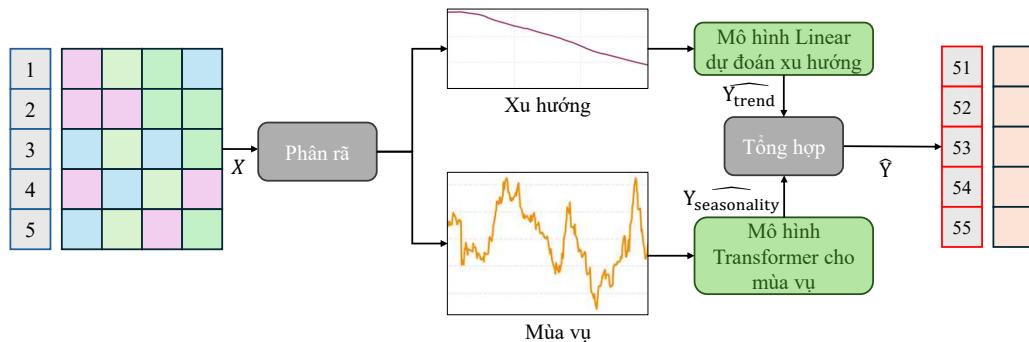
29.8.2 Hybrid model: kết hợp phân rã và transformer

Hạn chế hiện tại: Ở phần giới thiệu, chúng ta đã chỉ ra điểm yếu của Transformer là độ phức tạp tính toán và nguy cơ học vẹt nhiều. Ngược lại, điểm mạnh nhất của DLinear là khả năng tách tín hiệu thành xu hướng và mùa vụ. Vậy tại sao không kết hợp điểm mạnh của cả hai?



Hình 46: Với lớp mô hình Transformer, khi dự đoán tương lai chúng ta có thể hình lại toàn bộ quá khứ. Đồng thời, mô hình không cần chạy qua các giai đoạn để đến được tương lai.

Hướng cải tiến: Chúng ta có thể xây dựng một mô hình lai ghép (hybrid). Bước 1 là phân rã, sử dụng moving average từ DLinear để tách chuỗi đầu vào \mathbf{x} thành hai thành phần: \mathbf{x}_{trend} và $\mathbf{x}_{seasonal}$. Bước 2 là xử lý hai luồng dữ liệu của chúng ta có một cách riêng biệt. Với nhánh trend (\mathbf{x}_{trend}), thành phần này thường rất mượt và dễ đoán nên chúng ta chỉ cần dùng một mô hình **Linear** đơn giản để dự đoán nó. Ngược lại, nhánh seasonal ($\mathbf{x}_{seasonal}$) phức tạp hơn, chứa các dao động và mẫu khó nắm bắt; đây chính là nơi một mô hình **Transformer** (ví dụ: Autoformer hoặc một Transformer đơn giản) có thể phát huy thế mạnh để tìm ra các phụ thuộc phức tạp. Cuối cùng, bước 3 là tổng hợp, cộng kết quả dự đoán từ hai nhánh lại để ra được dự đoán cuối cùng.



Hình 47: Sơ đồ huấn luyện mô hình mẫu cho cách cải tiến hybrid nói trên. Chúng ta có thể sử dụng block Attention đơn giản hoặc toàn bộ mô hình phức tạp như Autoformer vào khu vực Transformer. Cách kết hợp vẫn giữ nguyên là phép cộng.

29.9 Câu hỏi trắc nghiệm

1. (Lý thuyết) Dự báo nhiều bước trực tiếp (Direct Multi-step, DMS). Phát biểu nào đúng về DMS trong các mô hình LTSF-Linear?
 - (a) Dự báo tuần tự từng bước và dùng đầu ra bước t làm đầu vào bước $t+1$ (recursive/rollout).
 - (b) Dự báo đồng thời T bước tương lai thông qua một ánh xạ duy nhất $\mathbb{R}^L \rightarrow \mathbb{R}^T$.
 - (c) Cần RNN/Transformer để dự báo được T bước.
 - (d) Luôn yêu cầu đổi tần suất (resampling) của chuỗi trước khi học.
2. (Lý thuyết) Trong DLinear, $MA_m(\cdot)$ được dùng với mục đích chính là:
 - (a) Uớc lượng thành phần tần số thấp (low-pass) của chuỗi để thu được trend.
 - (b) Khuếch đại thành phần tần số cao (high-pass) nhằm nhấn mạnh seasonal/residual.
 - (c) Thay thế dữ liệu gốc bằng z-score.
 - (d) Sinh nhẫn phân lớp hai lớp.
3. (Lý thuyết) Với cửa sổ $\mathbf{x} \in \mathbb{R}^L$, chọn mốc $\ell = x_L$, đặt $\mathbf{x}' = \mathbf{x} - \ell \mathbf{1}_L$, dự báo $\hat{\mathbf{y}}' = W\mathbf{x}' + b$, hoàn nguyên $\hat{\mathbf{y}} = \hat{\mathbf{y}}' + \ell \mathbf{1}_T$. Tính chất dịch mức cộng tính (additive level-shift equivariance) nào sau đây là đúng?
 - (a) $\hat{\mathbf{y}}(\mathbf{x} + \Delta \mathbf{1}_L) = \hat{\mathbf{y}}(\mathbf{x})$.
 - (b) $\hat{\mathbf{y}}(\mathbf{x} + \Delta \mathbf{1}_L) = \hat{\mathbf{y}}(\mathbf{x}) + \Delta \mathbf{1}_T$.
 - (c) $\hat{\mathbf{y}}(\mathbf{x} + \Delta \mathbf{1}_L) = 2 \hat{\mathbf{y}}(\mathbf{x})$.
 - (d) Không có bảo toàn nào liên quan đến dịch mức.
4. (Tính toán) MA với phép lặp biên (nghĩa lặp lại các giá trị ở biên của chuỗi dữ liệu để đệm cho các phép tính ở đầu và cuối chuỗi). Cho $m=3$, $\mathbf{x} = [10, 12, 11, 13]$. Tính $MA_3(\mathbf{x})$ theo

$$MA_3(x)_t = \frac{1}{3}(x_{t-1}^* + x_t^* + x_{t+1}^*), \quad x^* \text{ lặp biên (replicate)}.$$

- (a) [10.67, 11.00, 12.00, 12.33]

- (b) [11.00, 11.00, 12.00, 12.00]
 (c) [10.00, 11.00, 11.67, 13.00]
 (d) [10.67, 11.33, 11.67, 12.33]
5. (Tính toán) NLinear (re-centering). $L=3, T=2$. $\mathbf{x} = [2, 3, 5], \ell = x_3 = 5$. $W = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. Tính $\hat{\mathbf{y}}$:
- $$\mathbf{x}' = \mathbf{x} - \ell\mathbf{1}, \quad \hat{\mathbf{y}}' = W\mathbf{x}' + b, \quad \hat{\mathbf{y}} = \hat{\mathbf{y}}' + \ell\mathbf{1}.$$
- (a) [2, 2]
 (b) [3, 3]
 (c) [4, 4]
 (d) [6, 6]
6. (Tính toán) NLinear – minh họa tính chất shift. Cho $L = 2, T = 1$, $\mathbf{x} = [2, 5]$, mốc $\ell = x_2 = 5, W = [1, 0], b = 0$. Tính $\hat{y}(\mathbf{x})$ và $\hat{y}(\mathbf{x}+3\mathbf{1})$. Kết quả là:
- (a) $\hat{y}(\mathbf{x}) = 2, \hat{y}(\mathbf{x}+3\mathbf{1}) = 2$
 (b) $\hat{y}(\mathbf{x}) = 2, \hat{y}(\mathbf{x}+3\mathbf{1}) = 5$
 (c) $\hat{y}(\mathbf{x}) = 2, \hat{y}(\mathbf{x}+3\mathbf{1}) = -1$
 (d) $\hat{y}(\mathbf{x}) = 2, \hat{y}(\mathbf{x}+3\mathbf{1}) = 4$
7. (Tính toán) DLinear – phân rã và hai đầu tuyến tính. $L=4, T=1, m=3$, lặp biên. $\mathbf{x} = [10, 12, 11, 13], \mathbf{x}_t = \text{MA}_3(\mathbf{x}), \mathbf{x}_s = \mathbf{x} - \mathbf{x}_t. W_t = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \end{bmatrix}, b_t = 0; W_s = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}, b_s = 1$. Tính $\hat{y} = (W_t\mathbf{x}_t + b_t) + (W_s\mathbf{x}_s + b_s)$.
- (a) 11.16
 (b) 12.05
 (c) 12.52
 (d) 13.09
8. (Tính toán) MSE (DMS) cho một mẫu. $T=3, \hat{\mathbf{y}} = [3, 4, 5], \mathbf{y} = [2, 5, 5]$. Tính $\frac{1}{3} \sum_{t=1}^3 (\hat{y}_t - y_t)^2$.

- (a) 0.2222
(b) 0.3333
(c) 0.6667
(d) 1.0000
9. (Tính toán) RMSE cho 7 bước. Với $\mathbf{e} = [1, -2, 2, -1, 0, 3, -3]$, $RMSE = \sqrt{\frac{1}{7} \sum_{t=1}^7 e_t^2}$ bằng:
(a) 1.732
(b) 2.000
(c) 2.160
(d) 2.236
10. (Tính toán) Linear – ánh xạ theo trục thời gian. $L=4, T=2, \mathbf{x} = [1, 0, 2, 1]$. $W = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Tính $\hat{\mathbf{y}} = W\mathbf{x} + b$.
(a) [3, 2]
(b) [2, 3]
(c) [1, 4]
(d) [4, 1]

1. **Hint:** Các file code gợi ý và dữ liệu nếu có được lưu trong thư mục có thể được tải [tại đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải [tại đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại [đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng tối đa 4 giờ.

AIO_QAs-Verified

■ Nhóm Riêng tư · 1,4K thành viên



Hình 48: Hình ảnh group facebook AIO Q&A

4. **Đề xuất phương pháp cải tiến project:** Một số phương pháp đề xuất cải tiến model này xem thêm ở 29.8
5. **Rubric:**

LTSF-Linear Project - Rubric		
Phần	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Nắm vững các khái niệm cơ bản của dự báo chuỗi thời gian. - Hiểu các đặc tính của dữ liệu tài chính (nhiều, phi tuyến tính, không dừng) và các thành phần của chuỗi (xu hướng, mùa vụ, nhiều). 	<ul style="list-style-type: none"> - Phân tích tại sao một mô hình đơn giản lại có thể hoạt động tốt trong miền dữ liệu nhiễu. - Trả lời các câu hỏi trắc nghiệm lý thuyết liên quan đến khái niệm DMS và các thách thức của LTSF.
2	<ul style="list-style-type: none"> - Hiểu kiến trúc cơ sở của mô hình Linear. - Nắm rõ cơ chế phân rã của DLinear. - Nắm rõ cơ chế chuẩn hóa của NLinear. 	<ul style="list-style-type: none"> - Hoàn thiện code (from scratch) cho các hàm ‘forward’ của ba mô hình Linear, DLinear, và NLinear. - Hoàn thiện code định nghĩa các lớp mô hình Linear, DLinear, NLinear trong project thực tế
3	<ul style="list-style-type: none"> - Hiểu quy trình xử lý dữ liệu chuỗi thời gian thực tế. - Nắm bắt các hướng cải tiến và mở rộng: mở rộng sang bài toán đa biến (multivariate) hoặc kết hợp mô hình lai (hybrid) như DLinear + Transformer. 	<ul style="list-style-type: none"> - Xây dựng pipeline dữ liệu cho bài toán dự đoán giá cổ phiếu VIC. - Phân tích kết quả dự đoán và thảo luận về giới hạn cũng như các hướng cải tiến tiếp theo.

Chương 30

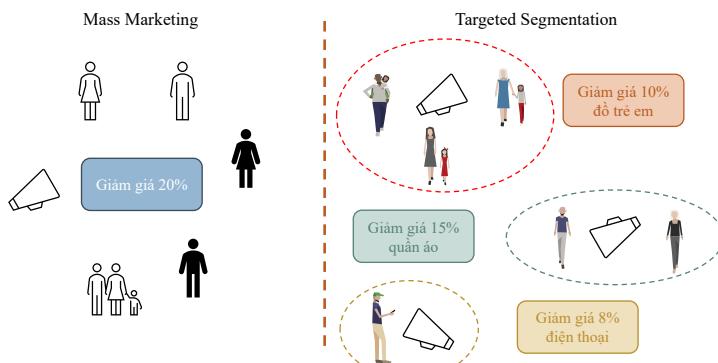
Project 2: Tách phân khúc khách hàng cho chiến lược marketing dùng Unsupervised Learning

30.1 Phân khúc khách hàng

Trong kỷ nguyên Big Data, việc áp dụng một chiến lược kinh doanh duy nhất cho toàn bộ thị trường (hay còn gọi là mass marketing) không còn mang lại hiệu quả tối ưu. Khách hàng ngày nay mong đợi sự cá nhân hóa trong trải nghiệm mua sắm và dịch vụ. Đây là lúc bài toán “Customer Segmentation” (tạm dịch: phân khúc khách hàng) trở nên cấp thiết.

Phân khúc khách hàng là quá trình chia cơ sở khách hàng thành các nhóm nhỏ dựa trên những đặc điểm tương đồng như hành vi mua sắm, nhân khẩu học, hoặc tần suất chi tiêu. Mục tiêu cốt lõi của việc này bao gồm:

- **Tối ưu hóa Marketing (Targeted Marketing):** Thay vì gửi một email quảng cáo cho tất cả, doanh nghiệp có thể gửi khuyến mãi “đồ mẹ và bé” cho nhóm khách hàng là phụ huynh, hoặc “thiết bị công nghệ” cho nhóm khách hàng trẻ yêu thích kỹ thuật số.
- **Giữ chân khách hàng (Customer Retention):** Nhận diện nhóm khách hàng có nguy cơ rời bỏ (Churn rate cao) để có chính sách chăm sóc đặc biệt.
- **Phân bổ nguồn lực:** Tập trung ngân sách vào nhóm khách hàng mang lại giá trị lợi nhuận cao nhất (nhóm VIP).



Hình 49: Sự chuyển dịch từ tiếp thị đại trà kém hiệu quả sang phân khúc mục tiêu được cá nhân hóa.

Để thực hiện việc phân loại này một cách tự động và chính xác trên tập dữ liệu lớn, chúng ta không thể thực hiện thủ công. Ta cần sự hỗ trợ của các thuật toán Machine Learning, cụ thể là nhóm thuật toán học không giám sát (Unsupervised Learning).

Trong dự án này, chúng ta sẽ sử dụng **K-Means Clustering** (Phân cụm K-Means) - một trong những thuật toán phổ biến và hiệu quả nhất để giải quyết bài toán phân nhóm dữ liệu chưa được dán nhãn.

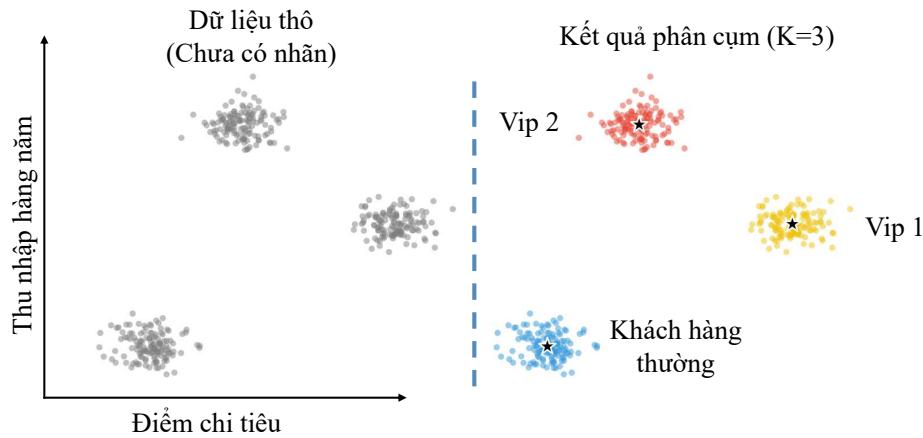
Tiêu chí	Phân loại thủ công	K-Means Clustering
Độ phức tạp	Khó xử lý nhiều chiều dữ liệu cùng lúc.	Xử lý tốt đa chiều (Tuổi, Thu nhập, Tần suất mua...).
Tính khách quan	Dễ bị ảnh hưởng bởi cảm tính người phân loại.	Dựa hoàn toàn vào khoảng cách toán học giữa các điểm dữ liệu.
Khả năng mở rộng	Tốn thời gian khi dữ liệu lớn.	Tốc độ xử lý nhanh, phù hợp với Big Data.

Bảng 2: So sánh hiệu quả giữa phân loại thủ công và thuật toán K-Means.

Trong bài học này, chúng ta sẽ đóng vai là một Data Scientist cho một

chuỗi bán lẻ. Nhiệm vụ của chúng ta là phân tích dữ liệu giao dịch và xây dựng mô hình K-Means để nhóm khách hàng. Các bước thực hiện bao gồm:

1. **Khám phá dữ liệu (EDA):** Hiểu cấu trúc và phân bố của tập dữ liệu.
2. **Tiền xử lý:** Chuẩn hóa dữ liệu để đảm bảo thuật toán hoạt động chính xác.
3. **Lựa chọn số cụm K tối ưu:** Sử dụng phương pháp Elbow Method hoặc Silhouette score để tìm K tốt nhất.
4. **Huấn luyện mô hình:** Khởi tạo và huấn luyện K-Means.
5. **Phân tích kết quả:** Trực quan hóa các cụm và đưa ra đề xuất kinh doanh (Actionable Insights).



Hình 50: Trực quan hóa cách K-Means nhóm các điểm dữ liệu thô (trái) thành các cụm có tâm xác định (phải).

Danh sách các module trong project. Để tiện theo dõi, bảng dưới đây tổng hợp các phần chính trong module của chúng ta. Toàn bộ dự án được chứa trong link Github [này](#).

Bảng thành phần Project

Thành phần	Ý nghĩa và chức năng
<code>src/clustering_library.py</code>	Thư viện trung tâm của dự án, chứa các lớp (class) xử lý chính: <code>DataCleaner</code> (làm sạch dữ liệu), <code>FeatureEngineer</code> (tạo đặc trưng khách hàng), <code>ClusterAnalyzer</code> (thực hiện phân cụm K-Means) và <code>DataVisualizer</code> (trực quan hóa).
<code>notebooks/01_cleaning_and_eda.ipynb</code>	Notebook bước 1: Thực hiện tải dữ liệu thô, xử lý các giá trị thiếu, lọc dữ liệu rác (đơn hàng bị hủy) và thực hiện phân tích khám phá dữ liệu (EDA) để hiểu tổng quan về tập dữ liệu.
<code>notebooks/02_feature_engineering.ipynb</code>	Notebook bước 2: Chịu trách nhiệm tạo ra 16 đặc trưng (features) hành vi khách hàng từ dữ liệu giao dịch, thực hiện biến đổi Box-Cox để chuẩn hóa phân phối và Scaling dữ liệu.
<code>notebooks/03_modeling.ipynb</code>	Notebook bước 3: Tìm số cụm tối ưu (sử dụng Elbow Method và Silhouette Score), huấn luyện mô hình K-Means, và phân tích đặc điểm của từng nhóm khách hàng được tạo ra.
<code>data/</code>	Thư mục chứa dữ liệu đầu vào (<code>online_retail.csv</code>) và lưu trữ các dữ liệu trung gian đã qua xử lý (processed data) để sử dụng giữa các bước phân tích.
<code>setup_code.py</code>	Script hỗ trợ khởi tạo cấu trúc thư mục chuẩn cho dự án, tạo các file cấu hình ban đầu và cài đặt môi trường cần thiết. 934

30.2 Phân tích và làm sạch dữ liệu

Trong bất kỳ dự án nào, việc hiểu và làm sạch dữ liệu là bước nền tảng quan trọng nhất. Dữ liệu thực tế thường chứa nhiễu, giá trị thiếu hoặc các ngoại lệ (outliers) có thể làm sai lệch kết quả của mô hình phân cụm. Trong phần này, ta sẽ sử dụng bộ dữ liệu *Online Retail* để thực hiện quy trình từ làm sạch đến phân tích khám phá (Exploratory Data Analysis - EDA).

❓ Tại sao chúng ta dùng OOP?

Để đảm bảo tính chuyên nghiệp khi trình bày cho người nghe, dự án này áp dụng tư duy Lập trình hướng đối tượng (OOP). Thay vì viết hàng trăm dòng script xử lý rối rắm vào chính những file notebook mà chúng ta trình bày, chúng ta đóng gói logic vào các lớp (Class) trong một file riêng. Sau đó, khi làm việc với notebook ta có, ta sẽ sử dụng hàm `import` để sử dụng các lớp nói trên. Điều này giúp notebook sạch sẽ, tập trung vào kết quả và dễ dàng tái sử dụng.

Ví dụ minh họa sự tinh gọn:

```
1 from clustering_library import DataCleaner, DataVisualizer  
2  
3 # Code ngắn gọn, dễ hiểu cho người đọc  
4 cleaner = DataCleaner(file_path='online_retail.csv')  
5 df_uk = cleaner.clean_data()
```

Tiếp theo, chúng ta sẽ sử dụng class `DataCleaner` này để xử lý tập dữ liệu *Online Retail*, bao gồm việc loại bỏ các hóa đơn hủy và lọc khách hàng UK.

30.2.1 Tải và kiểm tra dữ liệu ban đầu

Đầu tiên, ta cần tải dữ liệu và kiểm tra cấu trúc cơ bản. Bộ dữ liệu bao gồm các giao dịch từ tháng 12/2010 đến tháng 12/2011 của một doanh nghiệp bán lẻ trực tuyến tại Anh.

Ta sử dụng lớp `DataCleaner` từ thư viện `clustering_library` để quản lý việc tải dữ liệu:

Tải dữ liệu

```
1 from clustering_library import DataCleaner, DataVisualizer
2
3 # Khởi tạo và tải dữ liệu
4 data_path = "../data/raw/online_retail.csv"
5 cleaner = DataCleaner(data_path)
6 df = cleaner.load_data()
7
8 # Kiểm tra thông tin cơ bản
9 print(f"Kích thước: {df.shape}")
10 print(f"Số lượng khách hàng: {df['CustomerID'].nunique()}")
```

Dữ liệu thô ban đầu chứa hơn 500,000 dòng, tuy nhiên bao gồm cả các giao dịch bị hủy, dữ liệu thiếu thông tin khách hàng và các giao dịch từ nhiều quốc gia khác nhau.

30.2.2 Làm sạch dữ liệu

Để đảm bảo tính nhất quán cho việc phân khúc khách hàng, ta thực hiện các bước làm sạch sau:

1. **Tính toán tổng giá trị:** Tạo cột TotalPrice = Quantity × UnitPrice.
2. **Xử lý đơn hàng hủy:** Loại bỏ các hóa đơn bắt đầu bằng ký tự 'C' (Cancelled).
3. **Lọc theo địa lý:** Chỉ giữ lại các giao dịch tại **United Kingdom** để tránh nhiễu do sự khác biệt về hành vi mua sắm giữa các quốc gia.
4. **Xử lý dữ liệu thiếu:** Loại bỏ các bản ghi không có CustomerID vì ta cần định danh khách hàng để phân cụm.
5. **Làm sạch giá trị:** Loại bỏ các giao dịch có số lượng hoặc đơn giá âm/bằng 0.

Thực hiện làm sạch dữ liệu

```
1 # Thực hiện quy trình làm sạch
2 df_uk = cleaner.clean_data()
3
4 # Tạo thêm các đặc trưng thời gian (Giờ, Ngày trong tuần)
5 cleaner.create_time_features()
6
7 print(f"Số lượng giao dịch sau khi làm sạch: {len(df_uk)}")
```

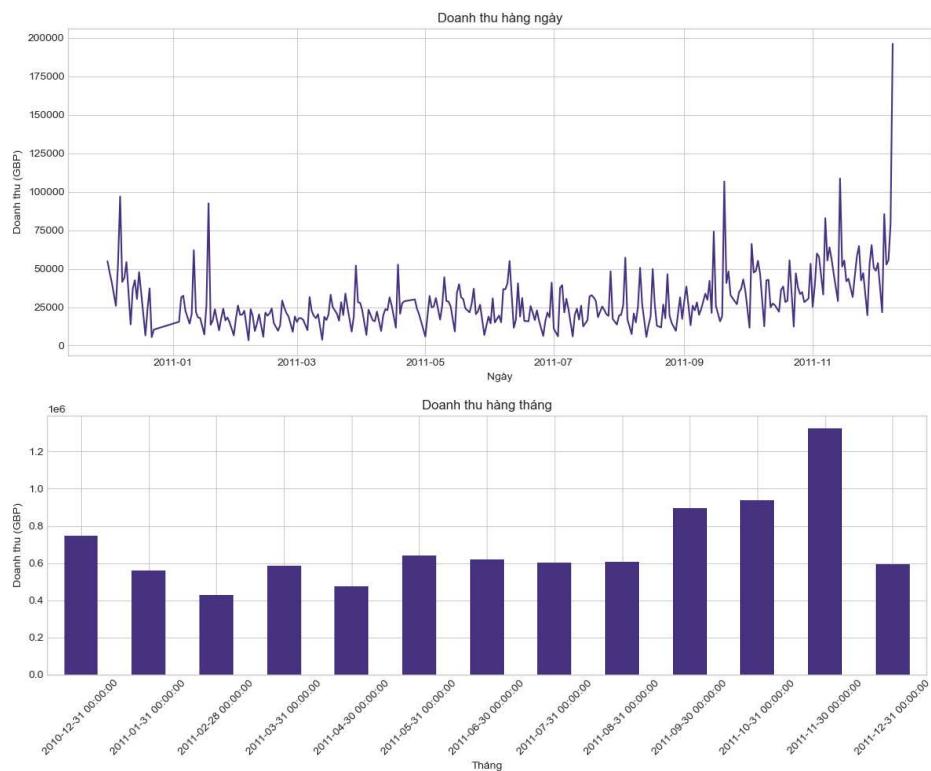
Sau quá trình này, ta thu được bộ dữ liệu sạch, sẵn sàng cho việc phân tích sâu hơn.

30.2.3 EDA

Sau khi có được bộ dữ liệu đã được làm sạch, sau đây chúng ta sẽ thực hiện việc nghiên cứu bộ dữ liệu này qua phương pháp trực quan hóa. Các bạn muốn tìm hiểu về cách vẽ nên các chart như dưới đây có thể xem thêm ở file `src/clustering_library.py` và lớp `DataVisualizer` trong dự án.

Doanh thu theo thời gian

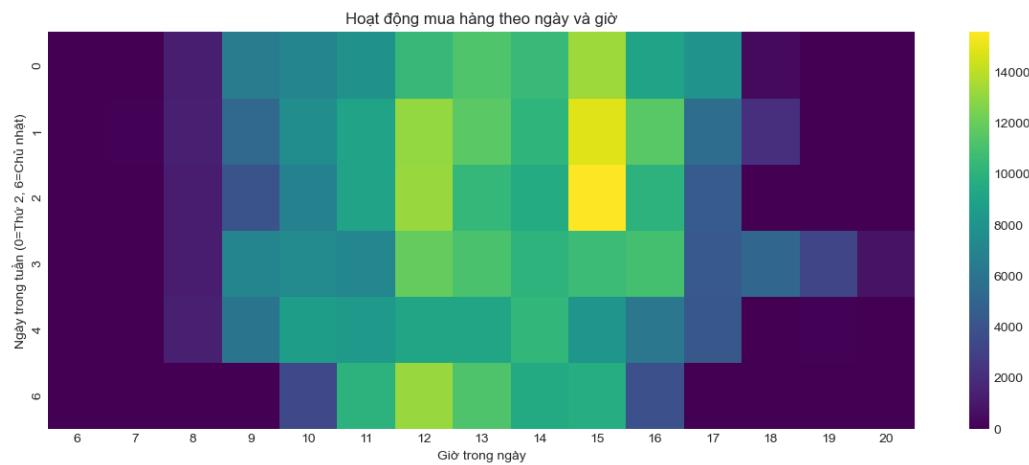
Trước tiên, việc hiểu rõ xu hướng doanh thu giúp ta nhận diện tính mùa vụ và sức khỏe chung của doanh nghiệp. Biểu đồ dưới đây thể hiện doanh thu theo ngày và theo tháng.



Hình 51: Biểu đồ xu hướng doanh thu theo ngày và tháng. Ta có thể quan sát thấy sự biến động doanh thu và các đỉnh điểm mua sắm vào dịp cuối năm.

Hành vi mua sắm

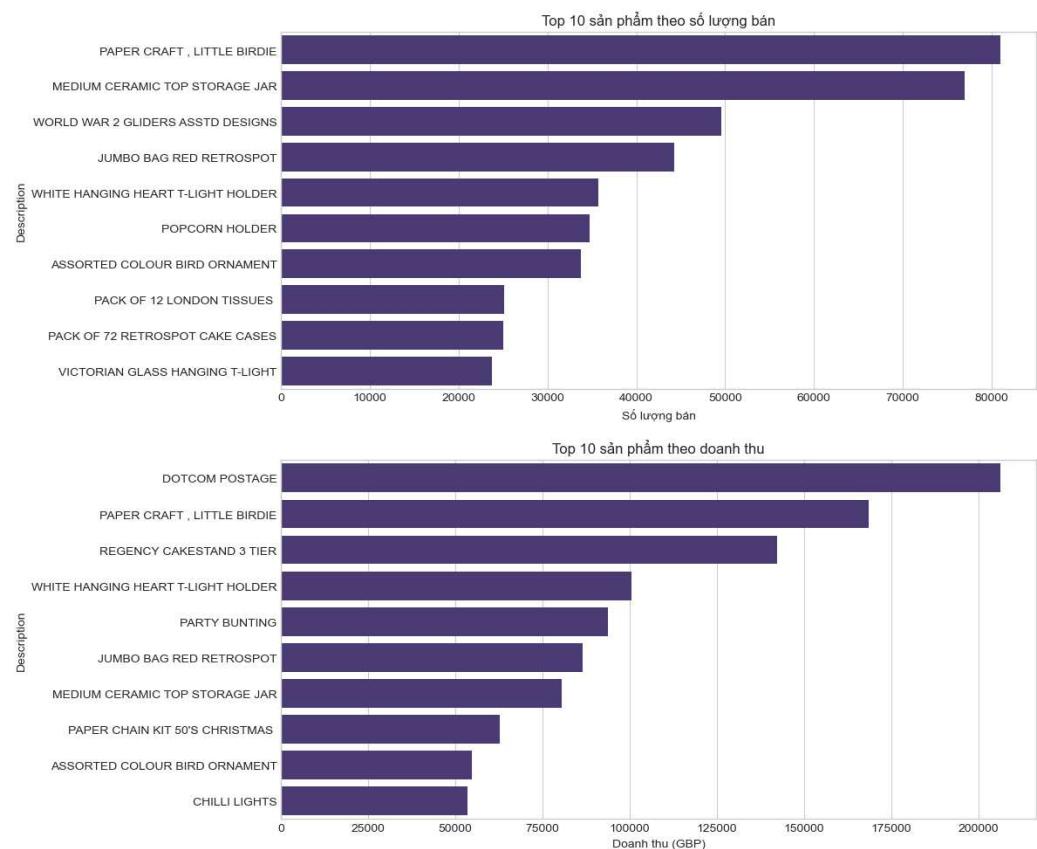
Để tối ưu hóa các chiến dịch marketing, ta cần biết khách hàng thường mua sắm vào khung giờ nào trong ngày và ngày nào trong tuần.



Hình 52: Heatmap thể hiện tần suất mua hàng theo “giờ” trong ngày và “ngày” trong tuần. Vùng màu sáng hơn thể hiện mật độ giao dịch cao hơn. Khách hàng của chúng ta thường mua vào giờ hành chính.

Phân tích sản phẩm và khách hàng

Ta cũng cần xem xét các sản phẩm bán chạy nhất và phân phối chi tiêu của khách hàng để có cái nhìn tổng quan về danh mục sản phẩm hay còn gọi là product portfolio.



Hình 53: Top 10 sản phẩm bán chạy nhất theo số lượng và doanh thu. Các sản phẩm này thường là các mặt hàng trang trí hoặc quà tặng phổ biến.

Phân tích RFM (Recency - Frequency - Monetary)

Và cuối cùng, trước khi đi vào các kỹ thuật phân cụm phíc tạp, ta xem xét phân phối của ba chỉ số cơ bản RFM. Đây không chỉ là các con số thống kê, mà là ba trụ cột giúp trả lời các câu hỏi chiến lược về hành vi khách hàng:

- **Recency (sự mới mẻ):** Số ngày kể từ lần mua cuối cùng.
→ *Ý nghĩa:* Đo lường **mức độ gắn kết**. Khách hàng mới mua gần đây có xác suất quay lại cao hơn, trong khi Recency cao báo hiệu nguy cơ rời bỏ.
- **Frequency (tần suất):** Tổng số đơn hàng đã thực hiện.

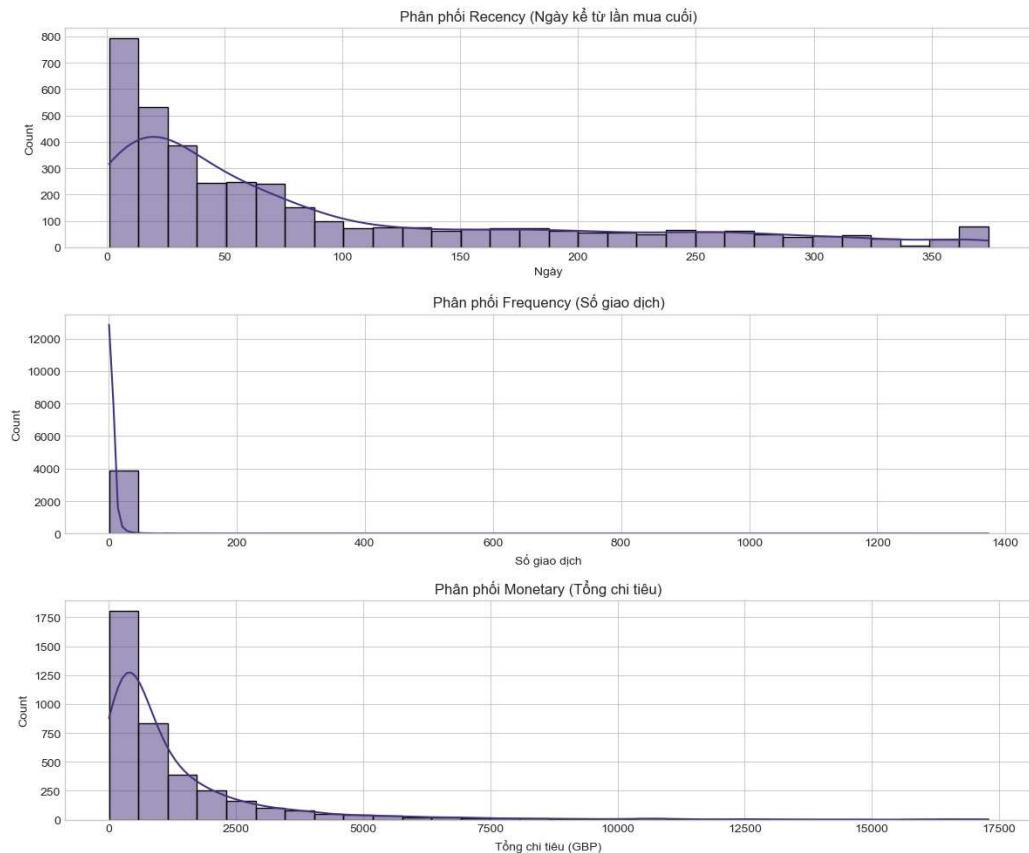
→ *Ý nghĩa*: Đo lường **độ trung thành**. Giúp phân biệt giữa khách vãng lai và khách hàng thân thiết có thói quen mua sắm định kỳ.

- **Monetary (giá trị tiền tệ)**: Tổng số tiền khách hàng đã chi tiêu.

→ *Ý nghĩa*: Đo lường **giá trị trọng đài**. Giúp nhận diện nhóm “Cá mập” - những người đóng góp phần lớn doanh thu theo nguyên lý Pareto (80/20).

Tính toán chỉ số RFM

```
1 # Tính toán RFM
2 rfm_data = cleaner.calculate_rfm()
3 visualizer.plot_rfm_analysis(rfm_data)
```



Hình 54: Phân phối của các chỉ số Recency, Frequency và Monetary. Đồ thị cho thấy sự phân phối lệch (skewed), đặc trưng của dữ liệu bán lẻ (quy luật 80/20).

Sau quá trình làm sạch và EDA, ta đã có một bộ dữ liệu chất lượng cao từ thị trường UK. Các phân tích ban đầu cho thấy sự đa dạng lớn trong hành vi khách hàng, từ những người mua sắm thường xuyên đến những khách hàng vãng lai. Đây là tiền đề vững chắc để ta bước sang giai đoạn tiếp theo: Feature Engineering (tạm dịch: tạo thêm đặc trưng) để xây dựng các biến số đầu vào cho mô hình K-Means.

⌚ Tổng quan bước 1

Trong phần này, chúng ta đã hoàn tất quy trình chuẩn bị dữ liệu từ tập dữ liệu Online Retail thô. Sau khi tiền xử lý, ta tinh lọc được một bộ dữ liệu chất lượng cao gồm **354,321** giao dịch từ **3,920** khách hàng duy nhất trong giai đoạn từ tháng 12/2010 đến tháng 12/2011.

Quá trình EDA đã cho ta thấy những xu hướng quan trọng: doanh thu toàn cục có xu hướng tăng trưởng tích cực theo thời gian và hoạt động mua sắm tập trung chủ yếu vào khung giờ hành chính các ngày trong tuần. Đặc biệt, việc phân tích sơ bộ chỉ số **RFM** (Recency, Frequency, Monetary) cho thấy sự đa dạng lớn trong tần suất và giá trị chi tiêu của từng cá nhân.

30.3 Tạo đặc trưng

Sau khi làm sạch dữ liệu, bước tiếp theo là chuyển đổi dữ liệu giao dịch thô thành các đặc trưng có ý nghĩa ở cấp độ khách hàng. Thay vì chỉ sử dụng mô hình RFM truyền thống (3 biến), ta sẽ xây dựng một bộ 16 đặc trưng toàn diện để nắm bắt đa chiều hành vi mua sắm của người dùng.

30.3.1 Tại sao cần mở rộng Features?

Mô hình RFM (Recency, Frequency, Monetary) là tiêu chuẩn trong việc phân khúc khách hàng, nhưng nó có những hạn chế nhất định:

- **Thiếu chiều sâu về sản phẩm:** RFM không cho biết khách hàng mua đa dạng sản phẩm hay chỉ tập trung vào một vài mặt hàng.
- **Bỏ qua hành vi trong từng giao dịch:** RFM chỉ nhìn vào tổng thể, không phản ánh thói quen chi tiêu trong mỗi lần mua (ví dụ: mua ít nhưng giá trị cao vs mua nhiều nhưng giá trị thấp).
- **Độ nhạy với giá:** RFM không tách biệt rõ ràng giữa khách hàng mua hàng giá rẻ số lượng lớn và khách hàng mua hàng cao cấp số lượng ít.

Do đó, ta xây dựng bộ 16 features chia làm 4 nhóm chính để khắc phục các hạn chế trên.

30.3.2 Xây dựng bộ đặc trưng

Ta sử dụng lớp FeatureEngineer để tự động hóa quá trình này.

Khởi tạo Feature Engineering

```

1 from clustering_library import FeatureEngineer
2
3 # Tải dữ liệu đã làm sạch
4 data_path = '../data/processed/cleaned_uk_data.csv'
5 engineer = FeatureEngineer(data_path)
6 df = engineer.load_data()
7
8 # Tạo features
9 customer_features = engineer.create_customer_features()

```

Dưới đây là chi tiết ý nghĩa của 4 nhóm đặc trưng được tạo thêm. Để đọc đầy đủ ý nghĩa của từng thuộc tính một trong 16 thuộc tính, bạn có thể tham khảo file notebooks/02_feature_engineering.ipynb:

Nhóm 1: Chỉ số cơ bản (Volume & Value)

Nhóm này mở rộng từ dữ liệu RFM, cung cấp cái nhìn tổng quan về quy mô tiêu dùng.

- Sum_Quantity: Tổng số lượng sản phẩm đã mua.
- Mean_UnitPrice: Mức giá trung bình của các sản phẩm khách hàng chọn (độ nhạy về giá).
- Sum_TotalPrice: Tổng chi tiêu trọn đời (tương đương Monetary).
- Count_Invoice: Số lần mua hàng (tương đương Frequency).
- Count_Stock: Số lượng mã sản phẩm khác nhau đã mua (độ đa dạng).

Nhóm 2: Hành vi theo Giao dịch (Transaction Behavior)

Nhóm này trả lời câu hỏi: “Mỗi lần vào cửa hàng, khách hàng cư xử thế nào?”

- Mean_QuantitySumPerInvoice: Giỏ hàng trung bình to hay nhỏ?
- Mean_TotalPriceSumPerInvoice: Giá trị trung bình mỗi hóa đơn.
- Mean_StockCountPerInvoice: Trung bình mỗi lần mua bao nhiêu loại hàng?

Nhóm 3 & 4: Hành vi theo Sản phẩm (Product Preference)

Nhóm này đi sâu vào sở thích sản phẩm cụ thể.

- Mean_InvoiceCountPerStock: Trung bình một sản phẩm được mua lại bao nhiêu lần (độ trung thành với sản phẩm).
- Mean_UnitPriceMeanPerStock: Giá trị trung bình của từng dòng sản phẩm khách hàng chọn.

30.3.3 Chuẩn hóa và biến đổi Dữ liệu

Quan sát trực quan dữ liệu gốc qua biểu đồ Histogram và Boxplot (hình dưới), ta nhận thấy một vấn đề nghiêm trọng: dữ liệu khách hàng bị lệch phải (right-skewness). Hầu hết dữ liệu tập trung ở giá trị thấp (ví dụ: số lượng mua ít), nhưng có một “đuôi dài” các giá trị cực lớn kéo về phía bên phải.

Thay vì chỉ sử dụng Logarit thông thường, ta áp dụng phép biến đổi Box-Cox ở đây. Đây là kỹ thuật tìm ra tham số λ tối ưu để biến đổi dữ liệu về dạng gần với phân phối chuẩn (Gaussian) nhất có thể, giúp:

1. Co ngắn khoảng cách của các giá trị ngoại lai (xử lý đuôi dài).
2. Ổn định phương sai.
3. Tạo ra không gian dữ liệu đối xứng hơn cho K-Means hoạt động hiệu quả.

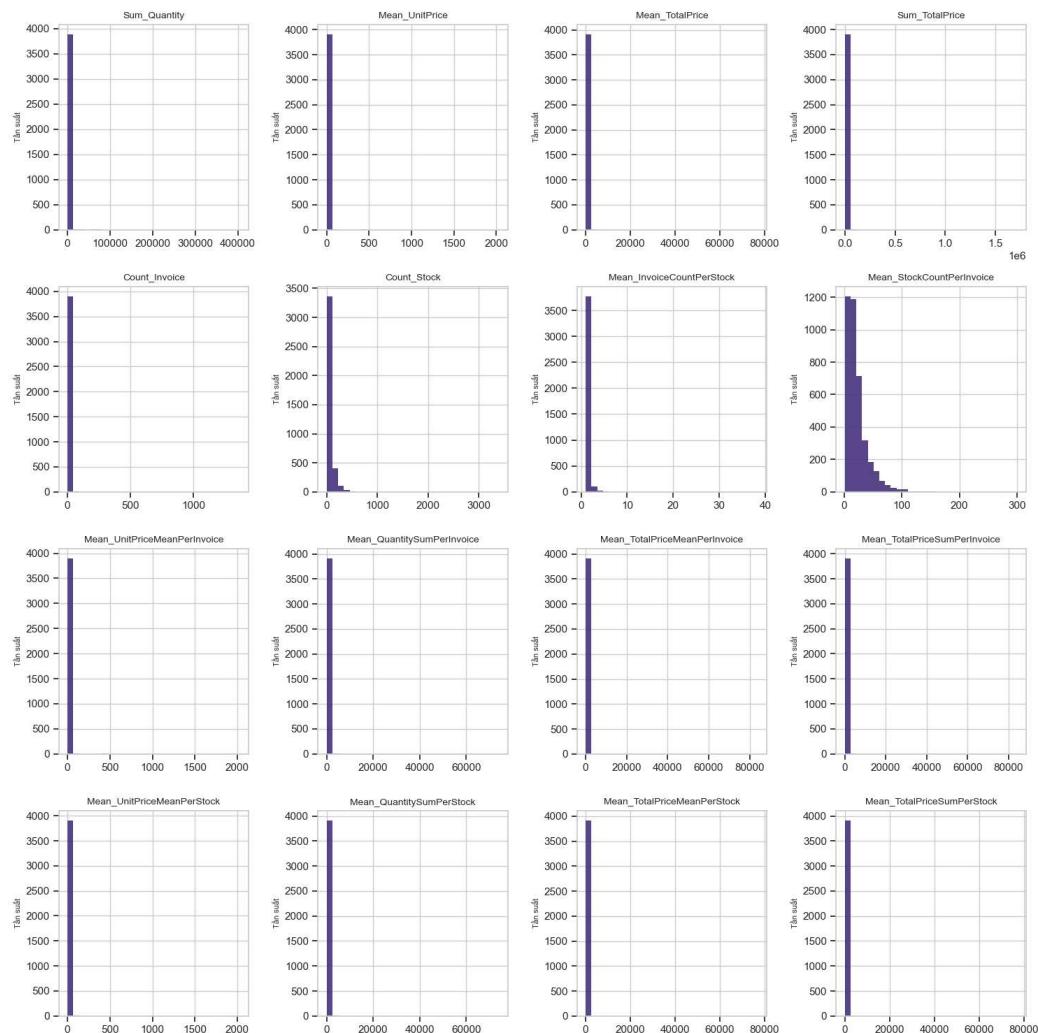
Box-Cox Transformation

Code cho thuật toán Box-Cox được trình bày như dưới đây. Trước hết, ‘CustomerID’ được tách ra làm index để không bị biến đổi nhầm, tránh

gây nhiễu mô hình. Tiếp theo, dữ liệu được cộng thêm 1 ($x + 1$) để xử lý các giá trị bằng 0 (do toán học của Box-Cox yêu cầu đầu vào dương tuyệt đối). Cuối cùng, hàm ‘boxcox’ được áp dụng lần lượt lên từng đặc trưng để tự động tìm ra tham số λ tối ưu riêng biệt, giúp đưa biểu đồ phân phối của các chỉ số RFM từ dạng lệch (skewed) về dạng hình chuông (Gaussian).

Định nghĩa hàm Box-Cox

```
1 def transform_features(self):
2     # Set CustomerID as index
3     customer_features_indexed = self.customer_features.set_index("CustomerID")
4
5     # Apply Box-Cox transformation
6     feature_values = customer_features_indexed.values + 1
7     self.customer_features_transformed = customer_features_indexed.
8         copy()
9
10    for i, feature in enumerate(self.feature_customer):
11        transformed, lambda_param = boxcox(feature_values[:, i])
12        self.customer_features_transformed.iloc[:, i] = transformed
13
14    return self.customer_features_transformed
```



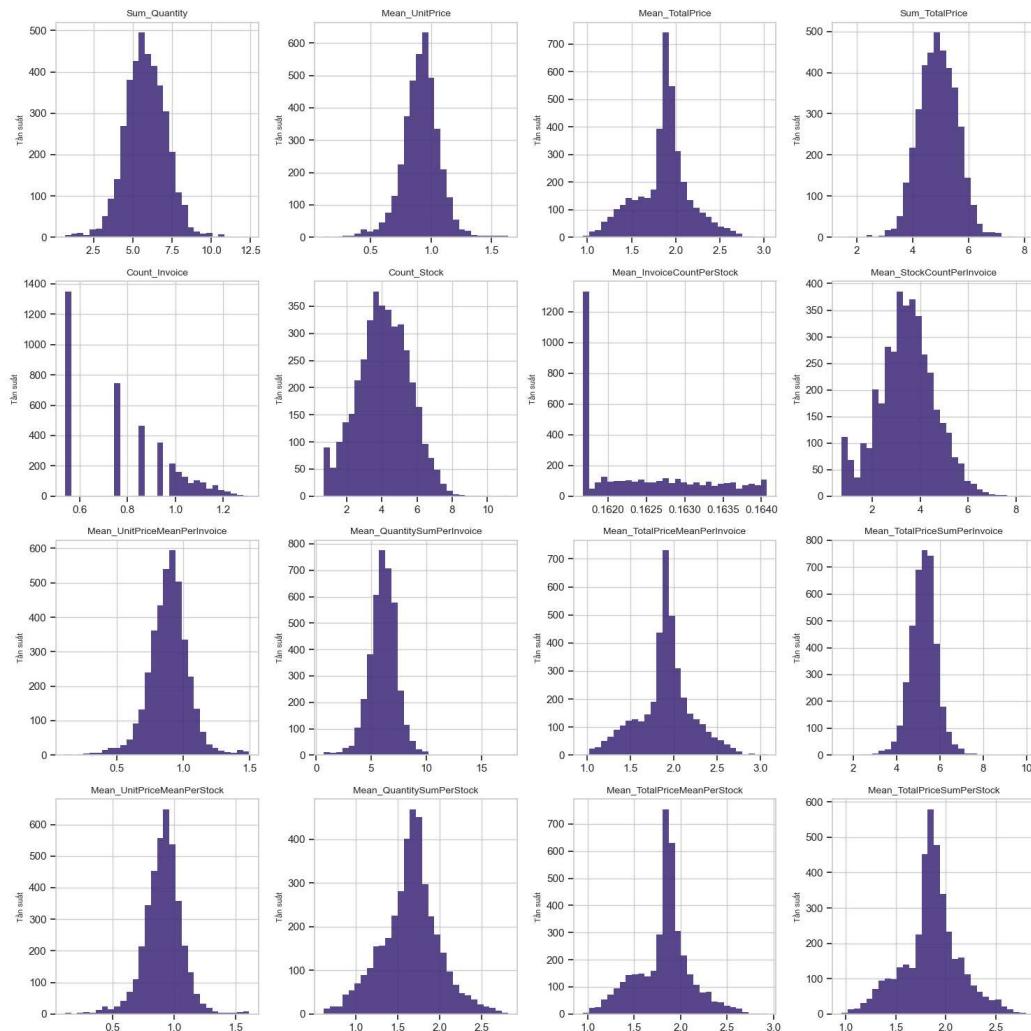
Hình 55: Phân phối của các features trước khi biến đổi. Ta thấy rõ sự lệch phải của dữ liệu.

Sử dụng hàm Box-Cox

```

1 # Áp dụng Box-Cox transformation
2 customer_features_transformed = engineer.transform_features()
3
4 # Trực quan hóa sau khi biến đổi
5 engineer.plot_features_histograms(transformed=True)

```



Hình 56: Phân phối của các features sau khi áp dụng Box-Cox. Dữ liệu đã trở nên đối xứng hơn, phù hợp cho K-Means.

Standard Scaling

Cuối cùng, ta đưa tất cả features về cùng một thang đo ($\text{mean}=0$, $\text{std}=1$) để đảm bảo không có feature nào lấn át các feature khác do đơn vị lớn hơn (ví dụ: Doanh thu hàng nghìn bảng vs Số lần mua hàng chục lần).

Standard Scaling

```

1 # Chuẩn hóa dữ liệu (Z-score normalization)
2 customer_features_scaled = engineer.scale_features()
3 engineer.save_features()

```

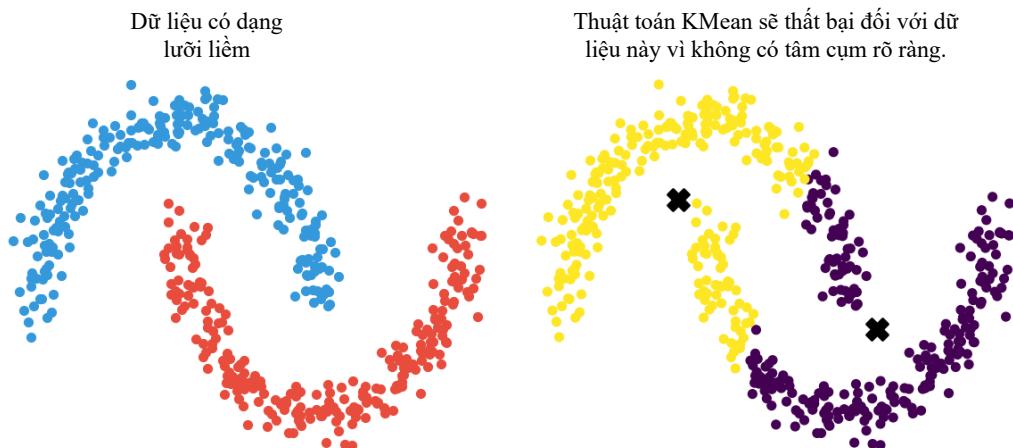
30.4 Mô hình hóa và phân cụm

Sau khi đã có bộ dữ liệu đặc trưng chất lượng cao, ta tiến hành bước quan trọng nhất: Phân nhóm khách hàng sử dụng thuật toán học máy không giám sát. Trong dự án này, ta sử dụng thuật toán K-Means clustering.

30.4.1 Giới thiệu về K-Means clustering

K-Means là một trong những thuật toán phân cụm phổ biến nhất nhờ sự đơn giản và hiệu quả. Mục tiêu của K-Means là chia N điểm dữ liệu thành K cụm sao cho khoảng cách giữa các điểm trong cùng một cụm là nhỏ nhất.

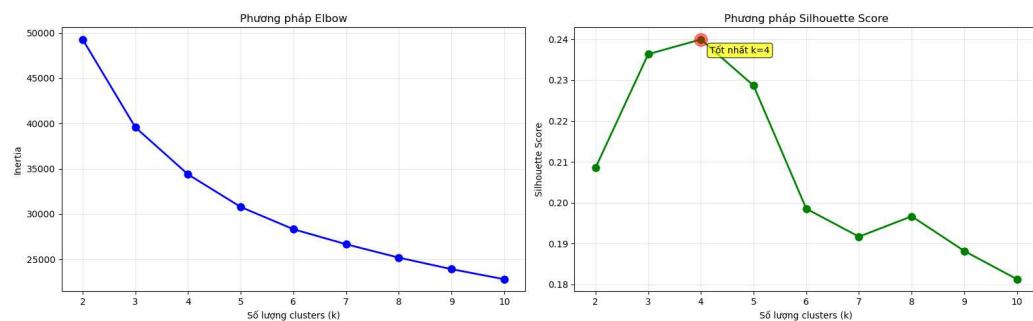
Để hiểu rõ hơn về cơ chế hoạt động và ứng dụng thực tế của K-Means, bạn có thể tham khảo bài viết chi tiết tại [đây](#).



Hình 57: Minh họa điểm yếu của K-Means với dữ liệu không có dạng hình cầu. Thuật toán thất bại trong việc phân tách hai cụm hình lưỡi liềm do dựa trên khoảng cách Euclidean.

30.4.2 Xác định số cụm tối ưu

Việc chọn số lượng cụm K phù hợp là bài toán mà ta cân bằng giữa độ chính xác và khả năng diễn giải các cụm đó. Để lựa chọn số lượng K phù hợp, ta sử dụng kết hợp hai phương pháp chính. Đầu tiên là “Elbow method”, giúp tìm điểm “khuỷu tay” nơi việc tăng thêm cụm không làm giảm đáng kể phuơng sai nội cụm. Phương pháp thứ hai là “Silhouette score”, dùng để đo lường độ tách biệt giữa các cụm, với giá trị càng gần 1 càng tốt.



Hình 58: Biểu đồ Elbow và Silhouette score. Dựa trên biểu đồ, ta thấy K=3 hoặc K=4 là các lựa chọn tối ưu.

30.4.3 Phân tích thành phần chính (PCA)

Trước khi phân cụm, ta sử dụng PCA để giảm chiều dữ liệu, giúp trực quan hóa dữ liệu nhiều chiều (16 chiều) xuống còn 2 hoặc 3 chiều để dễ quan sát cấu trúc phân bố.

Thực hiện PCA

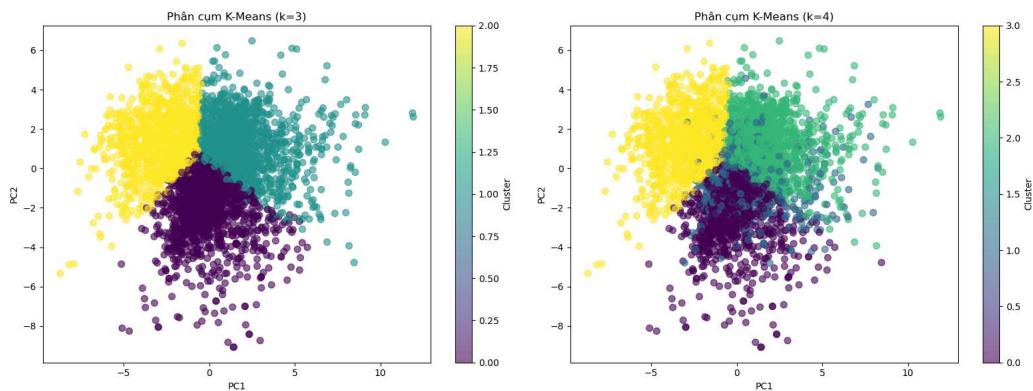
```

1 # Áp dụng PCA
2 df_pca = analyzer.apply_pca()
3
4 # Biểu đồ tỷ lệ phuơng sai được giải thích
5 analyzer.plot_pca_variance()

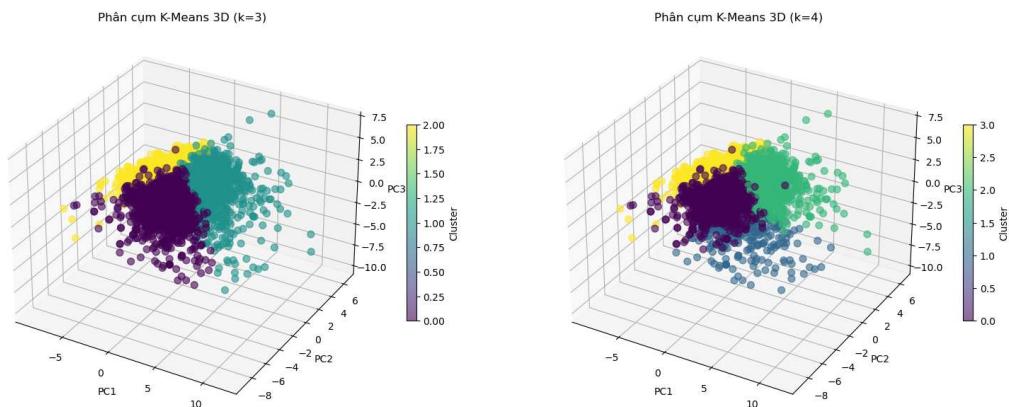
```

30.4.4 So sánh phân cụm K=3 và K=4

Việc lựa chọn số lượng cụm K là một quyết định quan trọng. Dựa trên biểu đồ Elbow và Silhouette score, ta cân nhắc giữa K=3 và K=4. Dưới đây là sự so sánh trực quan kết quả phân cụm trên không gian PCA 2D và 3D.



Hình 59: So sánh phân cụm trên không gian 2D. Bên trái là K=3, bên phải là K=4.



Hình 60: So sánh phân cụm trên không gian 3D. K=4 cho thấy sự tách biệt rõ ràng hơn ở vùng trung tâm.

Quan sát hình ảnh trực quan, ta thấy sự khác biệt rõ rệt:

Với K=3, dữ liệu được chia thành 3 vùng lớn:

- Vùng màu vàng bên trái: Tách biệt khá rõ, đại diện cho nhóm khách hàng có hành vi mua sắm đặc thù (thường là nhóm chi tiêu thấp hoặc ít hoạt động).
- Vùng màu xanh ngọc ở trên: Tập trung mật độ cao.
- Vùng màu tím ở dưới: Phân bố trải rộng hơn.

Với K=4, cấu trúc dữ liệu trở nên chi tiết hơn:

- Vùng màu vàng bên trái vẫn giữ nguyên sự ổn định, cho thấy đây là một nhóm khách hàng rất đặc trưng và tách biệt.
- Sự thay đổi lớn nhất nằm ở khói dữ liệu bên phải. Vùng giao thoa giữa nhóm xanh ngọc và tím ở K=3 nay đã được tách ra thành một cụm mới (màu xanh dương nhạt trong hình 3D hoặc màu xanh đậm hơn trong hình 2D).
- Điều này cho thấy K=4 giúp phát hiện ra một nhóm khách hàng trung gian hoặc nhóm chuyển đổi mà K=3 đã gộp chung vào các nhóm lớn. Về mặt kinh doanh, việc tách nhóm này ra giúp ta có chiến lược tiếp cận tinh tế hơn, tránh đánh đồng những khách hàng có hành vi “lưỡng chừng” với những khách hàng điển hình của nhóm lớn.

Dưới đây là mã nguồn để thực hiện phân cụm và vẽ các biểu đồ trên:

Thực hiện K-Means và trực quan hóa

```

1 # Áp dụng K-Means với k=3 và k=4
2 cluster_results = analyzer.apply_kmeans([3, 4])
3
4 # Trực quan hóa clusters trong không gian PCA 2D
5 analyzer.plot_clusters_pca([3, 4])
6
7 # Trực quan hóa clusters trong không gian PCA 3D
8 analyzer.plot_clusters_pca_3d([3, 4])

```

30.4.5 Phân tích chuyên sâu biểu đồ Radar

Biểu đồ Radar (hay Spider Chart) là công cụ đắc lực để trực quan hóa “hồ sơ” (profile) của từng nhóm khách hàng trên đa chiều dữ liệu. Thay vì chỉ

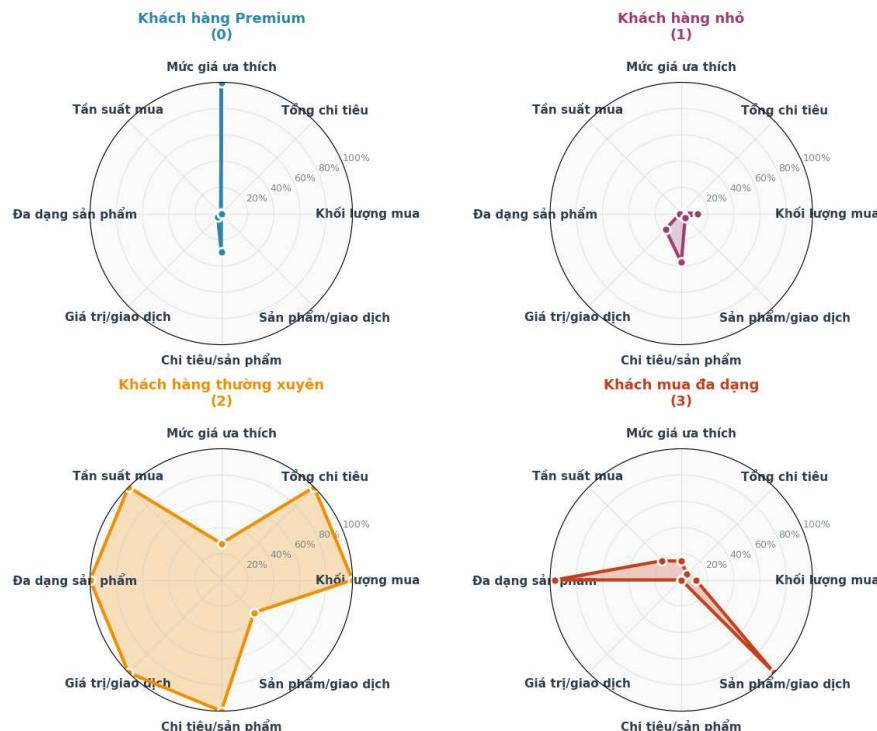
nhìn vào các con số thống kê khô khan, biểu đồ Radar giúp ta nhận diện ngay lập tức hình dáng hành vi của từng cụm.

So sánh cấu trúc phân cụm: K=3 và K=4

Trong quá trình thử nghiệm, ta nhận thấy sự chuyển dịch thú vị khi tăng số lượng cụm từ $K = 3$ lên $K = 4$. Việc này không chỉ đơn thuần là chia nhỏ dữ liệu, mà còn giúp tách biệt các nhóm hành vi bị gộp chung hay còn gọi là under-segmented.

Phân tích chi tiết từng Cluster (K=3)

(a) Kết quả phân cụm với $K=3$: Các nhóm còn khá tổng quát.

Phân tích chi tiết từng Cluster (K=4)

(b) Kết quả phân cụm với $K=4$: Nhóm “Khách hàng nhỏ” và “Thường xuyên” tách biệt rõ hơn.

Hình 61: So sánh hồ sơ khách hàng trên biểu đồ Radar.
Việc tăng số cụm từ $K=3$ lên $K=4$ giúp nhận diện chi tiết hơn các hành vi mua sắm của người tiêu dùng.

Dựa trên kết quả phân tích định lượng, ta có các quan sát quan trọng sau:

1. **Sự ổn định của nhóm premium (Cluster 0):** Dù ở $K = 3$ hay $K = 4$, nhóm này vẫn giữ nguyên đặc tính với mức giá trung bình (UnitPrice) cao vượt trội (6.35 so với mức trung bình 2.6 – 3.5). Đây là nhóm khách hàng High-margin (biên lợi nhuận cao), ít nhạy cảm về giá nhưng số lượng mua chỉ ở mức trung bình.
2. **Sự phân tách của nhóm mua số lượng lớn:** Tại $K = 3$, Cluster 1 chiếm tới 38.6% dữ liệu, gộp chung cả khách hàng doanh nghiệp lớn và các khách hàng mua sỉ nhỏ lẻ. Khi chuyển sang $K = 4$, nhóm này được tách thành hai phân khúc rõ ràng hơn:
 - **Cluster 1 (Mới - 16.5%):** Nhóm khách hàng nhỏ, mua ít hơn, có thể là các hộ kinh doanh bán lẻ.
 - **Cluster 2 (Mới - 33.6%):** Nhóm khách hàng mua sỉ thực thụ với tần suất và số lượng cực lớn.
3. **Nhóm khám phá (Cluster 3):** Đặc trưng bởi sự cân bằng trong đa dạng danh mục sản phẩm nhưng giá trị đơn hàng thấp. Đây là nhóm khách hàng đang trong giai đoạn tìm hiểu sản phẩm.

Việc lựa chọn $K = 4$ mang lại tính Actionability (khả năng hành động) cao hơn cho doanh nghiệp, cho phép thiết kế các chiến lược khuyến mãi riêng biệt cho nhóm số lượng lớn lớn và nhóm hộ kinh doanh nhỏ, thay vì áp dụng chung một chính sách bán sỉ chung chung.

30.4.6 Quy trình định danh khách hàng với ChatGPT

Việc gán nhãn cho các cụm số vô tri (Cluster 0, 1, 2...) thành các chân dung khách hàng có hồn là bước quan trọng để chuyển đổi từ kỹ thuật sang giá trị trong kinh doanh. Đây là lúc chúng ta kết hợp dữ liệu định lượng với “trực giác” của các mô hình ngôn ngữ lớn (LLMs). Thay vì đưa toàn bộ dữ liệu thô, chúng ta cần một bảng tóm tắt các chỉ số trung bình để ChatGPT có thể hiểu dễ dàng hơn bức tranh tổng quát của đề bài. Ta sử dụng code sau để tạo ra bảng markdown chứa các dòng dữ liệu đại diện cho toàn bộ một cluster:

Tạo bảng thống kê trung bình

```

1 cluster_results = analyzer.apply_kmeans([3, 4])
2
3 display(cluster_means_3.round(2).style.background_gradient(cmap='viridis', axis=0))
4 display(cluster_means_4.round(2).style.background_gradient(cmap='viridis', axis=0))

```

Sau khi có được các dòng dữ liệu tiêu biểu nói trên, ta sẽ sử dụng một prompt đơn giản để có thể yêu cầu mô hình LLMs cho ta tên của từng cụm. Một prompt hiệu quả không chỉ đưa dữ liệu, mà phải cung cấp luật và ngữ cảnh cần thiết cho mô hình. Dưới đây là mẫu Prompt chi tiết bạn có thể copy trực tiếp vào ChatGPT/Claude:

Prompt phân tích hồ sơ khách hàng

Role: Bạn là giám đốc marketing và chuyên gia phân tích dữ liệu của một chuỗi bán lẻ online tại UK.

Context: Tôi vừa thực hiện thuật toán K-Means Clustering và chia khách hàng thành 4 nhóm. Dưới đây là định nghĩa các biến: ...

Data: [DÁN BẢNG KẾT QUẢ TỪ BUỐC TRÊN VÀO ĐÂY]

Task: Hãy phân tích từng cụm (Cluster) và thực hiện các yêu cầu sau:

1. **Đặt tên (Naming):** Đặt 1 tên tiếng Anh chuyên nghiệp và 1 tên tiếng Việt dễ nhớ (Ví dụ: “Big Spenders”, “Lost Customers”).
2. **Mô tả (Persona):** Mô tả ngắn gọn hành vi của nhóm này trong 1 câu.
3. **Chiến lược (Action):** Đề xuất 1 hành động marketing cụ thể (Ví dụ: gửi email, upsell, kích cầu...).

Hãy trình bày kết quả dưới dạng bảng Markdown.

AI sẽ cung cấp một góc nhìn tham khảo rất tốt. Tuy nhiên, chúng ta cần đổi chiều lại với biểu đồ Radar Chart nói trên để đảm bảo tên gọi phản ánh đúng thực tế của mẫu dữ liệu, tránh trường hợp ảo giác của mô hình.

Ví dụ kết quả sau khi tinh chỉnh:

- **Cluster 0 - The Whales:** Chi tiêu cực lớn, mua thường xuyên. → Chiến lược: Chăm sóc VIP 1-1, mời tham gia sự kiện lớn.
- **Cluster 2 - Hibernating:** Đã từng mua nhưng lâu không quay lại. → Chiến lược: Gửi email “We miss you” kèm mã giảm giá. Tạo ra chương trình tri ân khách hàng.
- **Cluster 3 - Small Salse:** ...

30.5 Các hướng cải tiến

Mặc dù mô hình K-Means hiện tại đã mang lại những kết quả khả quan trong việc định hình chân dung khách hàng, phương pháp này vẫn tồn tại một số hạn chế như giả định các cụm có dạng hình cầu lồi và kích thước tương đồng. Để nâng cao chất lượng phân khúc và khai thác sâu hơn dữ liệu, ta có thể cân nhắc các hướng tiếp cận nâng cao sau:

1. **Giảm chiều dữ liệu phi tuyến tính:** PCA là thuật toán tuyến tính, có thể bỏ sót các cấu trúc dữ liệu phức tạp. Ta có thể thử nghiệm với t-SNE hoặc UMAP. Các thuật toán này bảo toàn cấu trúc cục bộ tốt hơn, giúp việc phân tách các cụm trở nên rõ ràng hơn trong không gian thấp chiều, đặc biệt khi dữ liệu có tính phi tuyến cao.

Minh họa sử dụng UMAP

```

1 import umap
2
3 # Khởi tạo UMAP reducer
4 reducer = umap.UMAP(
5     n_neighbors=15,
6     min_dist=0.1,
7     metric='euclidean'
8 )
9
10 # Giảm chiều dữ liệu xuống 2D
11 embedding = reducer.fit_transform(df_scaled)

```

2. **Phân cụm dựa trên mật độ (Density-based Clustering):** K-Means buộc phải gán mọi điểm dữ liệu vào một cụm, ngay cả khi đó là

nhiều. DBSCAN hoặc HDBSCAN là những lựa chọn thay thế tốt, cho phép phát hiện các cụm có hình dạng bất kỳ và tự động loại bỏ nhiều, giúp mô hình bền vững hơn trước các dữ liệu bất thường.

3. **Tích hợp thông tin sản phẩm:** Hiện tại, mô hình chủ yếu dựa trên hành vi giao dịch (RFM). Một hướng đi tiềm năng là sử dụng kỹ thuật Xử lý ngôn ngữ tự nhiên (NLP) như TF-IDF hoặc Word2Vec để phân tích mô tả sản phẩm trong lịch sử mua hàng. Điều này giúp gom nhóm khách hàng dựa trên “gu” thẩm mỹ hoặc nhu cầu sử dụng thực tế (ví dụ: nhóm thích đồ trang trí cổ điển so với nhóm thích đồ gia dụng hiện đại), thay vì chỉ dựa trên số tiền họ chi trả.

30.6 Câu hỏi trắc nghiệm

Trắc nghiệm ôn tập: Phân khúc khách hàng & K-Means

1. Đâu là mục tiêu cốt lõi của việc “Phân khúc khách hàng” so với tiếp thị đại trà?
2. Giảm thiểu tối đa chi phí vận hành hệ thống IT của doanh nghiệp.
3. Gửi cùng một email quảng cáo cho tất cả mọi người để tiết kiệm thời gian.
4. Tối ưu hóa Marketing, giữ chân khách hàng và phân bổ nguồn lực hiệu quả.
5. Loại bỏ hoàn toàn sự cần thiết của các nhân viên chăm sóc khách hàng.
6. Tại sao thuật toán K-Means Clustering được đánh giá cao hơn phương pháp phân loại thủ công trong bối cảnh Big Data?
7. Vì nó phụ thuộc nhiều vào cảm tính của người phân loại để đưa ra quyết định nhân văn hơn.
8. Vì nó có khả năng xử lý tốt dữ liệu đa chiều, tính khách quan cao và tốc độ xử lý nhanh.
9. Vì nó chỉ hoạt động tốt trên các tập dữ liệu nhỏ dưới 100 dòng.
10. Vì nó không đòi hỏi dữ liệu đầu vào phải được làm sạch hay chuẩn hóa.
11. Trong bước làm sạch dữ liệu của dự án này, các hóa đơn bắt đầu bằng ký tự nào sẽ bị loại bỏ?
12. Ký tự ‘A’ (Approved).
13. Ký tự ‘R’ (Return).
14. Ký tự ‘C’ (Cancelled).
15. Ký tự ‘F’ (Failed).

16. Tại sao mô hình RFM truyền thống (Recency, Frequency, Monetary) lại được cho là có hạn chế và cần được mở rộng thêm các features khác?
17. Vì RFM quá phức tạp để tính toán trên máy tính cá nhân thông thường.
18. Vì RFM thiếu chiều sâu về độ đa dạng sản phẩm, hành vi trong từng giao dịch và độ nhạy với giá.
19. Vì RFM chỉ áp dụng được cho các cửa hàng vật lý, không dùng được cho online.
20. Vì các chỉ số RFM không thể trực quan hóa được trên biểu đồ.
21. Mục đích chính của việc áp dụng phép biến đổi **Box-Cox** lên dữ liệu trước khi đưa vào mô hình K-Means là gì?
22. Để loại bỏ các giá trị thiếu (missing values) trong dữ liệu.
23. Để chuyển đổi dữ liệu dạng chữ (string) sang dạng số (numeric).
24. Để xử lý vấn đề dữ liệu bị lệch phải (right-skewness) và đưa về dạng gần phân phối chuẩn.
25. Để tăng số lượng dòng dữ liệu lên gấp đôi nhằm huấn luyện tốt hơn.
26. Tại sao trước khi áp dụng hàm `boxcox`, ta cần thực hiện phép tính $x + 1$ trên các giá trị dữ liệu?
27. Để đảm bảo dữ liệu luôn dương tuyệt đối, do toán học của Box-Cox yêu cầu đầu vào dương.
28. Để làm tròn số liệu doanh thu cho dễ nhìn hơn.
29. Để tạo ra sự nhiễu ngẫu nhiên giúp mô hình không bị overfitting.
30. Để chuyển đổi đơn vị tiền tệ từ Bảng Anh sang USD.
31. Hai phương pháp nào được sử dụng trong bài để xác định số cụm K tối ưu?
32. Random Forest và Decision Tree.
33. Elbow Method và Silhouette Score.

34. Gradient Descent và Backpropagation.
35. Mean Squared Error và R-squared.
36. Khi so sánh kết quả phân cụm giữa $K = 3$ và $K = 4$, sự khác biệt lớn nhất mang lại tính hành động cao hơn là gì?
37. Nhóm khách hàng VIP (Whales) bị biến mất hoàn toàn.
38. Nhóm mua số lượng lớn (vốn gộp chung ở $K=3$) được tách thành hai nhóm: hộ kinh doanh nhỏ và nhà bán sỉ thực thụ.
39. Tất cả các nhóm đều bị trộn lẫn vào nhau khiến việc phân biệt trở nên khó khăn hơn.
40. Nhóm khách hàng vãng lai chiếm 90% dữ liệu.
41. Biểu đồ nào được sử dụng để trực quan hóa “hồ sơ” (profile) hành vi của từng nhóm khách hàng trên đa chiều dữ liệu?
42. Biểu đồ Pie Chart.
43. Biểu đồ Radar (Spider Chart).
44. Biểu đồ Line Chart theo thời gian.
45. Biểu đồ Gantt Chart.
46. Để khắc phục hạn chế của K-Means (giả định cụm hình cầu) và xử lý dữ liệu phi tuyến tính tốt hơn, bài viết đề xuất hướng cải tiến nào?
47. Quay lại sử dụng phương pháp phân loại thủ công bằng Excel.
48. Sử dụng thuật toán giảm chiều dữ liệu t-SNE/UMAP và phân cụm dựa trên mật độ (DBSCAN).
49. Chỉ sử dụng duy nhất chỉ số Monetary để phân nhóm.
50. Tăng số lượng cụm K lên 100 để chi tiết hóa tối đa.

1. **Code:** Các file code được đề cập trong bài có thể được tải tại [đây](#).
2. **Q&A:** Bạn có thể đặt thêm câu hỏi về nội dung bài đọc trong group Facebook hỏi đáp tại [đây](#). Tất cả câu hỏi sẽ được trả lời trong vòng 3-4 tiếng.

AIO_QAs-Verified

🔒 Private group · 1.4K members



Hình 62: Hình ảnh group facebook AIO Q&A.

Phần VII

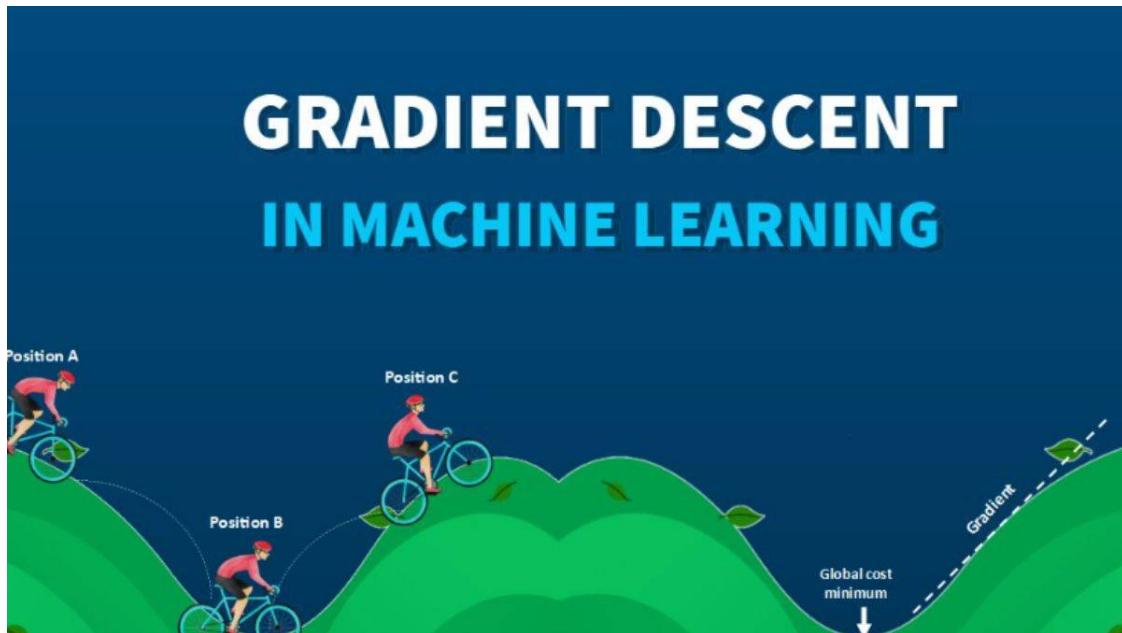
Module 7: Hiểu rõ cách hoạt động của Deep MLP

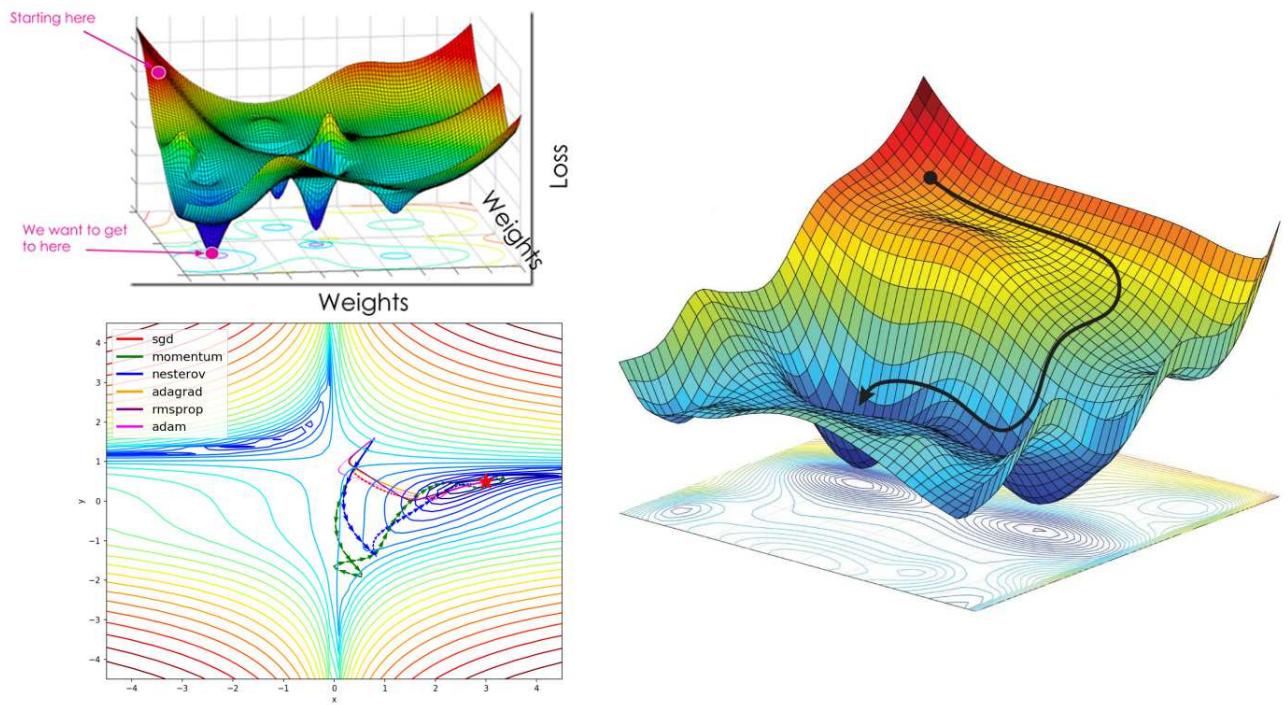
Chương 31

Bước Initializer và Optimizer cho mô hình deep learning

31.1 Mô Tả Lý Thuyết

Bài tập tuần này sẽ xoay quanh việc thực hiện các thuật toán optimization cơ bản trong deep learning. Ngoài ra bài tập tuần này cũng giới thiệu một vài nhánh optimization trong deep learning.





Bài tuần này chúng ta sẽ làm quen với 4 thuật toán optimization phổ biến trong deep learning:

- 1. Gradient Descent:** Là thuật toán đi tìm điểm minimum của một function dựa trên tính toán gradient của function đó. Function này phải thỏa mãn điều kiện là liên tục và có thể đạo hàm được phần lớn ở mọi nơi. Phương pháp này ta sẽ chọn một điểm xuất phát ngẫu nhiên (1 điểm bất kỳ trên function) sau đó dựa vào độ dốc và hướng của gradient để đi ngược hướng và tiến về nơi có vị trí thấp hơn, việc này được lặp đi lặp lại cho đến khi tìm được điểm minimum hoặc thỏa mãn điều kiện dừng

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \dots$ **Require:** Initial parameter θ $k \leftarrow 1$ **while** stopping criterion not met **do** Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$. Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$ Apply update: $\theta \leftarrow \theta - \epsilon_k \hat{\mathbf{g}}$ $k \leftarrow k + 1$ **end while**

2. **Gradient Descent + Momentum:** Phương pháp này khắc phục điểm yếu của Gradient Descent là không thể vượt qua local minimum để tiến về điểm local minimum khác thấp hơn. Gradient Descent + Momentum sử dụng exponentially weighted average gradient để tích lũy gradient từ trước đó và kết hợp với gradient hiện tại giúp tạo ra một giá trị đủ lớn vượt qua local minimum hiện tại và tiến về local minimum tốt hơn. Nó hoạt động tương tự như viên bi có khi lăn xuống kết hợp thêm đà (momentum) để vượt qua dốc và rơi vào nơi tốt hơn.

3. RMSProp:

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ , decay rate ρ **Require:** Initial parameter θ **Require:** Small constant δ , usually 10^{-6} , used to stabilize division by small numbersInitialize accumulation variables $r = 0$ **while** stopping criterion not met **do** Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$. Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$. Accumulate squared gradient: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$. Compute parameter update: $\Delta \theta = -\frac{\epsilon}{\sqrt{\delta + r}} \odot \mathbf{g}$. ($\frac{1}{\sqrt{\delta + r}}$ applied element-wise) Apply update: $\theta \leftarrow \theta + \Delta \theta$.**end while**

Đây là thuật toán mở rộng từ AdaGrad và cũng sử dụng gradient từ trước để tạo ra một thuật toán có thể adaptive learning rate thay vì learning rate cố định như các thuật toán trước. Ý tưởng chính của RMSProp là sử dụng exponentially weighted average bình phương gradient và chia gradient với căn bậc hai của average này.

4. **Adam:** Có thể xem Adam là thuật toán sử dụng kết hợp ý tưởng của Momentum và RMSProp vì Adam sử dụng exponentially weighted average gradient và exponentially weighted average bình phương gradient. Cách thức Adam hoạt động theo việc thừa hưởng điểm mạnh của momentum và adaptive learning rate, momentum giúp vượt qua các local minimum để tiến đến các local minimum tốt hơn, trong khi adaptive learning rate giúp cho việc hội tụ nhanh hơn thay vì phải mất một khoảng thời gian để dao động xung quanh điểm hội tụ.

Algorithm 1: Adam, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t .

Require: α : Stepsize
Require: $\beta_1, \beta_2 \in [0, 1]$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0 : Initial parameter vector
 $m_0 \leftarrow 0$ (Initialize 1st moment vector)
 $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
 $t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
 $t \leftarrow t + 1$
 $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
 $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
 $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
 $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
 $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
 $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

31.2 Bài Tập và Gợi Ý

1) Gợi Ý Các Bước Làm Bài:

1. **Giới thiệu sơ lược về bài tập:** Bài tập tuần này sẽ có gồm có các dạng chính như tính toán và trình bày những bước trong các thuật toán optimization, thực hiện code bằng numpy cho các thuật toán optimization. Ngoài ra các bài tập dạng optional sẽ thực hiện thay đổi các thuật toán optimization trên Pytorch và quan sát hiện tượng vanishing được khắc phục qua các thuật toán (Chi tiết xem phần Yêu cầu bài tập)

Các công thức và gợi ý dưới đây được rút gọn và làm đơn giản hơn để các bạn thuận tiện làm bài. Bài tập 1,2,3 và 4 sẽ optimize cho cho function gồm 2 biến như sau :

2. Bài 1 Gradient Descent:

- (a) Yêu cầu trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán Gradient Descent (tìm w_1 và w_2 sau 2 epoch) với epoch = 2.
 - **Bắt đầu** với epoch = 1
 - **STEP1:** Tìm giá trị dw_1 và dw_2 giá trị đạo hàm của hàm (1) theo w_1 và w_2 (partial derivative) tại điểm khởi tạo $[w_1, w_2]$
 - **STEP2:** Dùng công thức Gradient Descent (1.1) để cập nhật w_1 và w_2 . Hoàn thành epoch = 1
 - **STEP3:** epoch = 2 ta thực hiện tương tự với STEP1 và STEP2 như trên với w_1 và w_2 đã được cập nhật từ epoch = 1
- (b) Thực hiện code dùng numpy để tìm điểm minimum theo thuật toán Gradient Descent (tìm w_1 và w_2) với epoch = 30

```
1 def df_w(W):
2     """
3     Thực hiện tính gradient của dw1 và dw2
4     Arguments:
5     W -- np.array [w1, w2]
6     Returns:
7     dW -- np.array [dw1, dw2], array chứa giá trị đạo hàm theo w1 và w2
8     """
9     ##### YOUR CODE HERE #####
10
11
12     dW =
13     #####
14
15     return dW
```



```
1 def sgd(W, dW, lr):
2     """
3     Thực hiện thuật toán Gradient Descent để update w1 và w2
4     Arguments:
5     W -- np.array: [w1, w2]
6     dW -- np.array: [dw1, dw2], array chứa giá trị đạo hàm theo w1 và w2
7     lr -- float: learning rate
8     Returns:
9     W -- np.array: [w1, w2] w1 và w2 sau khi đã update
10    """
11    ##### YOUR CODE HERE #####
12
13
14    W =
15    #####
16    return W
```

Hình 31.1: Function thực hiện tính gradient và function thực hiện thuật toán GD

```

1 def train_p1(optimizer, lr, epochs):
2     """
3     Thực hiện tìm điểm minimum của function (1) dựa vào thuật toán
4     được truyền vào từ optimizer
5     Arguments:
6     optimize : function thực hiện thuật toán optimization cụ thể
7     lr -- float: learning rate
8     epoch -- int: số lượng lần (epoch) lặp để tìm điểm minimum
9     Returns:
10    results -- list: list các cặp điểm [w1, w2] sau mỗi epoch (mỗi lần cập nhật)
11    """
12
13    # initial point
14    W = np.array([-5, -2], dtype=np.float32)
15    # list of results
16    results = [W]
17    ##### YOUR CODE HERE #####
18    # Tao vòng lặp theo số lần epochs
19    # tìm gradient dW gồm dw1 và dw2
20    # dùng thuật toán optimization cập nhật w1 và w2
21    # append cặp [w1, w2] vào list results
22
23
24    #####
25    return results

```

Hình 31.2: Function thực hiện train tìm điểm minimum theo thuật toán GD

- **STEP1:** Tại function `df_w`, các bạn tìm đạo hàm của function (1) theo w_1 và w_2 (partial derivative). Tiếp theo các bạn dùng w_1 , w_2 lấy từ input `W` thế vào hai function vừa đạo hàm sẽ thu được dw_1 và dw_2 . Cuối cùng đưa $[dw_1, dw_2]$ vào một array và gán vào biến `W` để return.
- **STEP2:** Xây dựng hàm `sgd` theo công thức Gradient Descent (1.1) để cập nhật w_1 và w_2 . Sau khi thu được w_1 và w_2 mới, ta sẽ gán vào biến `W` (array chứa $[w_1, w_2]$ mới) để return
- **STEP3:** Xây dựng hàm `train_p1` nhận 3 input function `optimize`, learning rate và số lượng epoch để tìm điểm minimum của function (1). Đầu tiên khởi tạo điểm đầu tiên với $w_1 = -5$ và $w_2 = -2$ (line 14, hình 31.2), tạo list `results` chứa các điểm được cập nhật qua từng epoch (bao gồm cả điểm khởi tạo) (line 16, hình 31.2). Tạo một vòng lặp theo số lần epoch. Trong mỗi epoch, gọi function `df_w` để tìm gradient, rồi gọi

hàm sgd để thực hiện cập nhật w_1 và w_2 . Cuối cùng append cập w_1 và w_2 sau mỗi lần cập nhật vào list *results*

3. Bài 2 Gradient Descent + Momentum:

- (a) Yêu cầu trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán Gradient Descent + Momentum (tìm w_1 và w_2 sau 2 epoch) với epoch = 2.
 - Bắt đầu với epoch = 1
 - **STEP1:** Tìm giá trị dw_1 và dw_2 giá trị đạo hàm của hàm (1) theo w_1 và w_2 (partial derivative) tại điểm khởi tạo $[w_1, w_2]$
 - **STEP2:** Tìm giá trị v_1 và v_2 dựa vào dw_1 và dw_2 tìm được ở step 1 (công thức (2.1)).
 - **STEP3:** Dùng công thức (2.2) Gradient Descent + Momentum để cập nhật w_1 và w_2 . Hoàn thành epoch = 1
 - **STEP4:** epoch = 2 ta thực hiện tương tự với STEP1, STEP2 và STEP3 như trên với w_1 và w_2 đã được cập nhật từ epoch = 1
- (b) Thực hiện code dùng numpy để tìm điểm minimum theo thuật toán Gradient Descent + Momentum (tìm w_1 và w_2) với epoch = 30. Các bạn nên tham khảo file hint trước.
 - **STEP1:** Tương tự như STEP1 của Bài 1 Gradient Descent. Tại function *df_w*, các bạn tìm đạo hàm của function (1) theo w_1 và w_2 (partial derivative) và return về *W* là array chứa $[dw_1, dw_2]$
 - **STEP2:** Xây dựng hàm *sgd_momentum* theo công thức (2.1) và (2.2) Gradient Descent + Momentum để cập nhật v_1 , v_2 , w_1 và w_2 . Sau đó ta sẽ gán vào biến *V* (array chứa $[v_1, v_2]$ mới) và *W* (array chứa $[w_1, w_2]$ mới) để return
 - **STEP3:** Xây dựng hàm *train_p1* nhận 3 input function *optimize*, learning rate và số lượng epoch để tìm điểm minimum của function (1). Đầu tiên khởi tạo điểm đầu tiên với $w_1 = -5$ và $w_2 = -2$, khởi tạo $v_1 = 0$ và $v_2 = 0$, tạo list *results* chứa các điểm $[w_1, w_2]$ được cập nhật qua từng epoch (bao gồm cả điểm khởi tạo). Tạo một vòng lặp theo số lần epoch. Trong mỗi epoch, gọi function *df_w* để tìm gradient, rồi gọi hàm

sgd_momentum để thực hiện cập nhật v_1 , v_2 , w_1 và w_2 . Cuối cùng append cặp w_1 và w_2 sau mỗi lần cập nhật vào list $results$

4. Bài 3 RMSProp:

- (a) Yêu cầu trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán RMSProp (tìm w_1 và w_2 sau 2 epoch) với epoch = 2.
 - Bắt đầu với epoch = 1
 - **STEP1:** Tìm giá trị dw_1 và dw_2 giá trị đạo hàm của hàm (1) theo w_1 và w_2 (partial derivative) tại điểm khởi tạo $[w_1, w_2]$
 - **STEP2:** Tìm giá trị s_1 và s_2 dựa vào dw_1 và dw_2 vừa tìm được ở step 1 (công thức (3.1)).
 - **STEP3:** Dùng công thức (3.2) để cập nhật w_1 và w_2 . Hoàn thành epoch = 1
 - **STEP4:** epoch = 2 ta thực hiện tương tự với STEP1, STEP2 và STEP3 như trên với w_1 và w_2 đã được cập nhật từ epoch = 1
- (b) Thực hiện code dùng numpy để tìm điểm minimum theo thuật toán RMSProp (tìm w_1 và w_2) với epoch = 30. Các bạn nên tham khảo file hint trước.
 - **STEP1:** Tương tự như STEP1 của Bài 1 Gradient Descent. Tại function df_w , các bạn tìm đạo hàm của function (1) theo w_1 và w_2 (partial derivative) và return về W là array chứa $[dw_1, dw_2]$
 - **STEP2:** Xây dựng hàm RMSProp theo công thức (3.1) và (3.2) để cập nhật s_1 , s_2 , w_1 và w_2 . Sau đó ta sẽ gán vào biến S (array chứa $[s_1, s_2]$ mới) và W (array chứa $[w_1, w_2]$ mới) để return
 - **STEP3:** Xây dựng hàm $train_p1$ nhận 3 input function optimize, learning rate và số lượng epoch để tìm điểm minimum của function (1). Đầu tiên khởi tạo điểm đầu tiên với $w_1 = -5$ và $w_2 = -2$, khởi tạo $s_1 = 0$ và $s_2 = 0$, tạo list $results$ chứa các điểm $[w_1, w_2]$ được cập nhật qua từng epoch (bao gồm cả điểm khởi tạo). Tạo một vòng lặp theo số lần epoch. Trong mỗi epoch, gọi function df_w để tìm gradient, rồi gọi hàm

RMSProp để thực hiện cập nhật s_1 , s_2 , w_1 và w_2 . Cuối cùng append cặp w_1 và w_2 sau mỗi lần cập nhật vào list *results*

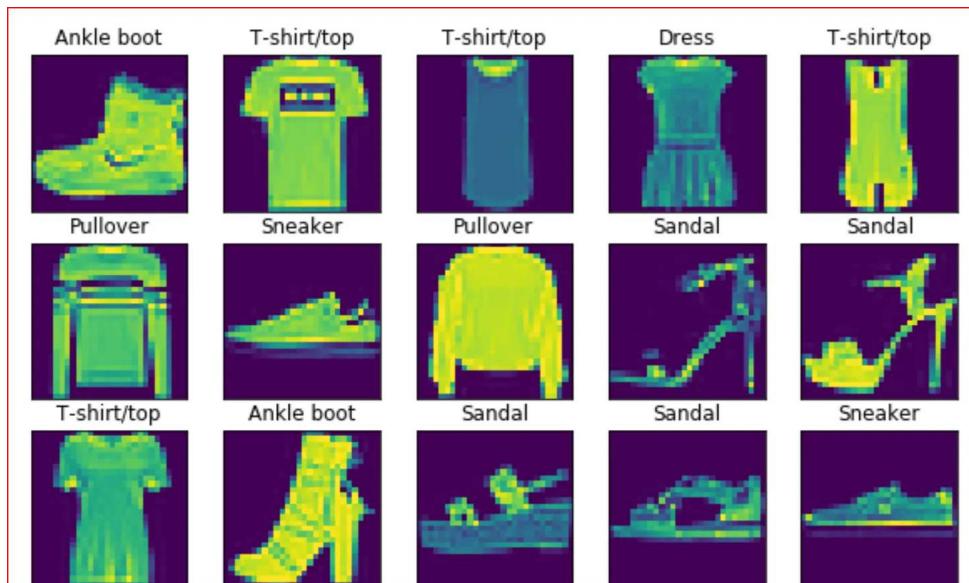
5. Bài 4 Adam:

- (a) Yêu cầu trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán Adam (tìm w_1 và w_2 sau 2 epoch) với epoch = 2.
 - Bắt đầu với epoch = 1
 - **STEP1:** Tìm giá trị dw_1 và dw_2 giá trị đạo hàm của hàm (1) theo w_1 và w_2 (partial derivative) tại điểm khởi tạo $[w_1, w_2]$
 - **STEP2:** Tìm giá trị v_1 và v_2 dựa vào dw_1 và dw_2 vừa tìm được ở step 1 (công thức ở (4.1)).
 - **STEP3:** Tìm giá trị s_1 và s_2 dựa vào dw_1 và dw_2 vừa tìm được ở step 1 (công thức (4.2)).
 - **STEP4:** Thực hiện bias-correction cho V và S để thu được V_{coor} và S_{coor} . Sau khi áp dụng bias_correction (4.3) và (4.4) ta sẽ thu được v_{coor1} , v_{coor2} , s_{coor1} và s_{coor2}
 - **STEP5:** Dùng công thức (4.5) để cập nhật w_1 và w_2 . Hoàn thành epoch = 1
 - **STEP6:** epoch = 2 ta thực hiện tương tự với STEP1 - STEP5 như trên với w_1 và w_2 đã được cập nhật từ epoch = 1
- (b) Thực hiện code dùng numpy để tìm điểm minimum theo thuật toán Adam (tìm w_1 và w_2) với epoch = 30. Các bạn nên tham khảo file hint trước.
 - **STEP1:** Tương tự như STEP1 của Bài 1 Gradient Descent. Tại function df_w, các bạn tìm đạo hàm của function (1) theo w_1 và w_2 (partial derivative) và return về W là array chứa $[dw_1, dw_2]$
 - **STEP2:** Xây dựng hàm Adam theo công thức (4.1), (4.2), (4.3), (4.4) và (4.5) để cập nhật v_1 , v_2 , s_1 , s_2 , w_1 và w_2 . Sau đó ta sẽ gán vào biến V (array chứa $[v_1, v_2]$ mới), S (array chứa $[s_1, s_2]$ mới) và W (array chứa $[w_1, w_2]$ mới) để return
 - **STEP3:** Xây dựng hàm train_p1 nhận 3 input function optimize, learning rate và số lượng epoch để tìm điểm minimum của function (1). Đầu tiên khởi tạo điểm đầu tiên với $w_1 = -5$ và $w_2 = -2$, khởi tạo $v_1 = 0$ và $v_2 = 0$, khởi tạo $s_1 = 0$ và

$s_2 = 0$, tạo list *results* chứa các điểm $[w_1, w_2]$ được cập nhật qua từng epoch (bao gồm cả điểm khởi tạo). Tạo một vòng lặp theo số lần epoch. Trong mỗi epoch, gọi function *df_w* để tìm gradient, rồi gọi hàm Adam để thực hiện cập nhật v_1, v_2, s_1, s_2, w_1 và w_2 . Cuối cùng append cặp w_1 và w_2 sau mỗi lần cập nhật vào list *results*

6. Bài 5 Vanishing Problem (Optional): Các bạn sẽ nhận được 1 file code để train một MLP model phân loại 10 class thời trang trên tập FashionMNIST. Model đã được thiết kế để xảy ra vấn đề Vanishing. Nhiệm vụ của các bạn là sẽ thay thế các optimizer đã làm ơ trên quan sát và đánh giá performance của các optimizer với Vanishing như thế nào

FashionMNIST là một tập dữ liệu hình ảnh bài viết của Zalando, được giới thiệu như một lựa chọn thách thức hơn so với tập dữ liệu MNIST truyền thống để đánh giá các thuật toán học máy. Nó được thiết kế để phục vụ như một sự thay thế trực tiếp cho MNIST gốc. Mỗi sample trong tập dữ liệu FashionMNIST là một hình ảnh xám 28x28, liên kết với một trong 10 class thời trang như áo sơ mi/áo phông, ... Có 60.000 hình ảnh cho tập train và 10.000 hình ảnh cho tập test.



Hình 31.3: Sample FashionMNIST Data

Load Data:

- Cell 1

- Dòng 1: Khởi tạo biến batch_size với giá trị 512. Đây chỉ định số lượng mẫu dữ liệu sẽ được đưa qua mạng trong mỗi lần cập nhật trọng số.
- Dòng 2: Khởi tạo biến num_epochs với giá trị 300.
- Dòng 3: Khởi tạo biến lr (learning rate) với giá trị 0.01. Đây là một tham số quan trọng chỉ định bước học hoặc tốc độ mà model cập nhật trọng số của nó dựa trên gradient của hàm mất mát.

- Cell 2:

- Dòng 1: Khởi tạo tập dữ liệu train_dataset sử dụng tập dữ liệu FashionMNIST. Các hình ảnh sẽ được lưu trữ trong thư mục './data', tải về nếu chưa có và biến đổi thành tensor.
- Dòng 3: Khởi tạo train_loader, một bộ tải dữ liệu, sử dụng train_dataset với batch size đã chỉ định và có tùy chọn xáo trộn dữ liệu.
- Dòng 4: Khởi tạo tập test_dataset tương tự như train_dataset nhưng đặt tham số train thành False để chỉ định rằng đây là tập test.
- Dòng cuối: Khởi tạo test_loader sử dụng test_dataset với batch size đã chỉ định và không có tùy chọn xáo trộn dữ liệu (mặc định là shuffle=False).

```
[ ] 1 batch_size = 512
2 num_epochs = 300
3 lr = 0.01

[ ] 1 train_dataset = FashionMNIST(root='./data', train=True,
2                                     download=True, transform=transforms.ToTensor())
3 train_loader = DataLoader(train_dataset, batch_size, shuffle=True)
4 test_dataset = FashionMNIST(root='./data', train=False,
5                               download=True, transform=transforms.ToTensor())
6 test_loader = DataLoader(test_dataset, batch_size)
```

Hình 31.4: Load Data

Build và Compile Model

- Cell 1

- (a) Trong cell này, chúng ta định nghĩa một lớp MLP kế thừa từ nn.Module, một lớp cơ bản trong thư viện PyTorch dùng để xây dựng các model học sâu.
- (b) Trong phương thức `__init__`, chúng ta khởi tạo mạng với: Một lớp đầu vào layer1 chuyển đổi từ số chiều `input_dims` đến `hidden_dims`. Bốn lớp ẩn layer2, layer3, layer4, và layer5 mỗi lớp chuyển đổi từ số chiều `hidden_dims` đến `hidden_dims`. Một lớp đầu ra output chuyển đổi từ `hidden_dims` đến `output_dims`. Hàm kích hoạt sigmoid được sử dụng sau mỗi lớp.
- (c) Phương thức forward xác định cách dữ liệu được truyền qua mạng. Đầu tiên, dữ liệu được làm phẳng bằng `nn.Flatten()`, sau đó truyền qua mỗi lớp và hàm kích hoạt sigmoid. Cuối cùng, dữ liệu được truyền qua lớp đầu ra và kết quả trả về.

- Cell 2

- (a) Khởi tạo model MLP với số chiều đầu vào là 784 (tương đương với hình ảnh 28x28 pixel), số chiều của lớp ẩn là 128, và số chiều đầu ra là 10 (đại diện cho 10 lớp trong tập dữ liệu FashionMNIST).
- (b) Khởi tạo hàm mất mát criterion (loss) sử dụng hàm mất mát CrossEntropyLoss, thường được sử dụng cho các bài toán phân loại nhiều lớp.
- (c) Khởi tạo thuật toán tối ưu hóa optimizer sử dụng SGD (Stochastic Gradient Descent) với tốc độ học lr và các tham số của model.

NOTE: Ở đây sử dụng optimizer là SGD các bạn sẽ thay đổi các optimizer theo yêu cầu của đề bài

```
 1 class MLP(nn.Module):
 2     def __init__(self, input_dims, hidden_dims, output_dims):
 3         super(MLP, self).__init__()
 4         self.layer1 = nn.Linear(input_dims, hidden_dims)
 5         self.layer2 = nn.Linear(hidden_dims, hidden_dims)
 6         self.layer3 = nn.Linear(hidden_dims, hidden_dims)
 7         self.layer4 = nn.Linear(hidden_dims, hidden_dims)
 8         self.layer5 = nn.Linear(hidden_dims, hidden_dims)
 9         self.output = nn.Linear(hidden_dims, output_dims)
10         self.sigmoid = nn.Sigmoid()
11
12     def forward(self, x):
13         x = nn.Flatten()(x)
14         x = self.layer1(x)
15         x = self.sigmoid(x)
16         x = self.layer2(x)
17         x = self.sigmoid(x)
18         x = self.layer3(x)
19         x = self.sigmoid(x)
20         x = self.layer4(x)
21         x = self.sigmoid(x)
22         x = self.layer5(x)
23         x = self.sigmoid(x)
24         out = self.output(x)
25
26         return out
27
28
[ ] 1 model = MLP(input_dims=784, hidden_dims=128, output_dims=10).to(device)
2 criterion = nn.CrossEntropyLoss()
3 optimizer = optim.SGD(model.parameters(), lr=lr)
```

Hình 31.5: Build và Compile Model

Train và Evaluate:

- Khởi tạo list: Bốn list train_losses, train_acc, val_losses, và val_acc được tạo ra để lưu trữ giá trị mất mát và độ chính xác của model trên tập huấn luyện và tập kiểm thử sau mỗi kỳ (epoch).
- Train: Vòng lặp for chạy qua số lượng epoch đã xác định trước (num_epochs). Trong mỗi epoch: model được chuyển sang chế độ huấn luyện (model.train()).

- Một vòng lặp bên trong chạy qua tất cả các batch dữ liệu trong train_loader.
- Gradient của các tham số model được đặt về 0.
- Model tiến hành dự đoán và mất mát được tính toán.
- Gradient được tính toán thông qua phương thức .backward().
- Các tham số model được cập nhật bằng thuật toán của optimizer.
- Giá trị mất mát và độ chính xác của model trên tập huấn luyện được log lại.
- Evaluate: Sau khi huấn luyện xong trên tất cả các batch:
 - model được chuyển sang chế độ đánh giá (model.eval()).
 - Test data được đưa qua model.
 - Mất mát và độ chính xác trên tập test được log lại.
- Lưu trữ và in kết quả: Giá trị mất mát trung bình và độ chính xác trung bình cho tập huấn luyện và tập kiểm thử được thêm vào các list tương ứng.

```
1 train_losses = []
2 train_acc = []
3 val_losses = []
4 val_acc = []
5 for epoch in range(num_epochs):
6     model.train()
7     t_loss = 0
8     t_acc = 0
9     cnt = 0
10    for X, y in train_loader:
11        X, y = X.to(device), y.to(device)
12        optimizer.zero_grad()
13        outputs = model(X)
14        loss = criterion(outputs, y)
15        loss.backward()
16        optimizer.step()
17        t_loss += loss.item()
18        t_acc += (torch.argmax(outputs, 1) == y).sum().item()
19        cnt += len(y)
20    t_loss /= len(train_loader)
21    train_losses.append(t_loss)
22    t_acc /= cnt
23    train_acc.append(t_acc)
24
25    model.eval()
26    v_loss = 0
27    v_acc = 0
28    cnt = 0
29    with torch.no_grad():
30        for X, y in test_loader:
31            X, y = X.to(device), y.to(device)
32            outputs = model(X)
33            loss = criterion(outputs, y)
34            v_loss += loss.item()
35            v_acc += (torch.argmax(outputs, 1)==y).sum().item()
36            cnt += len(y)
37    v_loss /= len(test_loader)
38    val_losses.append(v_loss)
39    v_acc /= cnt
40    val_acc.append(v_acc)
```

Hình 31.6: Train và Evaluate

2) Yêu Cầu Bài Tập:

1. **Gradient Descent:** Dựa vào thuật toán Gradient Descent các bạn thực hiện tìm điểm minimum của function (1) với các tham số sau khởi tạo $w_1 = -5, w_2 = -2, \alpha = 0.4$:
 - (a) Các bạn trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán Gradient Descent với epoch = 2 (tìm w_1 và w_2 sau 2 epoch). Các bạn có thể dùng latex hoặc doc.
 - (b) Các bạn thực hiện code dùng numpy để tìm điểm minimum theo thuật toán Gradient Descent với epoch = 30 (tìm w_1 và w_2 sau 30 epoch).
2. **Gradient Descent + Momentum:** Dựa vào thuật toán Gradient Descent và momentum các bạn thực hiện tìm điểm minimum của function (1) với các tham số sau khởi tạo $w_1 = -5, w_2 = -2, v_1 = 0, v_2 = 0, \alpha = 0.6, \beta = 0.5$:
 - (a) Các bạn trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán Gradient Descent + Momentum với epoch = 2 (tìm w_1 và w_2 sau 2 epoch). Các bạn có thể dùng latex hoặc doc.
 - (b) Các bạn thực hiện code dùng numpy để tìm điểm minimum theo thuật toán Gradient Descent + Momentum với epoch = 30 (tìm w_1 và w_2 sau 30 epoch).
3. **RMSProp:** Dựa vào thuật toán RMSProp các bạn thực hiện tìm điểm minimum của function (1) với các tham số sau khởi tạo $w_1 = -5, w_2 = -2, s_1 = 0, s_2 = 0, \alpha = 0.3, \gamma = 0.9, \epsilon = 10^{-6}$:
 - (a) Các bạn trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán RMSProp với epoch = 2 (tìm w_1 và w_2 sau 2 epoch). Các bạn có thể dùng latex hoặc doc.
 - (b) Các bạn thực hiện code dùng numpy để tìm điểm minimum theo thuật toán RMSProp với epoch = 30 (tìm w_1 và w_2 sau 30 epoch).
4. **Adam:** Dựa vào thuật toán Adam các bạn thực hiện tìm điểm minimum của function (1) với các tham số sau khởi tạo $w_1 = -5, w_2 = -2, \text{initial } v_1 = 0, v_2 = 0, s_1 = 0, s_2 = 0, \alpha = 0.2, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-6}$:

- (a) Các bạn trình bày chi tiết từng bước thực hiện tìm điểm minimum theo thuật toán Adam với epoch = 2 (tìm w_1 và w_2 sau 2 epoch). Các bạn có thể dùng latex hoặc doc.
- (b) Các bạn thực hiện code dùng numpy để tìm điểm minimum theo thuật toán Adam với epoch = 30 (tìm w_1 và w_2 sau 30 epoch).
5. **Vanishing Problem (Optional):** Bài tập này yêu cầu các bạn thay đổi các optimizer và quan sát sự giảm thiểu vấn đề vanishing của từng loại thuật toán. Một model được xây dựng theo các yêu cầu như sau: weight được khởi tạo random theo normal distribution ($\mu = 0, \sigma = 0.05$), Loss = Cross entropy, Optimizer = SGD, Hidden layers = 5, Nodes mỗi layer = 128, Activation = Sigmoid. Các bạn sẽ sử dụng các thuật toán như sau với model trên:
- (a) Gradient Descent
 - (b) Gradient Descent + Momentum
 - (c) RMSProp
 - (d) Adam
 - (e) ADOPT

Phần III: Trắc Nghiệm:

- (A). $-3.00577e-04, -3.005e-18$ (B). $-3.00577e-03, -3.005e-17$
(C). $-3.00577e-02, -3.005e-16$ (D). $-3.00577e-01, -3.005e-15$

13. Với bài tập **Adam**: câu (a) tại **epoch = 2** sau khi áp dụng Adam ta thu được w_1 và w_2 là (Kết quả có thể không chính xác hoàn toàn các bạn chọn kết quả làm tròn gần nhất):

(A). $-4.6002546, -1.6008245$ (B). $-5.6002546, -2.6008245$
(C). $-6.6002546, -3.6008245$ (D). $-7.6002546, -4.6008245$

14. Với bài tập **Adam**: câu (b) tại **epoch cuối cùng** sau khi áp dụng Adam ta thu được w_1 và w_2 là (Kết quả có thể không chính xác hoàn toàn các bạn chọn kết quả làm tròn gần nhất):

(A). $-0.71, 0.0679$ (B). $-0.51, 0.0679$
(C). $-0.11, 0.0679$ (D). $-0.31, 0.0679$

Phần IV: Phụ Lục

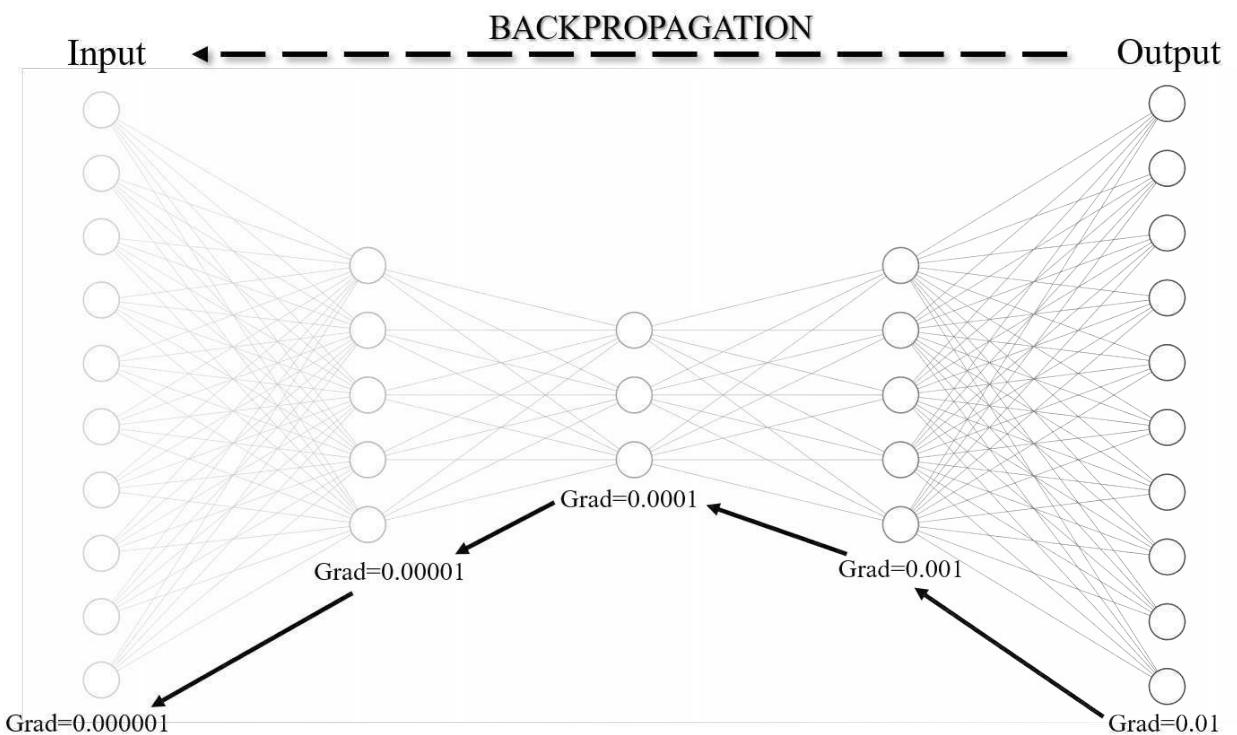
1. **Hint:** Các file code gợi ý có thể tải tại [Link](#)
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể tải tại [Link](#) (**Lưu ý** Sáng thứ 3 khi hết deadline phần bài tập ad mới copy các nội dung bài giải nêu trên vào đường dẫn)
3. **Rubric:**

Chương 32

Project 1: Vấn đề gradient vanishing và các giải pháp

32.1 Giới thiệu

Vanishing Gradient là một vấn đề thường gặp khi tăng cường độ phức tạp của mô hình bằng cách thêm nhiều lớp (layers) nhằm học được các đặc trưng phức tạp hơn từ một tập dữ liệu lớn. Khi mô hình trở nên sâu hơn, trong quá trình lan truyền ngược (backpropagation), giá trị gradient giảm dần qua từng lớp. Điều này dẫn đến việc các trọng số (weights) nhận được rất ít, hoặc thậm chí không có, sự cập nhật sau mỗi vòng lặp, khiến mô hình gần như không thể học được. Vấn đề này được gọi là Vanishing Gradient.

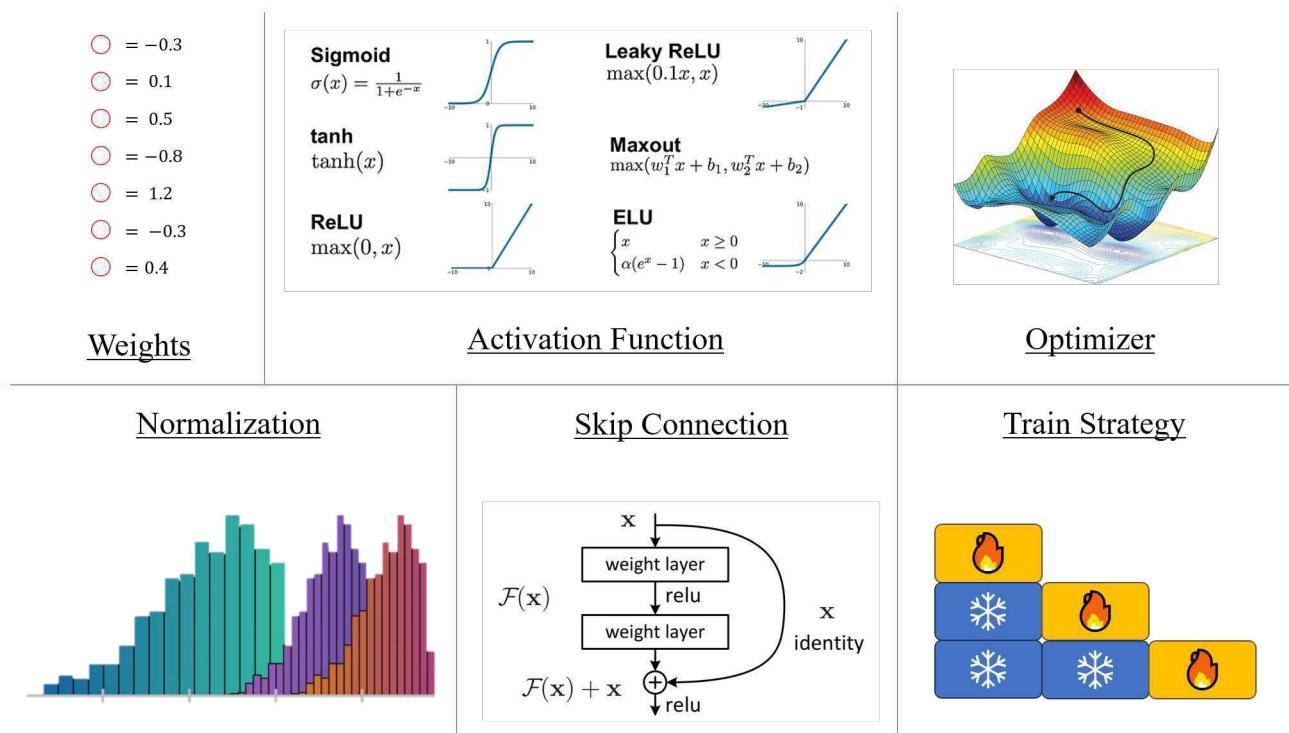


Hình 32.1: Minh họa vấn đề Vanishing Gradient trong một mạng Deep Learning.

Vanishing Gradient thường được nhận diện qua một số dấu hiệu điển hình:

- Trọng số của các layer gần output layer thay đổi đáng kể, trong khi trọng số của các layer gần input layer thay đổi rất ít hoặc hầu như không thay đổi.
- Trọng số có thể dần tiệm cận 0 trong quá trình huấn luyện, làm giảm hiệu quả cập nhật.
- Mô hình học rất chậm, và quá trình huấn luyện có thể bị đình trệ từ rất sớm, thường chỉ sau vài epoch đầu tiên.
- Phân phối của trọng số tập trung quanh giá trị 0, hạn chế khả năng truyền tải thông tin qua các lớp mạng.

Nguyên nhân chính của hiện tượng Vanishing Gradient xuất phát từ quy tắc chuỗi (chain rule) trong quá trình lan truyền ngược. Gradient tại các layer gần input layer giảm dần theo tích gradient qua các lớp trước đó, dẫn đến gần như bằng 0 trong các mạng sâu. Đây là một trong những thách thức lớn nhất trong huấn luyện một deep neural network, đòi hỏi các phương pháp xử lý hiệu quả để khắc phục.



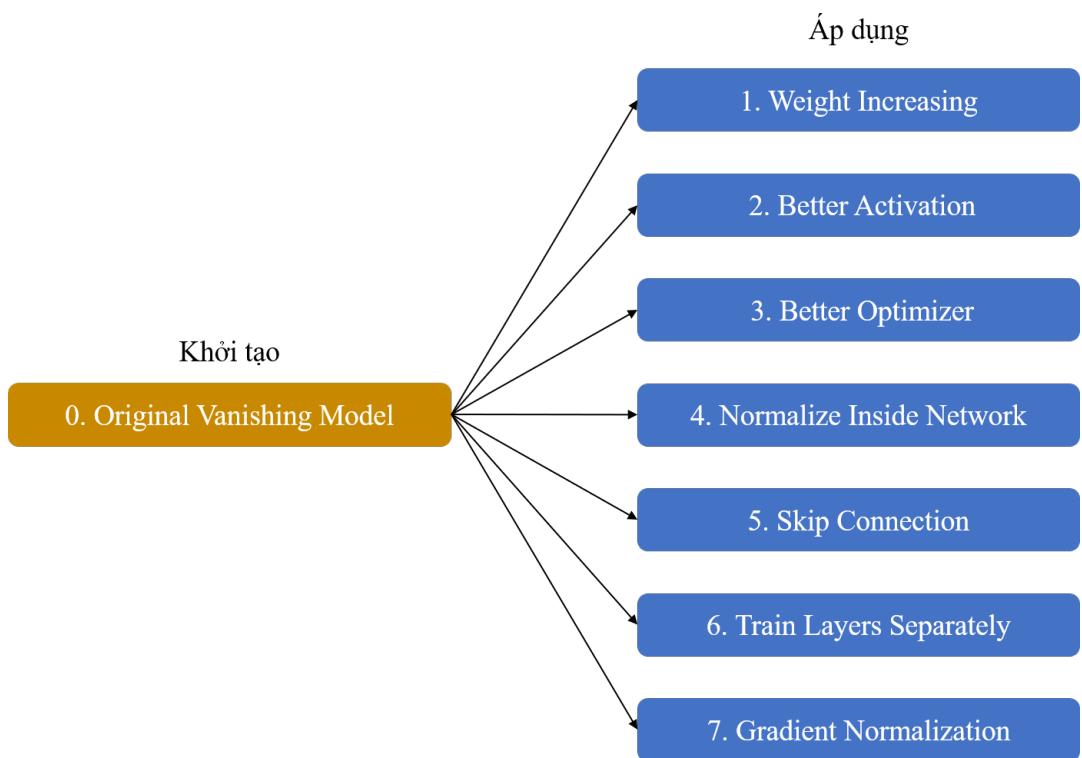
Hình 32.2: Một số phương pháp khắc phục Vanishing Gradient được đề cập trong project. Nguồn ảnh: [Activation Function](#), [Optimizer](#), [Normalization](#), [Skip Connection](#).

Trong project này, chúng ta sẽ tìm hiểu vấn đề Vanishing Gradient trong mạng MLP và thử nghiệm các giải pháp cải tiến hiệu quả học của mô hình. Tổng quan nội dung bao gồm:

- Thiết lập một mạng MLP điển hình dễ bị Vanishing Gradient với các đặc điểm: hàm kích hoạt Sigmoid, độ sâu lớn, và cấu trúc không tối ưu để quan sát cách gradient giảm dần qua các lớp.

2. Thử nghiệm các giải pháp khác nhau để khắc phục Vanishing Gradient và đánh giá mức độ cải thiện. Các giải pháp bao gồm:
 - (a) **Weight Increasing:** Thay đổi cách khởi tạo trọng số ban đầu để tăng giá trị gradient, qua đó giảm thiểu khả năng gradient trở nên quá nhỏ.
 - (b) **Better Activation:** Thử nghiệm các hàm kích hoạt tiên tiến, không bị bão hòa dễ dàng như Sigmoid, nhằm giữ gradient ổn định hơn.
 - (c) **Better Optimizer:** Sử dụng các thuật toán tối ưu hiện đại hơn để tăng tốc độ hội tụ và giảm thiểu vấn đề Vanishing Gradient.
 - (d) **Normalize Inside Network:** Áp dụng kỹ thuật chuẩn hóa để giữ giá trị đầu ra trong khoảng ổn định, từ đó giúp gradient không giảm quá nhanh qua các lớp.
 - (e) **Skip Connection:** Tích hợp các skip connection trong kiến trúc mạng để tạo đường dẫn gradient trực tiếp từ các lớp trước đến các lớp sau, như kiến trúc ResNet đã chứng minh hiệu quả.
 - (f) **Train Layers Separately (Fine-Tuning):** Huấn luyện tuần tự các lớp để giảm thiểu tác động của các lớp sâu hơn đến gradient.
 - (g) **Gradient Normalization:** Chuẩn hóa gradient trong quá trình lan truyền ngược để duy trì giá trị gradient hợp lý.

Pipeline của project có thể được minh họa như sau:



Hình 32.3: Pipeline minh họa tổng quan nội dung project.

Thông qua phân tích và so sánh kết quả từ các giải pháp, chúng ta sẽ rút ra điểm mạnh, yếu và tính ứng dụng của từng phương pháp, từ đó đề xuất các chiến lược phù hợp để khắc phục vấn đề Vanishing Gradient trong mạng MLP.

32.2 Cài đặt chương trình

Dựa trên mô tả project ở phần trước, bước đầu tiên ta sẽ tiến hành xây dựng chương trình huấn luyện và đánh giá một mô hình MLP trên bộ dữ liệu Fashion MNIST. Mô hình này sẽ được thiết kế với chủ đích thể hiện được vấn đề Vanishing Gradient, và ta gọi mô hình này baseline model.

- Import các thư viện cần thiết:** Đầu tiên, chúng ta cần import các thư viện cần thiết cho việc xử lý dữ liệu, xây dựng mô hình, và các công cụ hỗ trợ. Các thư viện này bao gồm PyTorch cho việc cài đặt mô hình, cùng với torchvision để tải dữ liệu Fashion MNIST, và matplotlib để trực quan hóa.

```

1 import random
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import torch
5 import torch.optim as optim
6 import torchvision
7 import torchvision.transforms as transforms
8 from torch import nn
9 from torch.utils.data import Dataset, DataLoader,
    random_split
10 from torchvision.datasets import FashionMNIST

```

- Xác định phần cứng và cố định tham số ngẫu nhiên:** Để đảm bảo kết quả mô hình huấn luyện được là như nhau khi bắt đầu chạy chương trình ở bất kì Google Colab nào, chúng ta cần cố định tham số ngẫu nhiên (seed) cho mọi thư viện liên quan. Ngoài ra, chúng ta cũng cần xác định phần cứng (GPU nếu khả dụng hoặc CPU) để tối ưu hóa quá trình huấn luyện.

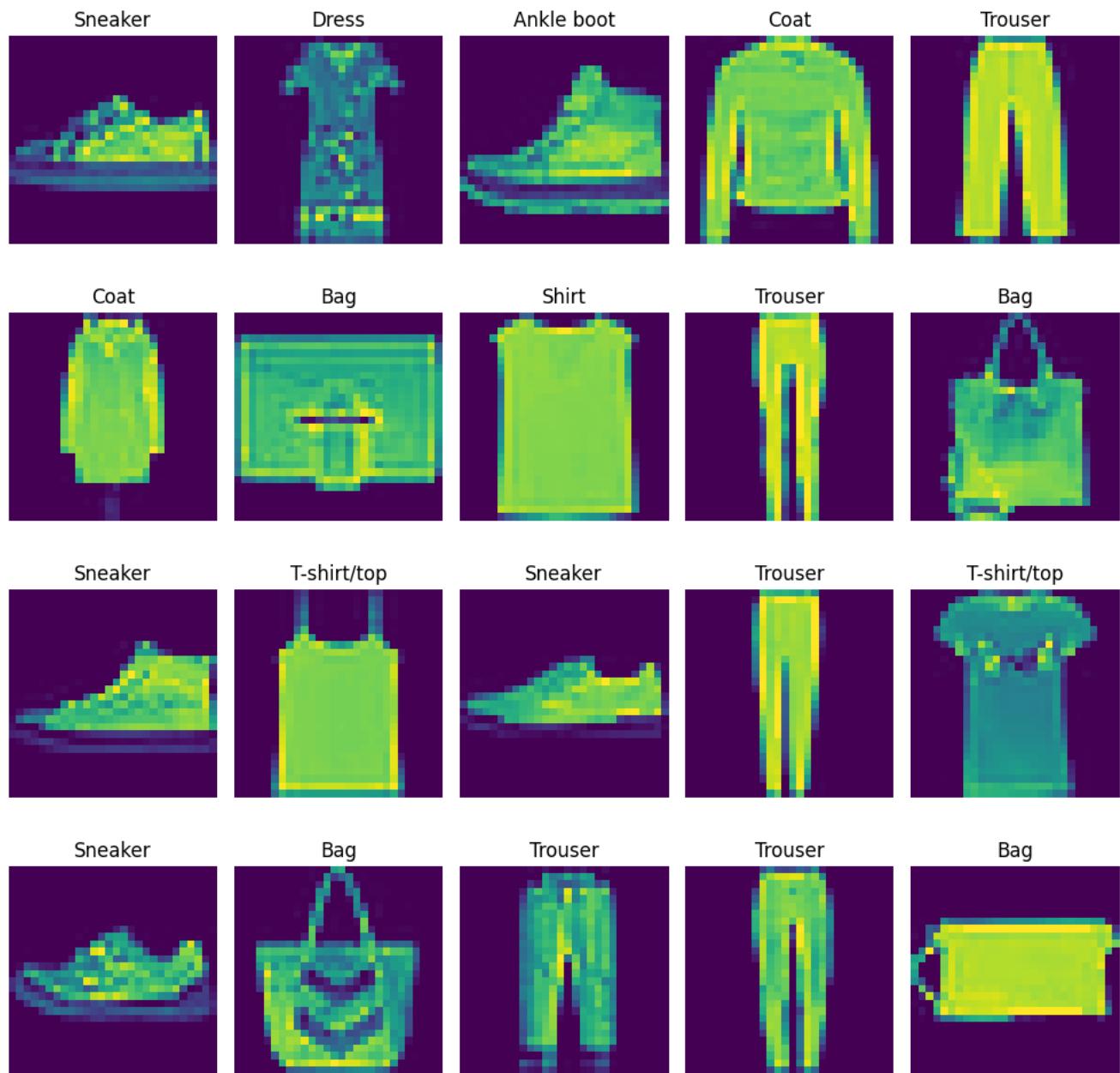
```

11 device = torch.device('cuda:0' if torch.cuda.is_available()
    () else 'cpu')
12
13 def set_seed(seed):
14     random.seed(seed)
15     np.random.seed(seed)
16     torch.manual_seed(seed)
17     torch.cuda.manual_seed(seed)
18     torch.cuda.manual_seed_all(seed)
19     torch.backends.cudnn.deterministic = True

```

```
20     torch.backends.cudnn.benchmark = False
21
22 SEED = 42
23 set_seed(SEED)
```

3. **Tải bộ dữ liệu:** Bộ dữ liệu **Fashion MNIST** là một tập dữ liệu phổ biến, bao gồm các hình ảnh grayscale 28x28 của các loại quần áo, được sử dụng để phân loại 10 nhãn khác nhau. PyTorch cung cấp hàm tải tự động cho bộ dữ liệu này. Theo đó, chúng ta tải cả tập train và test, đồng thời chuyển đổi dữ liệu sang tensor.



Hình 32.4: Minh họa một vài mẫu dữ liệu từ bộ dữ liệu Fashion MNIST.

```
24 train_dataset = FashionMNIST('./data',  
25         train=True,
```

```

26                                     download=True ,
27                                     transform=transforms .
28 test_dataset = FashionMNIST('./data' ,
29                             train=False ,
30                             download=True ,
31                             transform=transforms.ToTensor
32

```

4. **Chia bộ dữ liệu train/val/test:** Vì bộ dữ liệu gốc đã được chia sẵn tập train và tập test, chúng ta chỉ cần tạo thêm tập validation là đủ. Để tạo tập validation, ta chia bộ train ban đầu thành hai phần với tỉ lệ chia là 90% cho train và 10% cho validation. Các tập dữ liệu này sau đó được gói vào DataLoader để dễ dàng sử dụng trong quá trình huấn luyện.

```

32 train_ratio = 0.9
33 train_size = int(len(train_dataset) * train_ratio)
34 val_size = len(train_dataset) - train_size
35
36 train_subset, val_subset = random_split(train_dataset, [
37     train_size, val_size])
38
39 train_loader = DataLoader(train_subset, batch_size=
40     batch_size, shuffle=True)
41 val_loader = DataLoader(val_subset, batch_size=batch_size
42     , shuffle=False)
43 test_loader = DataLoader(test_dataset, batch_size=
44     batch_size, shuffle=False)
45
46 print(f"Train size: {len(train_subset)}")
47 print(f"Validation size: {len(val_subset)}")
48 print(f"Test size: {len(test_dataset)}")

```

5. **Xây dựng mô hình MLP:** Tiếp theo, chúng ta xây dựng mạng MLP với nhiều lớp ẩn. Mỗi lớp ẩn được nối tiếp bởi hàm kích hoạt Sigmoid. Sau khi định nghĩa kiến trúc mạng, chúng ta sẽ khởi tạo mô hình và khai báo hàm mất mát (CrossEntropyLoss) cùng thuật toán tối ưu (SGD).

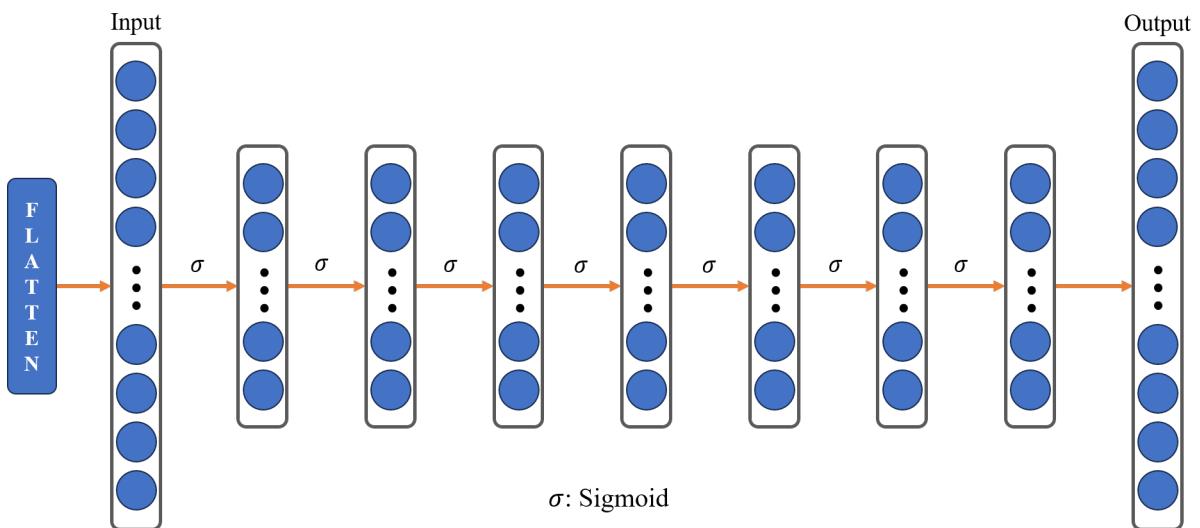
```

45 class MLP(nn.Module):
46     def __init__(self, input_dims, hidden_dims,
47                  output_dims):
48         super(MLP, self).__init__()

```

```
48         self.layer1 = nn.Linear(input_dims, hidden_dims)
49         self.layer2 = nn.Linear(hidden_dims, hidden_dims)
50         self.layer3 = nn.Linear(hidden_dims, hidden_dims)
51         self.layer4 = nn.Linear(hidden_dims, hidden_dims)
52         self.layer5 = nn.Linear(hidden_dims, hidden_dims)
53         self.layer6 = nn.Linear(hidden_dims, hidden_dims)
54         self.layer7 = nn.Linear(hidden_dims, hidden_dims)
55         self.output = nn.Linear(hidden_dims, output_dims)
56
57
58     def forward(self, x):
59         x = nn.Flatten()(x)
60         x = self.layer1(x)
61         x = nn.Sigmoid()(x)
62         x = self.layer2(x)
63         x = nn.Sigmoid()(x)
64         x = self.layer3(x)
65         x = nn.Sigmoid()(x)
66         x = self.layer4(x)
67         x = nn.Sigmoid()(x)
68         x = self.layer5(x)
69         x = nn.Sigmoid()(x)
70         x = self.layer6(x)
71         x = nn.Sigmoid()(x)
72         x = self.layer7(x)
73         x = nn.Sigmoid()(x)
74         out = self.output(x)
75
76     return out
77
78 input_dims = 784
79 hidden_dims = 128
80 output_dims = 10
81 lr = 1e-2
82
83 model = MLP(input_dims=input_dims,
84               hidden_dims=hidden_dims,
85               output_dims=output_dims).to(device)
86
87 criterion = nn.CrossEntropyLoss()
88 optimizer = optim.SGD(model.parameters(), lr=lr)
```

Theo đó, để mô hình thể hiện được vấn đề Vanishing Gradient, ta thiết kế mô hình với số lớp ẩn là 7, mỗi lớp ẩn sẽ có 128 node, và hàm kích hoạt được sử dụng là hàm Sigmoid.



Hình 32.5: Minh họa kiến trúc thành phần của mô hình baseline trong project.

6. **Huấn luyện mô hình:** Khi đã chuẩn bị đầy đủ các thành phần cần thiết bao gồm mô hình, DataLoader, hàm mất mát và thuật toán tối ưu, chúng ta có thể tiến hành huấn luyện mô hình. Quá trình huấn luyện bao gồm hai giai đoạn: train và validation. Trong mỗi epoch, mô hình được tối ưu trên tập train và đánh giá trên tập validation. Kết quả được lưu lại để phục vụ việc đánh giá và trực quan hóa sau này.

```
1 epochs = 100
2 train_loss_lst = []
3 train_acc_lst = []
4 val_loss_lst = []
5 val_acc_lst = []
6
7 for epoch in range(epochs):
8     train_loss = 0.0
9     train_acc = 0.0
10    count = 0
11
12    model.train()
13    for X_train, y_train in train_loader:
14        X_train, y_train = X_train.to(device), y_train.to(
15            device)
16        optimizer.zero_grad()
17        outputs = model(X_train)
18        loss = criterion(outputs, y_train)
```

```

18         loss.backward()
19         optimizer.step()
20         train_loss += loss.item()
21         train_acc += (torch.argmax(outputs, 1) == y_train
22             ).sum().item()
23         count += len(y_train)
24
25         train_loss /= len(train_loader)
26         train_loss_lst.append(train_loss)
27         train_acc /= count
28         train_acc_lst.append(train_acc)
29
30         val_loss = 0.0
31         val_acc = 0.0
32         count = 0
33         model.eval()
34         with torch.no_grad():
35             for X_val, y_val in val_loader:
36                 X_val, y_val = X_val.to(device), y_val.to(
37                     device)
38                 outputs = model(X_val)
39                 loss = criterion(outputs, y_val)
40                 val_loss += loss.item()
41                 val_acc += (torch.argmax(outputs, 1) == y_val
42                     ).sum().item()
43                 count += len(y_val)
44
45         val_loss /= len(val_loader)
46         val_loss_lst.append(val_loss)
47         val_acc /= count
48         val_acc_lst.append(val_acc)
49
50         print(f"EPOCH {epoch+1}/{epochs}, Train_Loss: {train_loss:.4f}, Train_Acc: {train_acc:.4f},
51             Validation Loss: {val_loss:.4f}, Val_Acc: {val_acc:.4f}")

```

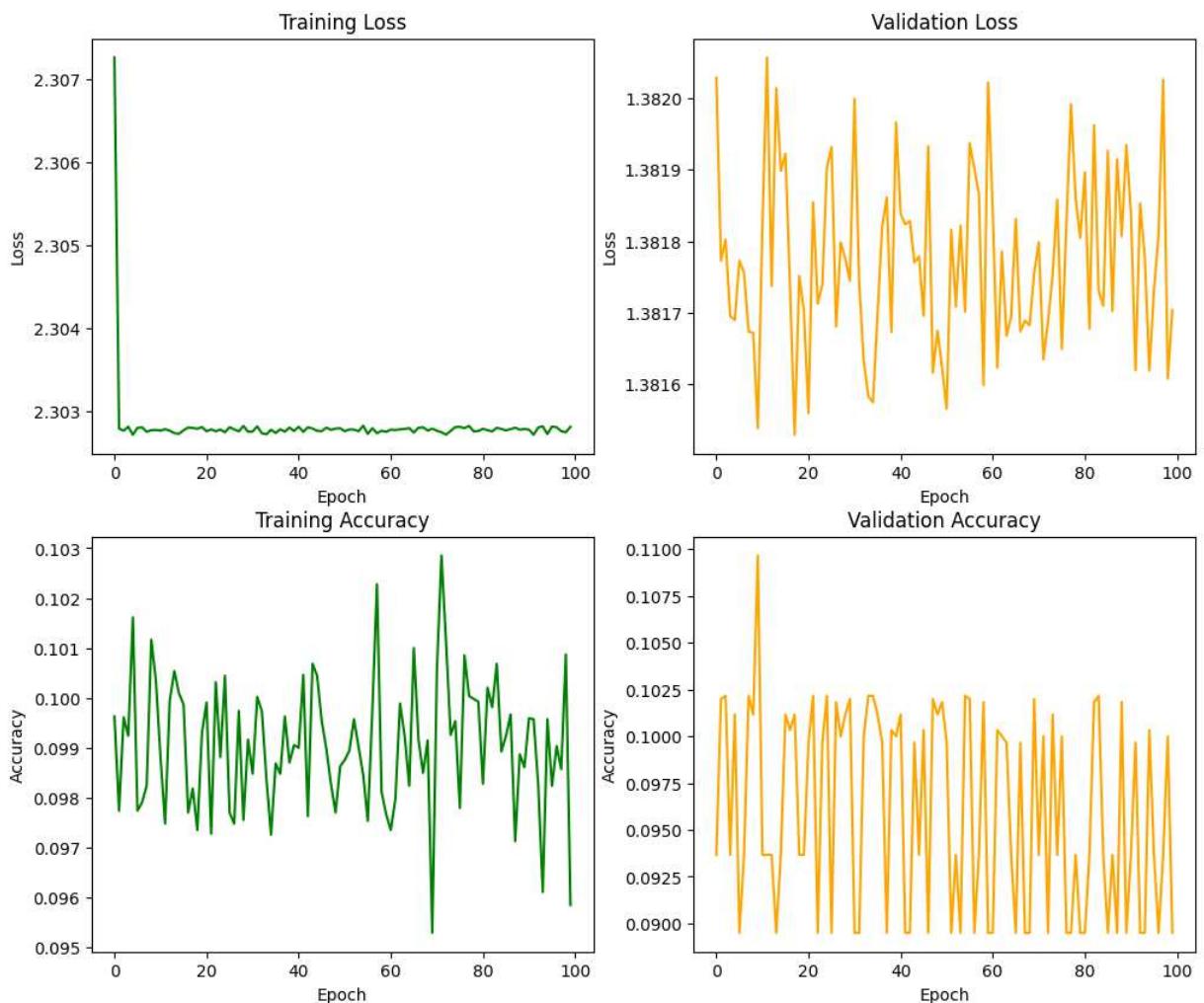
7. **Trực quan hóa kết quả huấn luyện:** Sau khi hoàn tất quá trình huấn luyện, việc trực quan hóa kết quả giúp chúng ta hiểu rõ hơn về cách mô hình học và hiệu suất của nó trên các tập dữ liệu. Dưới đây là cách vẽ biểu đồ Loss và Accuracy cho cả tập train và validation.

```

1 fig, ax = plt.subplots(2, 2, figsize=(12, 10))
2 ax[0, 0].plot(train_loss_lst, color='green')
3 ax[0, 0].set(xlabel='Epoch', ylabel='Loss')

```

```
4 ax[0, 0].set_title('Training Loss')
5
6 ax[0, 1].plot(val_loss_lst, color='orange')
7 ax[0, 1].set(xlabel='Epoch', ylabel='Loss')
8 ax[0, 1].set_title('Validation Loss')
9
10 ax[1, 0].plot(train_acc_lst, color='green')
11 ax[1, 0].set(xlabel='Epoch', ylabel='Accuracy')
12 ax[1, 0].set_title('Training Accuracy')
13
14 ax[1, 1].plot(val_acc_lst, color='orange')
15 ax[1, 1].set(xlabel='Epoch', ylabel='Accuracy')
16 ax[1, 1].set_title('Validation Accuracy')
17
18 plt.show()
```



Hình 32.6: Trực quan hóa hiệu suất mô hình baseline trong quá trình training. Có thể thấy mô hình không thể đạt sự hội tụ tốt do vướng phải hiện tượng Vanishing Gradient.

8. **Đánh giá mô hình:** Cuối cùng, sau khi đã huấn luyện mô hình và kiểm tra trên tập validation, chúng ta cần đánh giá hiệu suất của mô hình trên tập test. Quá trình đánh giá này sử dụng độ đo Accuracy để kiểm tra khả năng phân loại của mô hình trên dữ liệu chưa từng thấy.

```

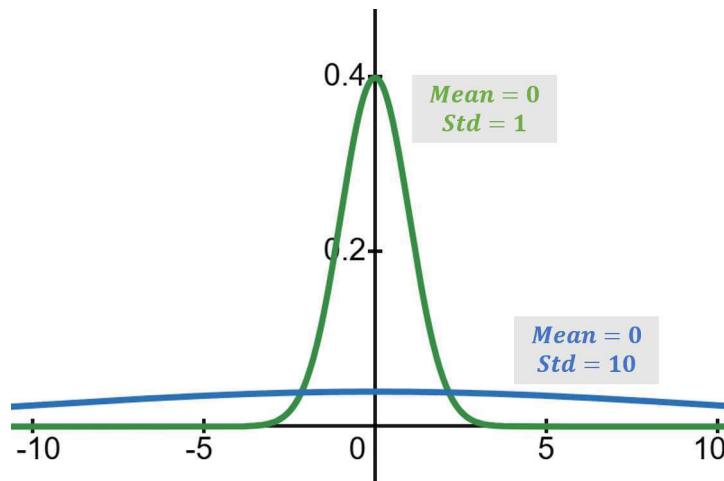
1 test_target = []
2 test_predict = []

```

```
3
4 model.eval()
5 with torch.no_grad():
6     for X_test, y_test in test_loader:
7         X_test = X_test.to(device)
8         y_test = y_test.to(device)
9         outputs = model(X_test)
10
11     test_predict.append(outputs.cpu())
12     test_target.append(y_test.cpu())
13
14     test_predict = torch.cat(test_predict)
15     test_target = torch.cat(test_target)
16     test_acc = (torch.argmax(test_predict, 1) ==
17                 test_target).sum().item() / len(test_target)
18
19     print('Evaluation on test set:')
20     print(f'Accuracy: {test_acc}')
```

Dựa trên baseline model đã xây dựng, chúng ta sẽ triển khai code cài đặt các phương pháp cải tiến nhằm giảm thiểu vấn đề Vanishing Gradient trong baseline model. Mỗi phương pháp các bạn cần phải chạy lại toàn bộ chương trình huấn luyện sau khi áp dụng cải tiến để thấy được kết quả. Song vì hầu hết các phần code sẽ tương đồng với code baseline, code cho các giải pháp sẽ chỉ đề cập đến đoạn cần cập nhật. Các bạn cần đọc kĩ code baseline và chỉnh sửa sao cho phù hợp nhất.

1. **Weight Increasing:** Kỹ thuật này tập trung vào việc khởi tạo trọng số ban đầu với các giá trị lớn hơn nhằm tăng giá trị gradient trong các bước lan truyền ngược. Điều này giúp hạn chế việc gradient bị triệt tiêu qua các lớp, đặc biệt khi sử dụng các hàm kích hoạt dễ bão hòa như Sigmoid. Tại đây, chúng ta triển khai ý tưởng này bằng cách khởi tạo trọng số với phân phối chuẩn (normal distribution) và sử dụng hai thiết lập khác nhau để đánh giá tác động: một với độ lệch chuẩn (std) là 1.0 và một với std là 10.0.



Hình 32.7: Normal Distribution với mean bằng 0 và standard deviation bằng 1 và 10.

(a) $std = 1.0$:

```
1 class MLP(nn.Module):
2     def __init__(self, input_dims, hidden_dims,
3                  output_dims):
4         super(MLP, self).__init__()
5         self.layer1 = nn.Linear(input_dims,
6                                hidden_dims)
7         self.layer2 = nn.Linear(hidden_dims,
8                                hidden_dims)
9         self.layer3 = nn.Linear(hidden_dims,
10                                hidden_dims)
11        self.layer4 = nn.Linear(hidden_dims,
12                                hidden_dims)
13        self.layer5 = nn.Linear(hidden_dims,
14                                hidden_dims)
15        self.layer6 = nn.Linear(hidden_dims,
16                                hidden_dims)
17        self.layer7 = nn.Linear(hidden_dims,
18                                hidden_dims)
19        self.output = nn.Linear(hidden_dims,
20                               output_dims)
21
22        for module in self.modules():
23            if isinstance(module, nn.Linear):
24                nn.init.normal_(module.weight, mean
25=0.0, std=1.0)
```

```

16             nn.init.constant_(module.bias, 0.0)
17
18     def forward(self, x):
19         x = nn.Flatten()(x)
20         x = self.layer1(x)
21         x = nn.Sigmoid()(x)
22         x = self.layer2(x)
23         x = nn.Sigmoid()(x)
24         x = self.layer3(x)
25         x = nn.Sigmoid()(x)
26         x = self.layer4(x)
27         x = nn.Sigmoid()(x)
28         x = self.layer5(x)
29         x = nn.Sigmoid()(x)
30         x = self.layer6(x)
31         x = nn.Sigmoid()(x)
32         x = self.layer7(x)
33         x = nn.Sigmoid()(x)
34         out = self.output(x)
35
36     return out

```

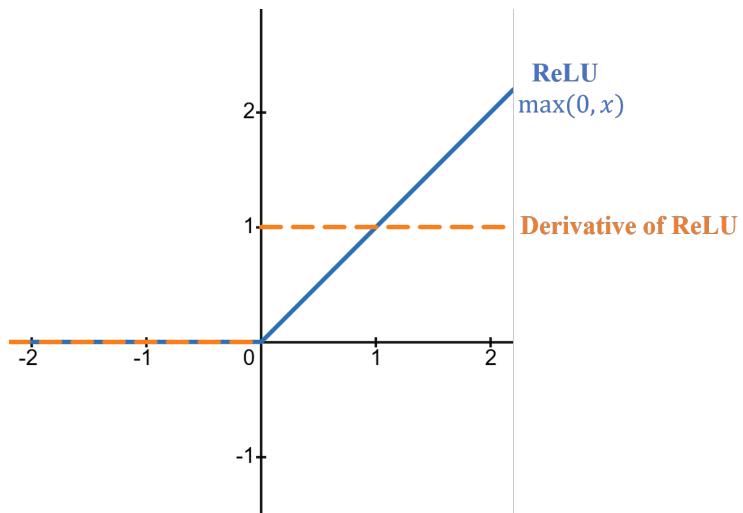
- (b) $std = 10.0$: Các bạn thay đổi vào vị trí khởi tạo trọng số của đoạn code trên như sau:

```

1 for module in self.modules():
2     if isinstance(module, nn.Linear):
3         nn.init.normal_(module.weight, mean=0.0, std
4                         =10.0)
5         nn.init.constant_(module.bias, 0.0)

```

2. **Better Activation:** Kỹ thuật này liên quan đến việc thay đổi hàm kích hoạt trong mạng thành các hàm tiên tiến hơn. Những hàm này có khả năng hạn chế hiện tượng bão hòa, giúp gradient duy trì giá trị hợp lý qua các lớp mạng. Tại đây, chúng ta triển khai ý tưởng này bằng cách thay thế tất cả các hàm Sigmoid trong mạng baseline bằng ReLU. Ngoài ra, trọng số của các lớp Linear cũng được khởi tạo với độ lệch chuẩn nhỏ hơn ($std = 0.05$) để phù hợp với tính chất của ReLU.



Hình 32.8: Đồ thị hàm ReLU.

```

1  class MLP(nn.Module):
2      def __init__(self, input_dims, hidden_dims,
3          output_dims):
4          super(MLP, self).__init__()
5          self.layer1 = nn.Linear(input_dims, hidden_dims)
6          self.layer2 = nn.Linear(hidden_dims, hidden_dims)
7          self.layer3 = nn.Linear(hidden_dims, hidden_dims)
8          self.layer4 = nn.Linear(hidden_dims, hidden_dims)
9          self.layer5 = nn.Linear(hidden_dims, hidden_dims)
10         self.layer6 = nn.Linear(hidden_dims, hidden_dims)
11         self.layer7 = nn.Linear(hidden_dims, hidden_dims)
12         self.output = nn.Linear(hidden_dims, output_dims)
13
14         for m in self.modules():
15             if isinstance(m, nn.Linear):
16                 nn.init.normal_(m.weight, mean=0.0, std
17 = 0.05)
18                 nn.init.constant_(m.bias, 0.0)
19
20     def forward(self, x):
21         x = nn.Flatten()(x)
22         x = self.layer1(x)
23         x = nn.ReLU()(x)
24         x = self.layer2(x)
25         x = nn.ReLU()(x)
26         x = self.layer3(x)

```

```

25         x = nn.ReLU()(x)
26         x = self.layer4(x)
27         x = nn.ReLU()(x)
28         x = self.layer5(x)
29         x = nn.ReLU()(x)
30         x = self.layer6(x)
31         x = nn.ReLU()(x)
32         x = self.layer7(x)
33         x = nn.ReLU()(x)
34         out = self.output(x)
35
36     return out

```

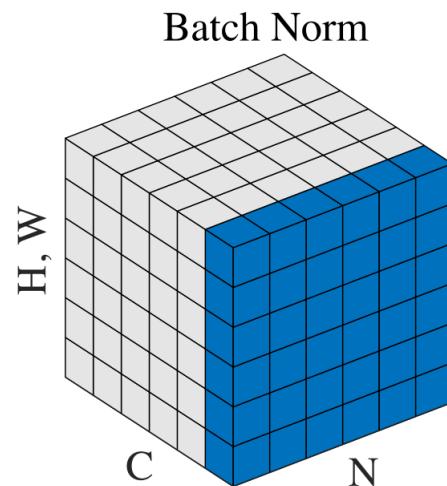
3. **Better Optimizer:** Lựa chọn thuật toán tối ưu phù hợp có thể giúp cải thiện vấn đề Vanishing Gradient bằng cách điều chỉnh gradient linh hoạt hơn trong quá trình lan truyền ngược. Tại đây, chúng ta thay SGD bằng Adam, một thuật toán hiện đại với khả năng tự động điều chỉnh tốc độ học, để kiểm tra hiệu quả cải thiện của nó trong việc giảm thiểu Vanishing Gradient.

```

1 input_dims = 784
2 hidden_dims = 128
3 output_dims = 10
4
5 # Baseline network
6 model = MLP(input_dims=input_dims,
7               hidden_dims=hidden_dims,
8               output_dims=output_dims).to(device)
9
10 criterion = nn.CrossEntropyLoss()
11
12 lr = 1e-3
13 optimizer = optim.Adam(model.parameters(), lr=lr)

```

4. **Normalize Inside Network:** Batch Normalization là một kỹ thuật phổ biến để giữ các đầu vào của từng lớp mạng trong khoảng ổn định, giúp gradient không bị triệt tiêu hoặc phóng đại. Kỹ thuật này cũng góp phần tăng tốc độ hội tụ và cải thiện khả năng tổng quát hóa của mô hình. Theo đó, ta sẽ thử nghiệm hai kỹ thuật normalization:



Hình 32.9: Minh họa Batch Normalization. Nguồn ảnh: [Link](#).

- (a) **Normalize Inside Network:** Batch Normalization là một kỹ thuật phổ biến giúp chuẩn hóa đầu ra của từng lớp, giữ giá trị trong khoảng ổn định và cải thiện tốc độ hội tụ. Trong phần này, chúng ta sẽ khai báo thêm `nn.BatchNorm1d` và đặt sau mỗi lớp Linear trong mạng baseline. Batch Normalization giúp giảm thiểu nguy cơ bão hòa và duy trì gradient ổn định qua các lớp.

```
1  class MLP(nn.Module):
2      def __init__(self, input_dims, hidden_dims,
3          output_dims):
4          super(MLP, self).__init__()
5          self.hidden_dims = hidden_dims
6          self.layer1 = nn.Linear(input_dims,
7              hidden_dims)
8          self.bn1 = nn.BatchNorm1d(hidden_dims)
9          self.layer2 = nn.Linear(hidden_dims,
10             hidden_dims)
11         self.bn2 = nn.BatchNorm1d(hidden_dims)
12         self.layer3 = nn.Linear(hidden_dims,
13             hidden_dims)
14         self.bn3 = nn.BatchNorm1d(hidden_dims)
15         self.layer4 = nn.Linear(hidden_dims,
16             hidden_dims)
17         self.bn4 = nn.BatchNorm1d(hidden_dims)
18         self.layer5 = nn.Linear(hidden_dims,
19             hidden_dims)
20         self.bn5 = nn.BatchNorm1d(hidden_dims)
21         self.layer6 = nn.Linear(hidden_dims,
22             hidden_dims)
23         self.bn6 = nn.BatchNorm1d(hidden_dims)
24         self.layer7 = nn.Linear(hidden_dims,
25             hidden_dims)
26         self.bn7 = nn.BatchNorm1d(hidden_dims)
27         self.output = nn.Linear(hidden_dims,
28             output_dims)
29
30         for module in self.modules():
31             if isinstance(module, nn.Linear):
32                 nn.init.normal_(module.weight, mean
33 = 0.0, std=0.05)
34                 nn.init.constant_(module.bias, 0.0)
35
36     def forward(self, x):
37         x = nn.Flatten()(x)
38
39         x = self.layer1(x)
40         x = self.bn1(x)
41         x = nn.Sigmoid()(x)
42         x = self.layer2(x)
43         x = self.bn2(x)
44         x = nn.Sigmoid()(x)
45         x = self.layer3(x)
```

```
36         x = self.bn3(x)
37         x = nn.Sigmoid()(x)
38         x = self.layer4(x)
39         x = self.bn4(x)
40         x = nn.Sigmoid()(x)
41         x = self.layer5(x)
42         x = self.bn5(x)
43         x = nn.Sigmoid()(x)
44         x = self.layer6(x)
45         x = self.bn6(x)
46         x = nn.Sigmoid()(x)
47         x = self.layer7(x)
48         x = self.bn7(x)
49         x = nn.Sigmoid()(x)
50
51     out = self.output(x)
52
53     return out
```

- (b) **Customized Normalization Layer:** Chúng ta xây dựng một class layer normalization mới như sau:

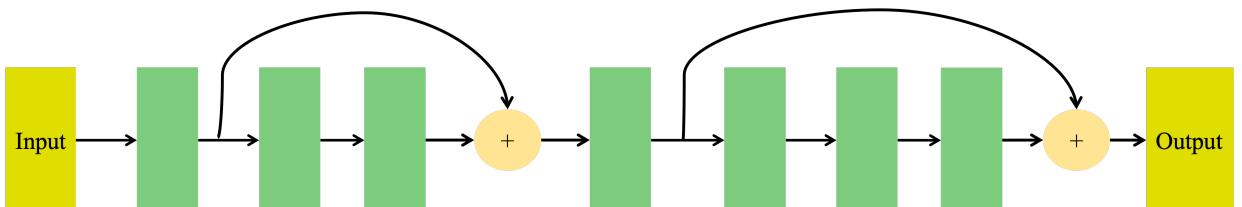
```

1 class MyNormalization(nn.Module):
2     def __init__(self):
3         super().__init__()
4
5     def forward(self, x):
6         mean = torch.mean(x)
7         std = torch.std(x)
8         return (x - mean) / std

```

Sau đó, các bạn hãy thay thế các vị trí được áp dụng `nn.BatchNorm1d()` bằng `MyNormalization()`.

5. **Skip Connection:** Được giới thiệu trong kiến trúc của mạng ResNet, giúp gradient có thể lan truyền trực tiếp qua các lớp mà không bị triệt tiêu. Điều này không chỉ cải thiện khả năng hội tụ mà còn giữ được thông tin từ các tầng trước đó. Tại đây, chúng ta triển khai skip connections bằng cách cộng đầu ra của các lớp trước (skip connections) với các lớp tiếp theo tại một số điểm trong mạng.



Hình 32.10: Minh họa về kỹ thuật skip connection được ứng dụng trong baseline network.

```

1 class MLP(nn.Module):
2     def __init__(self, input_dims, hidden_dims,
3                  output_dims):
4         super(MLP, self).__init__()
5         self.layer1 = nn.Linear(input_dims, hidden_dims)
6         self.layer2 = nn.Linear(hidden_dims, hidden_dims)
7         self.layer3 = nn.Linear(hidden_dims, hidden_dims)
8         self.layer4 = nn.Linear(hidden_dims, hidden_dims)
9         self.layer5 = nn.Linear(hidden_dims, hidden_dims)
10        self.layer6 = nn.Linear(hidden_dims, hidden_dims)
11        self.layer7 = nn.Linear(hidden_dims, hidden_dims)

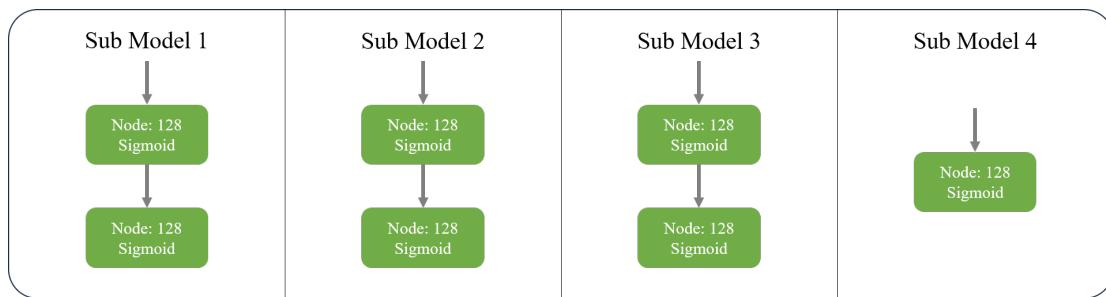
```

```

11         self.output = nn.Linear(hidden_dims, output_dims)
12
13     for module in self.modules():
14         if isinstance(module, nn.Linear):
15             nn.init.normal_(module.weight, mean=0.0,
16                             std=0.05)
17             nn.init.constant_(module.bias, 0.0)
18
19     def forward(self, x):
20         x = nn.Flatten()(x)
21         x = self.layer1(x)
22         x = nn.Sigmoid()(x)
23         skip = x
24
25         x = self.layer2(x)
26         x = nn.Sigmoid()(x)
27         x = self.layer3(x)
28         x = nn.Sigmoid()(x)
29         x = skip + x
30
31         x = self.layer4(x)
32         x = nn.Sigmoid()(x)
33         skip = x
34
35         x = self.layer5(x)
36         x = nn.Sigmoid()(x)
37         x = self.layer6(x)
38         x = nn.Sigmoid()(x)
39         x = self.layer7(x)
40         x = nn.Sigmoid()(x)
41         x = skip + x
42
43         out = self.output(x)
44
45     return out

```

6. **Train layers separately (fine-tuning):** Trong các mạng rất sâu, việc huấn luyện toàn bộ các lớp có thể dẫn đến hiệu suất thấp do vấn đề Vanishing Gradient. Bằng cách chỉ huấn luyện một số lớp cụ thể, mô hình có thể tập trung vào việc học các đặc trưng quan trọng hơn mà không bị ảnh hưởng bởi các lớp sâu hơn. Kỹ thuật này được triển khai bằng cách xây dựng các mô hình nhỏ hơn tương ứng với từng số lượng lớp cần huấn luyện, sau đó tăng dần các lớp được tham gia huấn luyện. Dưới đây là các giai đoạn thực hiện:



Hình 32.11: Mô hình ban đầu gồm 7 layer sẽ được chia thành 4 mô hình con với số lượng layer 2:2:2:1.

- (a) **Xây dựng các mô hình thành phần:** Các thành phần mạng (module) nhỏ hơn được xây dựng tương ứng với số lớp mong muốn, mỗi thành phần được cài đặt với 1 hoặc 2 lớp ẩn. Chúng ta bắt đầu với **MLP_1layer** (1 lớp) và **MLP_2layers** (2 lớp). Trọng số ban đầu của mỗi lớp được khởi tạo bằng phân phối chuẩn với độ lệch chuẩn *std* là 0.05 để hạn chế vấn đề gradient bị triệt tiêu.

```

1  class MLP_1layer(nn.Module):
2      def __init__(self, input_dims, output_dims):
3          super(MLP_1layer, self).__init__()
4          self.layer1 = nn.Linear(input_dims,
5                                 output_dims)
6
7          for module in self.modules():
8              if isinstance(module, nn.Linear):
9                  nn.init.normal_(module.weight, mean
10                     =0.0, std=0.05)
11                 nn.init.constant_(module.bias, 0.0)
12
13
14
15
16
17
18 class MLP_2layers(nn.Module):
19     def __init__(self, input_dims, output_dims):
20         super(MLP_2layers, self).__init__()
21         self.layer1 = nn.Linear(input_dims,
22                               output_dims)
23         self.layer2 = nn.Linear(output_dims,
24                               output_dims)
25
26         for module in self.modules():
27             if isinstance(module, nn.Linear):
28                 nn.init.normal_(module.weight, mean
29                     =0.0, std=0.05)
30                 nn.init.constant_(module.bias, 0.0)
31
32
33
34
35

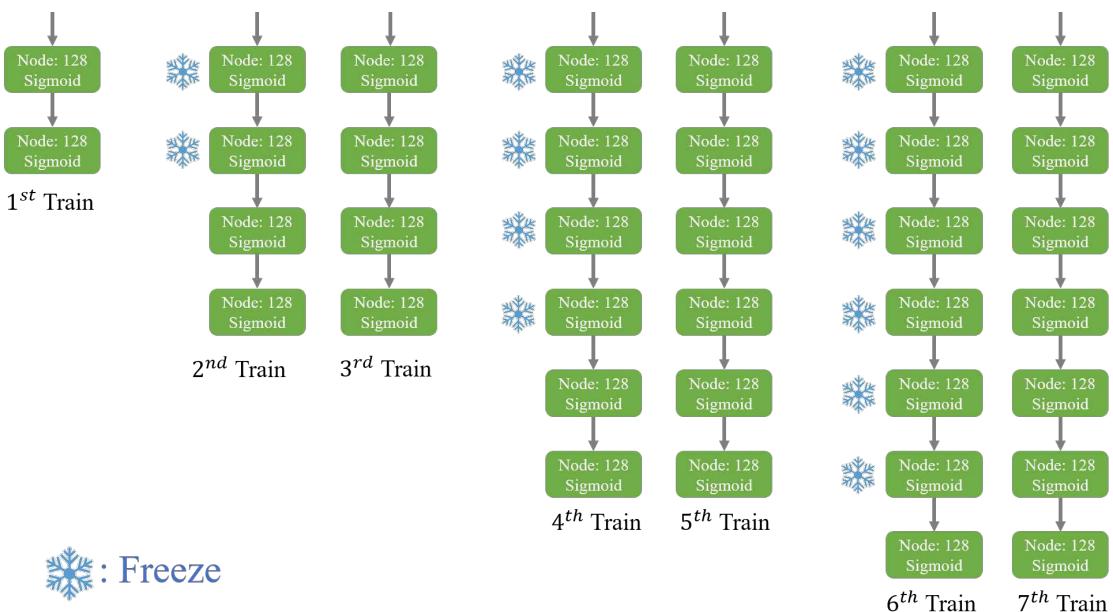
```

- (b) **Khởi tạo các module thành phần:** Các thành phần mạng first, second, third, và fourth được khởi tạo. Đây là các mô hình MLP_2layers hoặc MLP_1layer với các đặc trưng đầu vào và đầu ra cụ thể.

```

1 first = MLP_2layers(input_dims=784, output_dims=128)
2 second = MLP_2layers(input_dims=128, output_dims=128)
3 third = MLP_2layers(input_dims=128, output_dims=128)
4 fourth = MLP_1layer(input_dims=128, output_dims=128)
5
6 lr = 1e-2
7 criterion = nn.CrossEntropyLoss()

```



Hình 32.12: Lần lượt huấn luyện từng mô hình con. Sau khi một mô hình con đã huấn luyện xong, ta đóng băng trọng số đã train của mô hình, kết nối mô hình con tiếp theo và tiếp tục quá trình training.

- (c) **Giai đoạn 1 - Huấn luyện chỉ với thành phần đầu tiên:** Ở bước đầu tiên, chỉ thành phần `first` tham gia vào quá trình huấn luyện. Thành phần này được nối với một lớp đầu ra để thực hiện dự đoán, trong khi các thành phần khác chưa được thêm vào.

```

1 model = nn.Sequential(
2     first,
3     nn.Linear(128, 10)
4 ).to(device)
5
6 optimizer = optim.SGD(model.parameters(), lr=lr)

```

```

7
8 # Training code ...

```

- (d) **Giai đoạn 2 - Thêm thành phần thứ hai:** Sau khi huấn luyện xong `first`, chúng ta thêm thành phần `second` vào mạng. Thành phần `first` được giữ cố định (không cập nhật trọng số), và chỉ `second` được tham gia vào quá trình huấn luyện.

```

1 for param in first.parameters():
2     param.requires_grad = False
3
4 model = nn.Sequential(
5     first,
6     second,
7     nn.Linear(128, 10)
8 ).to(device)
9
10 optimizer = optim.SGD(model.parameters(), lr=lr)
11
12 # Training code ...

```

- (e) **Giai đoạn 3 - Cập nhật toàn bộ thành phần hiện có:** Khi đã huấn luyện riêng cho thành phần `second`, ta huấn luyện lại toàn bộ mạng hiện có mà không cố định thành phần nào.

```

1 for param in first.parameters():
2     param.requires_grad = True
3
4 model = nn.Sequential(
5     first,
6     second,
7     nn.Linear(128, 10)
8 ).to(device)
9
10 optimizer = optim.SGD(model.parameters(), lr=lr)
11
12 # Training code ...

```

- (f) **Giai đoạn 4 - Thêm thành phần thứ ba:** Tiếp theo, thành phần `third` được thêm vào mạng. Tương tự, `first` và `second` vẫn giữ cố định, chỉ `third` tham gia huấn luyện.

```

1 for param in first.parameters():
2     param.requires_grad = False
3 for param in second.parameters():

```

```

4     param.requires_grad = False
5
6 model = nn.Sequential(
7     first,
8     second,
9     third,
10    nn.Linear(128, 10)
11 ).to(device)
12
13 optimizer = optim.SGD(model.parameters(), lr=lr)
14
15 # Training code ...

```

- (g) **Giai đoạn 5 - Cập nhật toàn bộ thành phần hiện có:** Tương tự giai đoạn 3, sau khi thành phần mới (*third*) được huấn luyện riêng, ta huấn luyện lại toàn bộ mạng hiện có mà không cố định bất kỳ thành phần nào.

```

1 for param in first.parameters():
2     param.requires_grad = True
3 for param in second.parameters():
4     param.requires_grad = True
5
6 model = nn.Sequential(
7     first,
8     second,
9     third,
10    nn.Linear(128, 10)
11 ).to(device)
12
13 optimizer = optim.SGD(model.parameters(), lr=lr)
14
15 # Training code ...

```

- (h) **Giai đoạn 6 - Thêm thành phần thứ tư:** Lần thêm thành phần mới (*fourth*) lần cuối cùng này vẫn tương tự như các giai đoạn trước. Ta sẽ huấn luyện riêng cho thành phần mới và cố định các thành phần cũ.

```

1 for param in first.parameters():
2     param.requires_grad = False
3 for param in second.parameters():
4     param.requires_grad = False
5 for param in third.parameters():
6     param.requires_grad = False

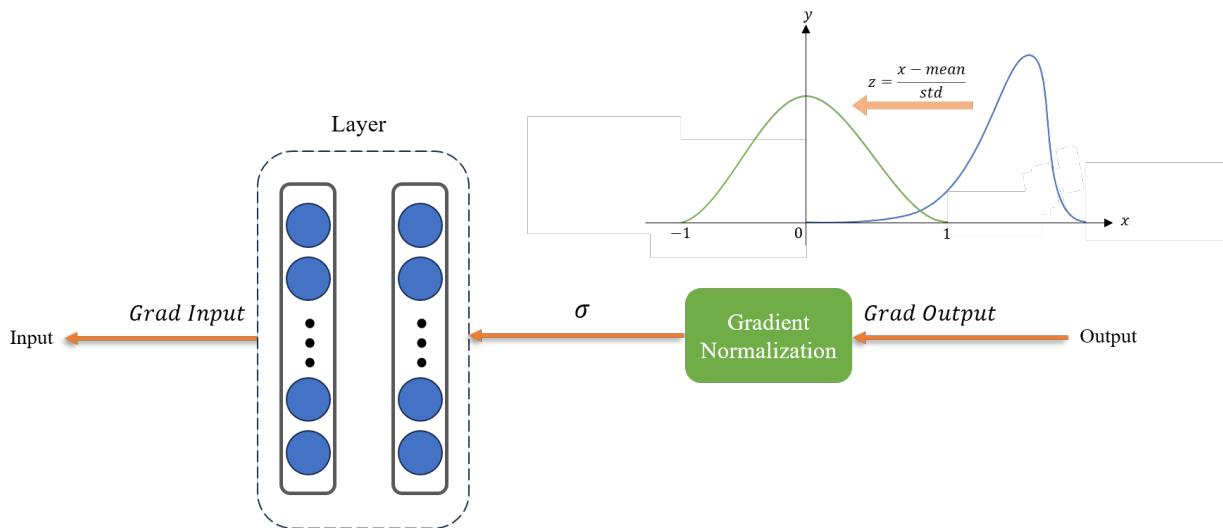
```

```
7
8 model = nn.Sequential(
9     first,
10    second,
11    third,
12    fourth,
13    nn.Linear(128, 10)
14 ).to(device)
15
16 optimizer = optim.SGD(model.parameters(), lr=lr)
17
18 # Training code ...
```

- (i) **Giai đoạn 7 - Mở khóa toàn bộ thành phần:** Cuối cùng, sau khi thêm đủ các thành phần, chúng ta mở khóa tất cả các lớp trong mạng và thực hiện huấn luyện toàn bộ mô hình. Điều này đảm bảo rằng tất cả các lớp đều được tối ưu hóa chung để đạt hiệu suất tốt nhất.

```
1 for param in first.parameters():
2     param.requires_grad = True
3 for param in second.parameters():
4     param.requires_grad = True
5 for param in third.parameters():
6     param.requires_grad = True
7
8 model = nn.Sequential(
9     first,
10    second,
11    third,
12    fourth,
13    nn.Linear(128, 10)
14 ).to(device)
15
16 optimizer = optim.SGD(model.parameters(), lr=lr)
17
18 # Training code ...
```

7. Gradient Normalization: Là một kỹ thuật với ý tưởng chuẩn hóa gradient trong quá trình lan truyền ngược. Kỹ thuật này đảm bảo gradient được duy trì trong một phạm vi hợp lý, tránh việc chúng trở nên quá nhỏ hoặc quá lớn, từ đó giúp cải thiện quá trình học của các lớp sâu hơn trong mạng. Tại đây, chúng ta cài đặt một lớp `GradientNormalizationLayer`, sử dụng cơ chế `autograd` của PyTorch để chuẩn hóa gradient trong giai đoạn lan truyền ngược. Cụ thể, gradient được điều chỉnh bằng cách chuẩn hóa theo trung bình và độ lệch chuẩn của chúng, đảm bảo các giá trị gradient không bị triệt tiêu hoặc phóng đại.



Hình 32.13: Minh họa về kỹ thuật Gradient Normalization. Gradient sẽ được chuẩn hóa khi thực hiện lan truyền ngược.

```

1 # Custom Gradient Normalization Layer
2 class GradientNormalization(torch.autograd.Function):
3     @staticmethod
4     def forward(ctx, input):
5         # Forward pass: pass input unchanged
6         ctx.save_for_backward(input)
7         return input
8
9     @staticmethod
10    def backward(ctx, grad_output):
11        # Normalize the gradient

```

```
12         mean = torch.mean(grad_output)
13         std = torch.std(grad_output)
14         grad_input = (grad_output - mean) / (std + 1e-6)
15         # Avoid division by zero
16         return grad_input
17
18 # Wrapper Module for GradientNormalization
19 class GradientNormalizationLayer(nn.Module):
20     def __init__(self):
21         super(GradientNormalizationLayer, self).__init__()
22
23     def forward(self, x):
24         return GradientNormalization.apply(x)
```

Sau định nghĩa class trên với triển khai gradient normalization, chúng ta có thể dễ dàng tích hợp lớp này vào các mạng MLP hiện có. Các bạn sẽ sử dụng lớp `GradientNormalizationLayer` sau các lớp Linear, đảm bảo rằng gradient được chuẩn hóa trước khi lan truyền ngược qua từng lớp.

Như vậy, chúng ta đã tìm hiểu và áp dụng một số kỹ thuật nhằm khắc phục vấn đề Vanishing Gradient trong mạng MLP, giúp cải thiện hiệu quả học của mô hình. Mặc dù vẫn còn nhiều kỹ thuật khác có thể được áp dụng, mục tiêu chính của project này là làm rõ sự tồn tại và tác động của vấn đề Vanishing Gradient trong Deep Learning, đồng thời minh họa các giải pháp cơ bản để xử lý vấn đề này.

32.3 Câu hỏi trắc nghiệm

1. Vấn đề Vanishing Gradient trong Deep Learning là gì?
 - (a) Gradients trở nên quá lớn.
 - (b) Gradients tiệm cận 0.
 - (c) Gradient không xác định.
 - (d) Gradient là số âm.
2. Hàm activation nào có nguy cơ cao nhất gặp phải vấn đề Vanishing Gradient?
 - (a) ReLU.
 - (b) Sigmoid.
 - (c) LeakyReLU.
 - (d) Tất cả các hàm trên.
3. Hậu quả tiềm ẩn của vấn đề Vanishing Gradient có thể là gì?
 - (a) Quá trình huấn luyện nhanh hơn.
 - (b) Trọng số ngừng cập nhật.
 - (c) Overfitting.
 - (d) Underfitting.
4. Dấu hiệu Vanishing Gradient xảy ra trong quá trình training một mô hình classification là gì?
 - (a) Giá trị loss giảm nhanh chóng.
 - (b) Accuracy tăng rất nhanh.
 - (c) Sự đình trệ về loss và accuracy.
 - (d) Tất cả đều đúng.
5. Vanishing Gradient phụ thuộc vào yếu tố nào?
 - (a) Độ sâu của network.

- (b) Hàm kích hoạt.
(c) Khởi tạo trọng số.
(d) Tất cả các yếu tố trên.
6. Xem xét một mạng Deep Neural Network sử dụng hàm Sigmoid. Nếu đầu ra của một neuron là 0.01, đạo hàm tại điểm này là bao nhiêu?
(a) 0.0099.
(b) 0.0098.
(c) 0.01.
(d) 0.1.
7. Nếu hàm activation là ReLU, đạo hàm sẽ là bao nhiêu khi đầu vào là số lớn hơn 0?
(a) 1.
(b) 0.
(c) Không xác định.
(d) Bằng giá trị đầu vào.
8. Giả sử một đạo hàm nhỏ dương g được lan truyền ngược qua một Neural Network sâu với n lớp sử dụng hàm kích hoạt Sigmoid, đạo hàm tại lớp đầu tiên sẽ như thế nào?
(a) $\approx g$.
(b) $\approx g^n$.
(c) $\approx \frac{1}{g}$.
(d) $\approx \frac{g}{2}$.
9. Giả sử bạn có một Neural Network đơn giản với chỉ một hidden layer sử dụng hàm Sigmoid. Đầu vào của layer này là 10, và trọng số là 0.1. Hãy tính đạo hàm của hàm Sigmoid tại layer này?
(a) ≈ 0.0249 .
(b) ≈ 0.1 .

- (c) ≈ 0.0099 .
 - (d) ≈ 0.1966 .
10. Layer nào bị ảnh hưởng nhiều nhất bởi vấn đề Vanishing Gradient?

- (a) Input layer.
- (b) Output layer.
- (c) Các hidden layer gần input.
- (d) Các hidden layer gần output.

32.4 Phụ lục

1. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

Mục	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Khái niệm Vanishing Gradient. - Dấu hiệu Vanishing Gradient. - Nguyên nhân Vanishing Gradient. - Các giải pháp khắc phục Vanishing Gradient. 	<ul style="list-style-type: none"> - Hiểu về khái niệm Vanishing Gradient. - Có thể nhận biết được Vanishing Gradient trong lúc train, và dựa vào các nguyên nhân để khắc phục và cải thiện model. - Có thể áp dụng các biện pháp giảm thiểu Vanishing Gradient như: khởi tạo weight, chọn activation function, optimizer, normalize layer, skip connection và chiến thuật train layer separately.
2	<ul style="list-style-type: none"> - Khởi tạo weight để giảm thiểu vấn đề vanishing cụ thể trong project này. - Khởi tạo weights với PyTorch. 	<ul style="list-style-type: none"> - Hiểu được tầm quan trọng của việc khởi tạo weights. - Biết khởi tạo weight với PyTorch.
3	<ul style="list-style-type: none"> - Thay đổi activation function để giảm thiểu vấn đề vanishing cụ thể trong project này. - Thay đổi activation function với PyTorch. 	<ul style="list-style-type: none"> - Hiểu được tầm quan trọng của việc sử dụng đúng activation function. - Biết thay đổi activation function trong PyTorch.

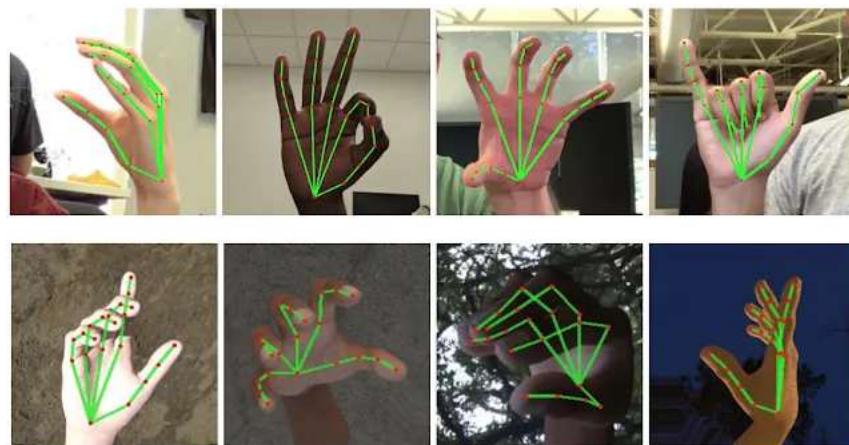
4	<ul style="list-style-type: none"> - Thay đổi các optimizer để giảm thiểu vấn đề vanishing cụ thể trong project này. - Thay đổi optimizer khi compile model trong PyTorch. 	<ul style="list-style-type: none"> - Hiểu được tầm quan trọng của các optimizer. - Biết thay đổi optimizer khi compile model trong PyTorch.
5	<ul style="list-style-type: none"> - Normalize bên trong network bằng BatchNormalization layer để giảm thiểu vấn đề vanishing cụ thể trong project này. - Normalize bên trong network bằng custom layer của riêng mình để giảm thiểu vấn đề vanishing cụ thể trong project này. - Sử dụng BatchNormalization với PyTorch. - Cách tạo ra custom layer với PyTorch. 	<ul style="list-style-type: none"> - Hiểu được tầm quan trọng của việc normalize trong network. - Biết sử dụng BatchNormalization với PyTorch. - Biết cách tạo ra custom layer với PyTorch.
6	<ul style="list-style-type: none"> - Khái niệm skip connection. - Thực hiện skip connection (cụ thể là residual connection) với PyTorch. 	<ul style="list-style-type: none"> - Hiểu được ứng dụng của skip connection trong việc giảm thiểu vanishing. - Biết xây dựng kiến trúc model có dùng skip connection (cụ thể là residual connection) với PyTorch.
7	<ul style="list-style-type: none"> - Chia model thành từng sub model sau đó dùng chiến thuật train để cho từng submodel học và cuối cùng ghép lại thành model lớn để train lại lần cuối. - Thực hiện chiến thuật trên với PyTorch. 	<ul style="list-style-type: none"> - Hiểu được cách chia model và train từng sub model. - Áp dụng chiến thuật trên với PyTorch.
8	<ul style="list-style-type: none"> - Chuẩn hóa gradient trong quá trình lan truyền ngược để giảm thiểu vấn đề gradient quá nhỏ hoặc quá lớn. - Thực hiện gradient normalization trong PyTorch. 	<ul style="list-style-type: none"> - Hiểu được vai trò của gradient normalization trong việc cải thiện hiệu quả học của mô hình. - Biết cách áp dụng gradient normalization trong PyTorch.

- *Hết* -

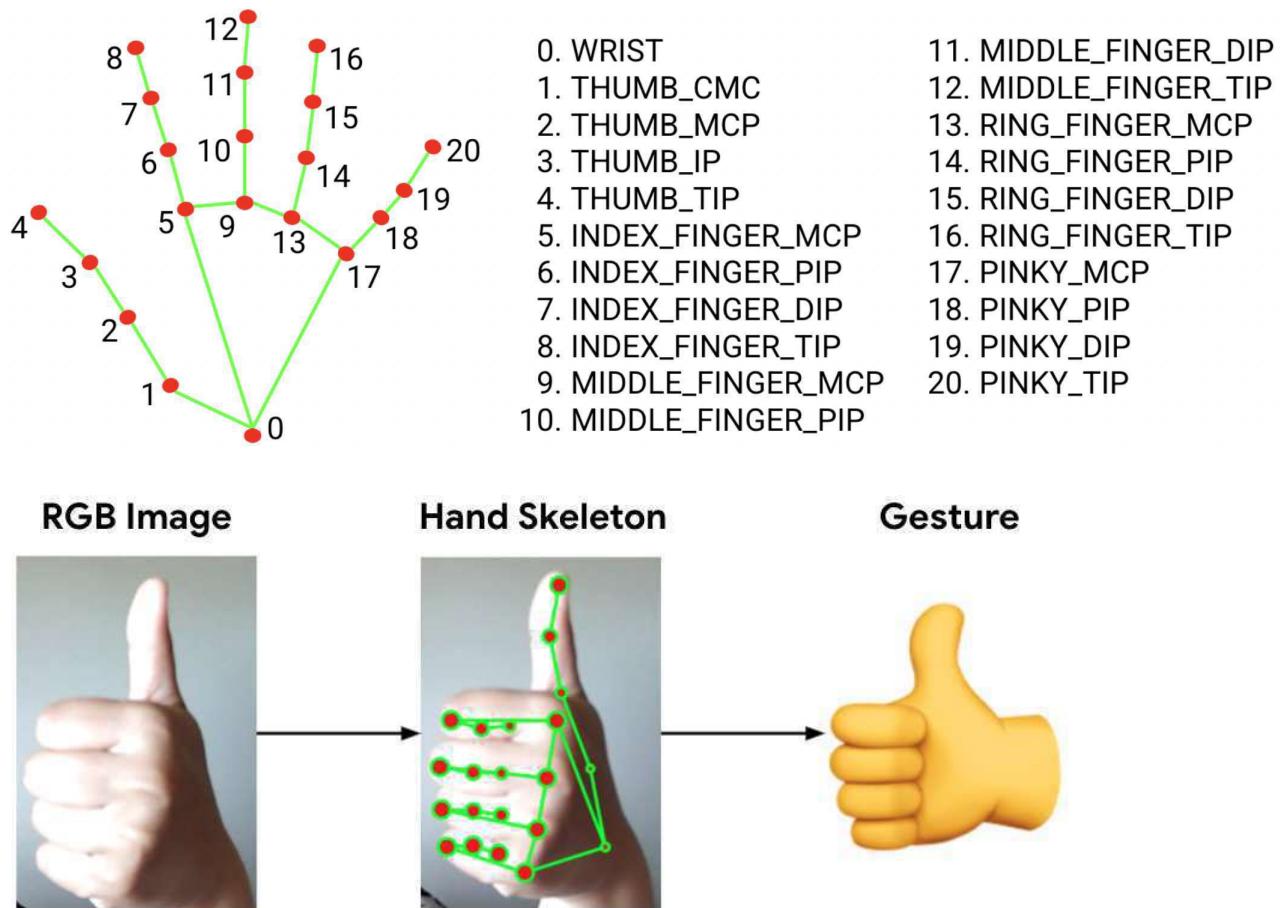
Chương 33

Project 2: Điều khiển thiết bị IoT qua việc nhận dạng các cử chỉ tay

33.1 Giới Thiệu



1. **Tổng quan project** Project Light Controlling Using Hand Gestures nhằm phát triển một hệ thống cho phép người dùng điều khiển đèn thông qua các cử chỉ tay. Project kết hợp công nghệ nhận diện cử chỉ Google với deep learning model để tạo ra một hệ thống nhận diện cử chỉ tay và thực thi lệnh tương ứng với các loại chữ chỉ của tay đã được định nghĩa trước. Project được triển khai qua ba giai đoạn chính:
 - (a) **Step 0 (Chuẩn Bị Data)**: Trong giai đoạn này, chúng tôi sử dụng MediaPipe Gesture Recognizer của Google kết hợp với camera để thu thập dữ liệu về các cử chỉ tay theo các class (class) do chúng tôi định nghĩa. Quá trình này bao gồm:



Hình 33.1: Hand Gesture Recognition

- Thu thập dữ liệu: Ghi lại các cử chỉ tay tương ứng với từng class.
 - Xử lý dữ liệu: Sử dụng MediaPipe để nhận diện và trích xuất đặc trưng của các cử chỉ
 - Lưu trữ dữ liệu: Chia dữ liệu thành bộ train, validation, và test, sau đó lưu vào các file CSV để sử dụng trong quá trình huấn luyện mô hình.
- (b) **Step 1 (Huấn Luyện Mô Hình Phân Loại):** Ở giai đoạn này, chúng tôi xây dựng một mô hình MLP (Multi-Layer Perceptron) để huấn luyện trên bộ dữ liệu đã chuẩn bị:

- Thiết kế mô hình: Xác định kiến trúc của MLP phù hợp với dữ liệu cử chỉ tay.
- Huấn luyện mô hình: Sử dụng bộ dữ liệu train và validation để huấn luyện và tinh chỉnh mô hình.
- Đánh giá mô hình: Kiểm tra hiệu suất của mô hình trên bộ dữ liệu test để đảm bảo độ chính xác trong việc phân loại các cử chỉ.

(c) **Step 2 (Triển Khai Hệ Thống Thực Tế):** Trong giai đoạn cuối, chúng tôi triển khai hệ thống điều khiển đèn theo thời gian thực:

- Nhận diện cử chỉ: Sử dụng webcam để nhận hình ảnh real-time, sau đó áp dụng MediaPipe Gesture Recognizer để trích xuất cử chỉ tay.
- Phân loại cử chỉ: Dựa dữ liệu cử chỉ vào mô hình MLP đã huấn luyện để xác định class tương ứng.
- Điều khiển đèn: Dựa trên class cử chỉ nhận được, thực hiện thao tác tắt/mở đèn trong mô phỏng. (Tùy chọn) Hệ thống có thể điều khiển thiết bị đèn thực tế bằng cách sử dụng module 4 relay giao tiếp qua Modbus RTU RS485.

2. **Giới Thiệu Hand Gestures Recognition** Hand Gestures Recognition là một lĩnh vực quan trọng trong tương tác người và máy, cho phép hệ thống hiểu và phản hồi lại các động tác tay của người dùng. Công nghệ này dựa trên việc phân tích hình ảnh hoặc video để xác định vị trí và hình dạng của bàn tay, từ đó nhận diện các cử chỉ cụ thể.

Các bước chính trong nhận diện cử chỉ tay:

- Phát hiện bàn tay: Xác định vị trí của bàn tay trong khung hình bằng các kỹ thuật xử lý ảnh.
- Trích xuất đặc trưng: Thu thập thông tin về hình dạng, vị trí các ngón tay, góc độ, v.v.
- Phân loại cử chỉ: Sử dụng các mô hình học máy để phân loại cử chỉ dựa trên các đặc trưng đã trích xuất.

Ứng dụng của nhận diện cử chỉ tay:

- Điều khiển thiết bị: Tương tác với máy tính, điện thoại, hoặc các thiết bị thông minh mà không cần chạm.
- Thực tế ảo và tăng cường: Cải thiện trải nghiệm người dùng trong môi trường ảo.
- Hỗ trợ người khuyết tật: Giúp người khuyết tật vận động hoặc giao tiếp dễ dàng hơn.

Công nghệ MediaPipe của Google: MediaPipe là một framework mã nguồn mở cung cấp các giải pháp tiên tiến cho xử lý đa phương tiện thời gian thực. MediaPipe Gesture Recognizer là một trong những giải pháp mạnh mẽ cho nhận diện cử chỉ tay, với khả năng:

- Nhận diện chính xác: Cung cấp mô hình tiên tiến với độ chính xác cao.
- Thực tế ảo và tăng cường: Cải thiện trải nghiệm người dùng trong môi trường ảo.
- Hỗ trợ người khuyết tật: Giúp người khuyết tật vận động hoặc giao tiếp dễ dàng hơn.

33.2 Thực Hành

Step 0: Chuẩn Bị Data và Môi Trường

Trong bước đầu tiên của dự án "Light Controlling Using Hand Gestures", chúng ta sẽ tập trung vào việc chuẩn bị dữ liệu cần thiết để huấn luyện mô hình nhận diện cử chỉ tay. Bước này bao gồm việc thiết lập môi trường làm việc, cài đặt các thư viện cần thiết, cấu hình các class cho các cử chỉ, và thu thập dữ liệu thông qua việc ghi lại các cử chỉ tay sử dụng MediaPipe Gesture Recognizer của Google.

1. Thiết Lập Môi Trường Làm Việc

1.1. Sử Dụng Conda Với Python 3.10: Chúng ta sẽ sử dụng Conda để tạo môi trường ảo cho project với yêu cầu sử dụng Python phiên bản 3.10

- Cài đặt Anaconda hoặc Miniconda: Nếu bạn chưa có Conda trên máy tính, hãy tải và cài đặt từ trang chủ:
 - Anaconda: <https://www.anaconda.com/download>
 - Miniconda: <https://docs.anaconda.com/miniconda/>
- Tạo môi trường mới với Python 3.10:

```
1 conda create -n gesture_env python=3.10.0
```

- Kích hoạt môi trường:

```
1 conda activate gesture_env
```

1.2. Cài Đặt Các Thư Viện Từ requirements.txt: Sau khi môi trường được activate, chúng ta tiến hành cài đặt các thư viện cần thiết được liệt kê trong file requirements.txt.

```
1 pip install -r requirements.txt
```

2. Cấu Hình Các Class Cử Chỉ Trong hand_gesture.yaml

2.1. hand_gesture.yaml:

File hand_gesture.yaml được sử dụng để định nghĩa các class cử chỉ tay sẽ được nhận diện và liên kết với các hành động điều khiển đèn tương ứng.

Việc sử dụng file YAML giúp chúng ta dễ dàng chỉnh sửa và mở rộng danh sách các cử chỉ mà không cần thay đổi code trực tiếp.

Nội dung file:

```
1 gestures:  
2   0: "turn_off"  
3   1: "light1"  
4   2: "light2"  
5   3: "light3"  
6   4: "turn_on"
```

Giải thích:

- **gestures:** Đây là key, chứa danh sách các cử chỉ.
- **0, 1, 2, 3, 4:** Các số đại diện cho class của từng loại cử chỉ.
- **"turn_off", "light1", "light2", "light3", "turn_on":** Tên của các cử chỉ, đồng thời cũng là hành động điều khiển đèn tương ứng.

2.2. Cách Chính Sửa và Mở Rộng: Nếu bạn muốn thêm hoặc chỉnh sửa các cử chỉ, chỉ cần thay đổi nội dung trong file hand_gesture.yaml. Ví dụ, nếu muốn thêm cử chỉ cho "light4":

```
1 gestures:  
2   ...  
3   5: "light4"
```

Sau đó, các bạn cần thu thập dữ liệu cho cử chỉ mới và cập nhật mô hình.

3. Giới Thiệu Về File generate_landmark_data.py

File generate_landmark_data.py là một script Python quan trọng trong dự án, chịu trách nhiệm thu thập dữ liệu về các landmarks của bàn tay khi thực hiện các cử chỉ đã định nghĩa. Dữ liệu này sau đó sẽ được sử dụng để huấn luyện mô hình phân loại trong bước tiếp theo.

3. 1. Tổng Quan các bước thực hiện thu thập cử chỉ data:

- **Ghi lại các cử chỉ tay qua webcam:** Người dùng thực hiện các class cử chỉ trước camera, và sử dụng các phím trên keyboard để ghi lại dữ liệu landmarks của bàn tay tương ứng với các class cử chỉ.
- **Gắn nhãn cho dữ liệu:** Mỗi cử chỉ sẽ được gắn với một nhãn tương ứng dựa theo cấu hình trong file hand_gesture.yaml.

- **Lưu trữ dữ liệu:** Dữ liệu được lưu vào các file CSV để sử dụng cho việc huấn luyện mô hình.

3.2. Import Các Thư Viện Cần Thiết:

```

1 import os
2 import cv2
3 import csv
4 import yaml
5 import numpy as np
6 import mediapipe as mp

```

- **os:** Tương tác với hệ điều hành.
- **cv2:** Thư viện OpenCV để xử lý hình ảnh và video.
- **csv:** Đọc và ghi các file CSV.
- **yaml:** Đọc và ghi các file YAML.
- **numpy:** Xử lý mảng và tính toán số học.
- **mediapipe:** Framework của Google để nhận diện bàn tay và các landmarks.

3.3. is_handsign_character function:

```

1 def is_handsign_character(char: str):
2     return ord('a') <= ord(char) < ord("q") or char == " "

```

- **Chức năng:** Kiểm tra xem ký tự nhập vào có phải là một chữ cái thường từ 'a' đến 'z' hoặc khoảng trắng không.

NOTE: Chúng ta sẽ assign class như trong file yaml config từ 0 trở đi tương ứng với phím 'a' trên keyboard. Ví dụ ta nhấn 'a' tương đương class 0, 'b' tương đương class 1, ... cho đến trước 'q'.

- **Mục đích:** Xác định các phím bấm hợp lệ để bắt đầu hoặc kết thúc việc ghi dữ liệu cho một class cử chỉ.

3.3. label_dict_from_config_file function:

```

1 def label_dict_from_config_file(relative_path):
2     with open(relative_path, "r") as f:
3         label_tag = yaml.full_load(f)["gestures"]
4     return label_tag

```

- **Chức năng:** Đọc file hand_gesture.yaml và trả về một từ điển chứa các nhãn cử chỉ.
- **Mục đích:** Sử dụng các nhãn này để gắn nhãn cho dữ liệu thu thập.

3.4. HandDatasetWriter class:

```

1 class HandDatasetWriter():
2     def __init__(self,filepath) -> None:
3         self.csv_file = open(filepath, "a")
4         self.file_writer = csv.writer(self.csv_file,delimiter
5 = ',', quotechar='|', quoting=csv.QUOTE_MINIMAL)
6     def add(self,hand,label):
7         self.file_writer.writerow([label,*np.array(hand).
8 flatten().tolist()])
9     def close(self):
10        self.csv_file.close()

```

- **Chức năng:** Ghi dữ liệu landmarks của bàn tay và nhãn tương ứng vào file CSV.
- **Method `__init__`:** Khởi tạo và mở file CSV để ghi dữ liệu.
- **Method `add`:** Thêm một dòng dữ liệu mới vào file CSV, bao gồm nhãn và tọa độ các landmarks.
- **Method `close`:** Đóng file CSV sau khi hoàn tất ghi dữ liệu.

3.5. HandLandmarksDetector class:

```

1 class HandLandmarksDetector():
2     def __init__(self) -> None:
3         self.mp_drawing = mp.solutions.drawing_utils
4         self.mp_drawing_styles = mp.solutions.drawing_styles
5         self.mp_hands = mp.solutions.hands
6         self.detector = self.mp_hands.Hands(False,
7 max_num_hands=1,min_detection_confidence=0.5)
8

```

```

8     def detectHand(self, frame):
9         hands = []
10        frame = cv2.flip(frame, 1)
11        annotated_image = frame.copy()
12        results = self.detector.process(cv2.cvtColor(frame,
13                                              cv2.COLOR_BGR2RGB))
14        if results.multi_hand_landmarks is not None:
15            for hand_landmarks in results.
16            multi_hand_landmarks:
17                hand = []
18                self.mp_drawing.draw_landmarks(
19                    annotated_image,
20                    hand_landmarks,
21                    self.mp_hands.HAND_CONNECTIONS,
22                    self.mp_drawing_styles.
23                    get_default_hand_landmarks_style(),
24                    self.mp_drawing_styles.
25                    get_default_hand_connections_style())
26                    for landmark in hand_landmarks.landmark:
27                        x,y,z = landmark.x,landmark.y,landmark.z
28                        hand.extend([x,y,z])
29                hands.append(hand)
30        return hands, annotated_image

```

- **Chức năng:** Phát hiện bàn tay trong khung hình và trích xuất các landmarks.
- **Method `__init__`:** Khởi tạo các thành phần cần thiết của MediaPipe để nhận diện bàn tay.
- **Method `detectHand`:**
 - Chuyển đổi frame từ BGR sang RGB.
 - Sử dụng MediaPipe để nhận diện bàn tay và trích xuất các landmarks.
 - Vẽ các landmarks và kết nối lên hình ảnh để hiển thị.
 - Trả về danh sách các landmarks và hình ảnh đã vẽ.

3.6. run function:

```

1 def run(data_path, sign_img_path, split="val", resolution
2       =(1280,720)):

```

```
3     hand_detector = HandLandmarksDetector()
4     cam = cv2.VideoCapture(0)
5     cam.set(3, resolution[0])
6     cam.set(4, resolution[1])
7
8     os.makedirs(data_path, exist_ok=True)
9     os.makedirs(sign_img_path, exist_ok=True)
10    print(sign_img_path)
11    dataset_path = f"./{data_path}/landmark_{split}.csv"
12    hand_dataset = HandDatasetWriter(dataset_path)
13    current_letter= None
14    status_text = None
15    cannot_switch_char = False
16
17
18    saved_frame = None
19    while cam.isOpened():
20        _,frame = cam.read()
21        hands,annotated_image = hand_detector.detectHand(
frame)
22
23        if(current_letter is None):
24            status_text = "press a character to record"
25
26        else:
27            label = ord(current_letter)-ord("a")
28            if label == -65:
29                status_text = f"Recording unknown , press
spacebar again to stop"
30                label = -1
31            else:
32                status_text = f"Recording {LABEL_TAG[label]}, 
press {current_letter} again to stop"
33
34        key = cv2.waitKey(1)
35        if(key == -1):
36            if(current_letter is None ):
37                # no current letter recording , just skip it
38                pass
39            else:
40                if len(hands) != 0:
41                    hand = hands[0]
42                    hand_dataset.add(hand=hand,label=label)
43                    saved_frame = frame
44        # some key is pressed
```

```

45         else:
46             # pressed some key, do not push this image,
47             # assign current letter to the key just pressed
48             key = chr(key)
49             if key == "q":
50                 break
51             if (is_handsign_character(key)):
52                 if(current_letter is None):
53                     current_letter = key
54                 elif(current_letter == key):
55                     # pressed again?, reset the current state
56                     if saved_frame is not None:
57                         if label >=0:
58                             cv2.imwrite(f"./{sign_img_path}/{LABEL_TAG[label]}.jpg", saved_frame)
59
60                         cannot_switch_char=False
61                         current_letter = None
62                         saved_frame = None
63                     else:
64                         cannot_switch_char = True
65                         # warned user to unbind the
66                         current_letter first
67                         if(cannot_switch_char):
68                             cv2.putText(annotated_image, f"please press {current_letter} again to unbind", (0,450), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)
69
70                             cv2.putText(annotated_image, status_text, (5,20), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)
71                             cv2.imshow(f"{split}", annotated_image)
72                             cv2.destroyAllWindows()

```

- **Chức năng:** Vòng lặp chính để thu thập dữ liệu cử chỉ.
- **Tham số:**
 - *data_path*: Đường dẫn lưu trữ dữ liệu CSV.
 - *sign_img_path*: Đường dẫn lưu trữ hình ảnh cử chỉ. **NOTE:** Chỉ có 1 ảnh cuối cùng của mỗi class sẽ được lưu vào đường dẫn để người dùng kiểm tra xem ảnh và class có đúng không.
 - *split*: Loại dữ liệu (train, val, test).
 - *resolution*: Độ phân giải của webcam.

- Flow:

- Khởi tạo HandLandmarksDetector() để nhận diện bàn tay và mở webcam.
- Tạo các thư mục cần thiết.
- Khởi tạo các biến trạng thái.
- Main loop:
 - * Đọc frame từ webcam.
 - * Nhận diện bàn tay và trích xuất landmarks.
 - * Xử lý trạng thái ghi dữ liệu dựa trên phím bấm.

NOTE:: Các landmarks và connection giữa các landmarks sẽ được vẽ lên frame và chúng ta có thể quan sát được các landmark này. Từ đó, chúng ta sẽ sử dụng các phím để ghi nhận các class cử chỉ tương ứng. Ví dụ, với 5 class cử chỉ, chúng ta bắt đầu bằng cử chỉ class 0 và nhấn phím 'a'. Sau đó, di chuyển bàn tay nhưng giữ nguyên dáng cử chỉ class 0 để camera liên tục chụp ảnh và thu thập các điểm đánh dấu của cử chỉ này vào file CSV. Quá trình này thường kéo dài từ 10 đến 30 giây cho mỗi class. Khi đã thu thập đủ dữ liệu cho class 0, nhấn 'a' một lần nữa để thoát chế độ thu thập dữ liệu của class đó. Tiếp theo, nhấn 'b' để bắt đầu thu thập dữ liệu cho cử chỉ của class 1 và lặp lại quy trình tương tự cho các class còn lại. Sau khi thu thập xong ta sẽ nhấn phím 'q' để thoát chương trình

- * Ghi dữ liệu khi cần thiết.
- * Hiển thị thông tin lên màn hình.
- Kết thúc và giải phóng tài nguyên.

3.7. main function:

```

1 if __name__ == "__main__":
2     LABEL_TAG = label_dict_from_config_file("hand_gesture.
yaml")
3     data_path = './data2'
4     sign_img_path = './sign_imgs2'
5     run(data_path, sign_img_path, "train", (1280,720))
6     run(data_path, sign_img_path, "val", (1280,720))
7     run(data_path, sign_img_path, "test", (1280,720))

```

- **Chức năng:** Điểm bắt đầu của chương trình.
- **Flow:**
 - Đọc class cử chỉ từ file hand_gesture.yaml.
 - Thiết lập đường dẫn lưu trữ dữ liệu và hình ảnh.
 - Gọi hàm run 3 lần để xây dựng train, val, test data.

3.8. Cách sử dụng generate_landmark_data.py để xây dựng train/val/test data:

1. Khởi Chạy Chương Trình:

- Mở terminal và chạy lệnh

```
1 python generate_landmark_data.py
```

2. Thu Thập Dữ Liệu

- **Bắt đầu ghi dữ liệu cho một cử chỉ:**
 - Nhấn phím tương ứng với cử chỉ muốn ghi (từ 'a' đến trước 'q').
 - Ví dụ: Nhấn phím 'a' để bắt đầu ghi dữ liệu cho cử chỉ có class 0 như đã định nghĩa ở file config hand_gesture.yaml.
- **Thực hiện cử chỉ trước camera:**
 - Đưa tay vào khung hình và thực hiện cử chỉ tương ứng.
 - Dữ liệu sẽ được tự động ghi khi cử chỉ đang được ghi và bàn tay được phát hiện.
- **Kết thúc ghi dữ liệu cho một cử chỉ:**
 - Nhấn lại phím đã chọn để dừng ghi dữ liệu cho cử chỉ đó.
- **Chuyển sang cử chỉ khác:**
 - Lặp lại quá trình trên cho các cử chỉ khác.
- **Kết thúc quá trình thu thập dữ liệu cho train hoặc val hoặc test:**
 - Sau khi thu thập đúng theo số lượng class trong file config hand_gesture.yaml. Các bạn nhấn phím 'q' kết thúc.

- Chúng ta sẽ thực công việc như trên 3 lần tương ứng với train, val và test data. Khi train kết thúc sẽ tự động đến val, rồi đến test sau đó sẽ tự thoát chương trình và ta thu được 3 file CSV của 3 tập data.

3. Lưu Ý Khi Thu Thập Dữ Liệu

- **Không thể chuyển đổi cử chỉ khi đang ghi:**
 - Nếu muốn chuyển sang cử chỉ khác, cần nhấn lại phím hiện tại để dừng ghi, sau đó nhấn phím mới.
- **Hiển thị thông tin trạng thái:**
 - Trạng thái hiện tại sẽ được hiển thị trên màn hình, giúp người dùng biết đang ghi cử chỉ nào.
- **Lưu hình ảnh mẫu của cử chỉ:**
 - Hình ảnh cuối cùng của cử chỉ sẽ được lưu lại trong thư mục sign_imgs để tham khảo.
- **Trong trường hợp muốn thoát nhanh chương trình:**
 - Nhấn 'q' liên tục cho đến khi thoát khỏi chương trình và xoá các file data để thực hiện lại từ đầu.

4. Dữ Liệu Được Lưu Trữ

- **File CSV:**
 - Dữ liệu landmarks và nhãn cử chỉ được lưu trong các file CSV trong thư mục data.
 - Có ba file tương ứng với các loại dữ liệu: landmark_train.csv, landmark_val.csv, landmark_test.csv
- **Hình ảnh mẫu:**
 - Hình ảnh của cử chỉ được lưu trong thư mục sign_imgs.

File generate_landmark_data.py là công cụ quan trọng giúp chúng ta thu thập và chuẩn bị dữ liệu cần thiết cho việc huấn luyện mô hình nhận diện cử chỉ tay. Bằng cách sử dụng script này, chúng ta có thể:

- **Tự động hóa quá trình thu thập dữ liệu:** Giảm thiểu công sức và thời gian so với việc ghi dữ liệu thủ công.

- Đảm bảo tính nhất quán của dữ liệu: Dữ liệu được thu thập và lưu trữ theo một định dạng chuẩn, dễ dàng cho việc tiền xử lý và huấn luyện mô hình.
- Dễ dàng mở rộng và tùy chỉnh: Có thể thêm hoặc chỉnh sửa các cử chỉ bằng cách cập nhật file hand_gesture.yaml và thu thập dữ liệu mới.

Tiếp theo, sau khi đã thu thập và chuẩn bị dữ liệu, chúng ta sẽ chuyển sang Bước 1 để xây dựng và huấn luyện model phân loại cử chỉ tay bằng cách xây dựng MLP (Multi-Layer Perceptron) model đơn giản. Model này sẽ học từ dữ liệu chúng ta đã thu thập

Step 1: Xây dựng và huấn luyện mô hình phân loại cử chỉ tay

Sau khi đã hoàn thành việc chuẩn bị dữ liệu ở bước 0, chúng ta chuyển sang bước tiếp theo là xây dựng và huấn luyện mô hình phân loại cử chỉ tay sử dụng MLP (Multi-Layer Perceptron). Mục tiêu của bước này là tạo ra một mô hình có khả năng nhận diện chính xác các cử chỉ tay dựa trên dữ liệu landmarks thu thập được. File notebook hand_gesture_recognition huấn luyện mô hình MLP dựa trên cấu hình các class cử chỉ trong hand_gesture.yaml và dữ liệu landmarks từ các file CSV để học cách phân loại cử chỉ tay.

1. Quy Trình Huấn Luyện

1. Load dữ liệu

- **Dữ liệu được chia thành ba tập:** train, val, và test đã được thu thập ở step 0. Nếu các bạn không thực hiện được thì có thể sử dụng data được cung cấp link download ở trong notebook
- Sử dụng class CustomImageDataset để đọc dữ liệu từ các file CSV và tạo bộ dữ liệu cho PyTorch.

2. Xây Dựng Mô Hình MLP

- Mô hình gồm nhiều layer Linear và ReLU để học các đặc trưng phức tạp từ dữ liệu.

- Sử dụng BatchNorm và Dropout để cải thiện hiệu suất và ngăn overfitting.

3. Huấn Luyện Mô Hình

- Sử dụng hàm loss CrossEntropyLoss
- Sử dụng bộ Adam optimizer
- Sử dụng Early Stopping để dừng huấn luyện khi mô hình không còn cải thiện trên tập val.
- Theo dõi accuracy của train và val sau mỗi epoch.

4. Đánh Giá Mô Hình

- Sau khi huấn luyện, sử dụng mô hình tốt nhất để đánh giá trên tập test.

NOTE: Các bạn phải hoàn thành được các câu hỏi bên dưới thì sẽ thực hiện được viên train mô hình để phân loại cử chỉ

Câu hỏi 13: Hoàn thành đoạn code bên dưới để xây dựng một model gồm có 4 hidden layer, lần lượt input và output là (63, 128), (128, 128), (128, 128), (128, 128). Layer đầu tiên được theo sau bởi một Relu và Batchnorm1d. Layer thứ 2, 3, và 4 được theo sau bởi Relu và Dropout với rate lần lượt là 0.4, 0.4, 0.6. Output layer có nhiệm vụ phân loại với input là 128 và output là số lượng class cử chỉ:

(A)

```
1 nn.Linear(63, 128),
2 nn.ReLU(),
3 nn.BatchNorm1d(128),
4 nn.Linear(128, 128),
5 nn.ReLU(),
6 nn.Dropout(p=0.4),
7 nn.Linear(128, 128),
8 nn.ReLU(),
9 nn.Dropout(p=0.4),
10 nn.Linear(128, 128),
11 nn.ReLU(),
12 nn.Dropout(p=0.6),
13 nn.Linear(128, len(list_label)),
```

(B)

```
1 nn.Linear(63, 128),  
2 nn.ReLU(),  
3 nn.Linear(128, 128),  
4 nn.ReLU(),  
5 nn.BatchNorm1d(128),  
6 nn.Dropout(p=0.5),  
7 nn.Linear(128, 128),  
8 nn.ReLU(),  
9 nn.Linear(128, len(list_label)),  
10 nn.Dropout(p=0.3),
```

(C)

```
1 nn.Linear(63, 128),  
2 nn.ReLU(),  
3 nn.BatchNorm1d(128),  
4 nn.Linear(128, 128),  
5 nn.Dropout(p=0.3),  
6 nn.ReLU(),  
7 nn.Linear(128, 128),  
8 nn.ReLU(),  
9 nn.Dropout(p=0.5),  
10 nn.Linear(128, 128),  
11 nn.ReLU(),  
12 nn.Linear(128, len(list_label)),
```

(D)

```
1 nn.Linear(63, 256),  
2 nn.ReLU(),  
3 nn.BatchNorm1d(256),  
4 nn.Linear(256, 256),  
5 nn.ReLU(),  
6 nn.Dropout(p=0.4),  
7 nn.Linear(256, 128),  
8 nn.ReLU(),  
9 nn.Dropout(p=0.4),  
10 nn.Linear(128, 128),  
11 nn.ReLU(),  
12 nn.Dropout(p=0.6),  
13 nn.Linear(128, len(list_label)),
```

Câu hỏi 14: Hoàn thành code để thực hiện forward dự đoán cùi chỉ với input

x. Thực hiện flatten x. Pass x vừa flatten vào linear_relu_stack. Return logits (outputs từ layer cuối cùng)

(A)

```
1 x = self.flatten(x)
2 x = self.linear_relu_stack(x)
3 return x
```

(B)

```
1 x = self.linear_relu_stack(x)
2 return self.flatten(x)
```

(C)

```
1
2 return self.linear_relu_stack(x)
```

(D)

```
1 x = self.flatten(x)
2 self.linear_relu_stack.forward(x)
3 return
```

Câu hỏi 15: Hoàn thành code để thực hiện khởi tạo DataLoader cho trainset với batch_size 40 và cho phép xáo trộn

(A)

```
1
2 trainloader = torch.utils.data.DataLoader(trainset,batch_size
    =40,shuffle=True)
```

(B)

```
1
2 trainloader = torch.utils.data.DataLoader(trainset,batch_size
    =50,shuffle=True)
```

(C)

```
1
2 trainloader = torch.utils.data.DataLoader(trainset,batch_size
    =20,shuffle=False)
```

(D)

```
1  
2 trainloader = torch.utils.data.DataLoader(dataset=trainset,  
batch_size=40, shuffle=False)
```

Câu hỏi 16: Hoàn thành code để thực hiện cấu hình Adam optimizer cho các tham số của model với tốc độ học là 0.0001

(A)

```
1  
2 optimizer = optim.Adam(model.parameters(), lr=0.0001)
```

(B)

```
1  
2 optimizer = optim.Adam(model, lr=0.0001)
```

(C)

```
1  
2 optimizer = optim.Adam(model.parameters, lr=0.0001)
```

(D)

```
1  
2 optimizer = optim.Adam([model.parameters], lr=0.0001)
```

Câu hỏi 17: Hoàn thành đoạn code bên dưới để print ra epoch hiện tại và minimum watched metric và thoát loop (A)

```
1  
2 print(f"stopping at epoch {epoch}, minimum: {early_stopper.  
watched_metrics}")  
3 break
```

(B)

```
1  
2 print("Early stopping triggered")  
3 break
```

(C)

```
1  
2 print(f"Stopping training at epoch {epoch} due to minimal  
loss change")
```

(D)

```
1
2 print(f"stopping at epoch {epoch}, minimum: {early_stopper.
    watched_metrics}")
```

Câu hỏi 18: Hoàn thành code để thực hiện khởi tạo DataLoader cho testset với batch_size 20 và không cho phép xáo trộn

(A)

```
1
2 test_loader = torch.utils.data.DataLoader(testset, batch_size
    =20, shuffle=False)
```

(B)

```
1
2 test_loader = torch.utils.data.DataLoader(testset, batch_size
    =20, shuffle=True)
```

(C)

```
1
2 test_loader = torch.utils.data.DataLoader(valet, batch_size
    =20, shuffle=False)
```

(D)

```
1
2 test_loader = torch.utils.data.DataLoader(trainset,
    batch_size=20, shuffle=False)
```

Câu hỏi 19: Hoàn thành code bên dưới để predict class của cùi chỉ và update accuracy với kết quả predict và true labels

(A)

```
1
2 preds = network(test_input)
3 acc_test.update(preds, test_label)
```

(B)

```
1
2 preds = network(train_input)
3 acc_test.update(preds, test_label)
```

(C)

```
1
2 preds = network(val_input)
3 acc_test.update(preds, test_label)
```

(D)

```
1
2 preds = network(train_input)
3 acc_test.update(preds, val_label)
```

Câu hỏi 20: Hoàn thành code để thực hiện khởi tạo NeuralNetwork model đã xây dựng ở trên, khởi tạo hàm loss sử dụng CrossEntropyLoss và khởi tạo early stopper với patience là 30 và min_delta là 0.01

(A)

```
1
2 model = NeuralNetwork()
3 loss_function = nn.CrossEntropyLoss()
4 early_stopper = EarlyStopper(patience=30, min_delta=0.01)
```

(B)

```
1 model = nn.CrossEntropyLoss()
2 loss_function = NeuralNetwork()
3 early_stopper = EarlyStopper(patience=30, min_delta=0.01)
```

(C)

```
1 model = NeuralNetwork()
2 loss_function = EarlyStopper(patience=30, min_delta=0.01)
3 early_stopper = nn.CrossEntropyLoss()
```

(D)

```
1 model = EarlyStopper(patience=30, min_delta=0.01)
2 loss_function = nn.CrossEntropyLoss()
3 early_stopper = NeuralNetwork()
```

Câu hỏi 21: Hoàn thành code để thực hiện reset gradients và dự đoán class cử chỉ của inputs

(A)

```
1
```

```
2 optimizer.zero_grad()  
3 preds = model(inputs)
```

(B)

```
1  
2 optimizer.clear()  
3 preds = model(inputs)
```

(C)

```
1  
2 optimizer.clean()  
3 preds = model(inputs)
```

(D)

```
1  
2 optimizer.reset()  
3 preds = model(labels)
```

Câu hỏi 22: Hoàn thành code để thực hiện tính loss dựa vào kết quả dự đoán và labels, sau đó thực hiện backward và update parameters thông qua optimizer

(A)

```
1  
2 loss = loss_function(preds, labels)  
3 loss.backward()  
4 optimizer.step()
```

(B)

```
1  
2 loss.backward()  
3 loss = loss_function(preds, labels)  
4 optimizer.update()
```

(C)

```
1  
2 loss = loss_function(preds, labels)  
3 optimizer.backward()  
4 loss.step()
```

(D)

```
1  
2 optimizer.update()  
3 loss = loss_function(preds,labels)  
4 loss.backward()
```

Việc huấn luyện model thành công là nền tảng để chúng ta chuyển sang bước 2, nơi mô hình sẽ được triển khai để nhận diện cử chỉ tay trong thời gian thực và điều khiển đèn theo yêu cầu.

Step 2: Triển khai model và điều khiển đèn bằng cử chỉ tay

Sau khi đã huấn luyện thành công mô hình phân loại cử chỉ tay ở bước 1, chúng ta chuyển sang bước cuối cùng của dự án: triển khai hệ thống nhận diện cử chỉ tay trong thời gian thực và điều khiển đèn theo cử chỉ. Bước này bao gồm việc sử dụng webcam để thu nhận hình ảnh, nhận diện cử chỉ tay thông qua mô hình đã huấn luyện, và thực hiện điều khiển đèn tương ứng.

Trong phần này, chúng ta sẽ phân tích chi tiết hai file chính:

- detect_simulation.py: File chính để nhận diện cử chỉ và điều khiển đèn.
- controller.py: Module hỗ trợ việc điều khiển thiết bị phần cứng thông qua giao tiếp Modbus RTU RS485.
- **Mục Tiêu của Bước 2**
 - **Nhận diện cử chỉ tay trong thời gian thực:** Sử dụng webcam để thu nhận hình ảnh và áp dụng mô hình đã huấn luyện để nhận diện cử chỉ.
 - **Điều khiển đèn dựa trên cử chỉ:** Thực hiện thao tác tắt/mở đèn trong mô phỏng hoặc điều khiển thiết bị đèn thực tế thông qua module relay và giao tiếp Modbus RTU RS485.
 - **Cung cấp tùy chọn cho người dùng:** Cho phép chạy ở chế độ mô phỏng (không cần phần cứng) hoặc chế độ thực tế (có phần cứng).
- **Tùy Chọn Chế Độ Hoạt Động**
 - **Chế độ mô phỏng** (device=False): Nếu bạn không có phần cứng, chương trình sẽ hiển thị mô phỏng ba đèn trên khung hình webcam.

- **Chế độ thực tế** (device=True): Nếu bạn có phần cứng (module 4 relay giao tiếp Modbus RTU RS485), chương trình sẽ điều khiển đèn thực tế thông qua module này.
- **Phần cứng (trong trường hợp các bạn muốn chạy chế độ thực tế)**
 - **4 relay giao tiếp Modbus RTU RS485:** Dùng để nhận tín hiệu từ laptop/PC chạy model để điều khiển 3 đèn tắt/mở
 - **Mạch chuyển đổi giao tiếp USB to RS485:** Giúp giao tiếp giữa laptop/PC với module 4 relay giao tiếp Modbus RTU RS485
 - **3 bóng đèn và chuôi đèn:** dùng chuôi đèn gắn vào bóng đèn và nối dây vào module 4 relay giao tiếp Modbus RTU RS485

Trong file detect_simulation.py thì class LightGesture có nhiệm vụ nhận diện ra cử chỉ của tay và điều khiển đèn

- **Thuộc tính:**

- device: Biến boolean xác định chế độ hoạt động (mô phỏng hoặc thực tế).
- detector: Đối tượng HandLandmarksDetector để nhận diện bàn tay.
- signs: Dictionary chứa các nhãn cử chỉ từ file hand_gesture.yaml.
- classifier: Mô hình NeuralNetwork đã được tải trọng số từ quá trình huấn luyện.
- controller: Đối tượng ModbusMaster để điều khiển phần cứng (chỉ khởi tạo khi device=True).
- light1, light2, light3: Trạng thái của các đèn (True là bật, False là tắt).

- **Method run**

- Khởi tạo webcam và thiết lập độ phân giải.
- Main Loop:
 - * Đọc frame từ webcam.

- * Sử dụng detector để nhận diện bàn tay và trích xuất landmarks.
- * Nếu phát hiện bàn tay:
 - Chuyển landmarks thành tensor và đưa vào mô hình classifier để dự đoán cử chỉ.
 - Dựa trên class cử chỉ dự đoán, cập nhật trạng thái các đèn trên mô phỏng và điều khiển phần cứng (nếu có).
 - Cập nhật status_text để hiển thị cử chỉ hiện tại.
- * Nếu không phát hiện bàn tay: Đặt status_text là None.
- * Gọi method light_device để vẽ mô phỏng đèn lên khung hình.
- * Hiển thị khung hình và status_text
- * Kiểm tra phím bấm để thoát chương trình (nhấn phím "q" để thoát).

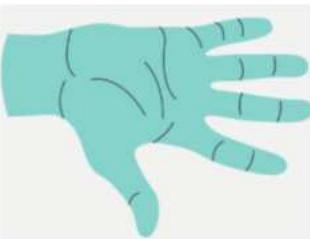
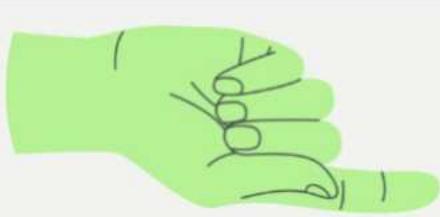
Hàm main: Tạo đối tượng LightGesture và chạy phương thức run.

- Thay thế model_path bằng đường dẫn tới mô hình đã huấn luyện và lưu trữ ở bước 1.
- Đặt device=False để chạy ở chế độ mô phỏng, device=True để chạy ở chế độ thực tế.

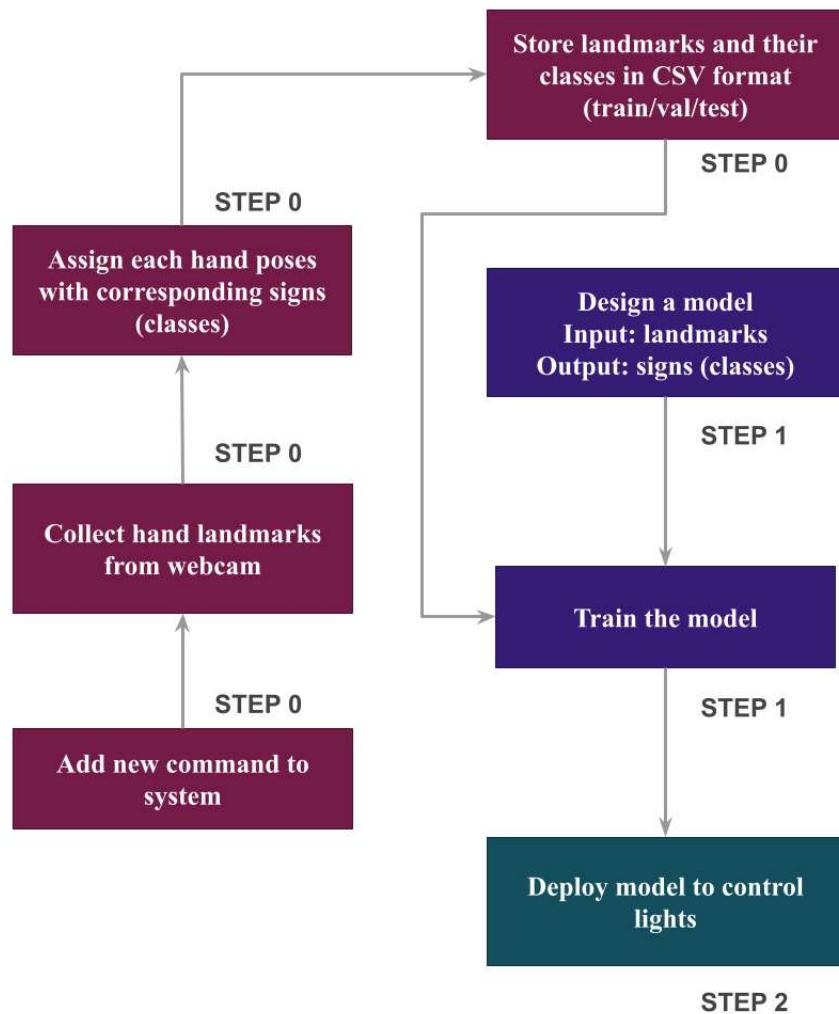
33.3 Phụ Lục

1. Code để thực hiện **step 0** [Link](#), Video hướng dẫn sử dụng generate_landmark_data.py để tạo train/val/test data [Link](#)
2. Code notebook (hint) [Link](#) để thực hiện **step 1** các bạn cần upload file data chạy được ở step 1 (nếu không chạy được ở step 0 các bạn có thể download data ở [Link](#)) và file config hand_gesture.yaml để chạy với notebook.
3. Code để thực hiện **step 2** . [Link](#). Video demo kết quả cuối cùng với thiết bị sử dụng file detect_simulation.py (cùng là một file với main.py trong demo) [Link](#)
4. **Solution:** Các file code cài đặt hoàn chỉnh và phà trả lời nội dung trắc nghiệm có thể tả tại [Link](#) (**Lưu ý** Sáng thứ 3 khi hết deadline phần bài tập ad mới copy các nội dung bài giải nêu trên vào đường dẫn)
5. **Rubric:**

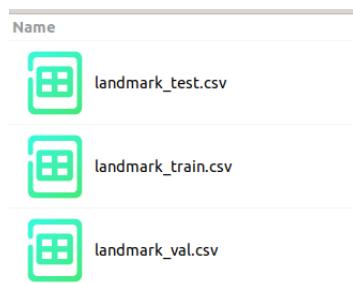
— Hết —

Class	Action	Hand Gesture
0	Turn All the Lights Off	
1	Turn the Lights 1 On	
2	Turn the Lights 2 On	
3	Turn the Lights 3 On	
4	Turn All the Lights On	

Hình 33.2: Class và action tương ứng với các cử chỉ để điều khiển đèn¹⁰⁴⁹



Hình 33.3: Các bước thực hiện project



Hình 33.4: Sau khi chạy chương trình sẽ thu được train, val và test CSV file

```

class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        list_label = label_dict_from_config_file("hand_gesture.yaml")

        self.linear_relu_stack = nn.Sequential(
            ##### Your Code Here #####
            '''Hoàn thành đoạn code để xây dựng một model gồm có 4 hidden layer,
            lâñ lượng input và output là (63, 128), (128, 128), (128, 128), (128, 128).
            Layer đầu tiên được theo sau bởi một Relu và Batchnorm1d.
            Layer thứ 2, 3, và 4 được theo sau bởi Relu và Dropout với rate lâñ lượt là 0.4, 0.4, 0.
            Output layer có nhiệm vụ phân loại với input là 128 và output là số lượng class cù'chỉ
            '''
            #####
        )

    def forward(self, x):
        ##### Your Code Here #####
        ''' Hoàn thành code để thực hiện forward dự đoán cù'chỉ với input x.
        Thực hi'ent flatten x
        Pass x vừa flatten vào linear_relu_stack
        Return logits (outputs từ layer cuối cùng)
        '''
        #####

```

```

trainset = CustomImageDataset(train_path)
##### Your Code Here #####
'''Hoàn thành code để thực hiện khởi tạo DataLoader cho trainset với
batch_size 40 và cho phép xáo trộn
'''
trainloader =
#####

```

```
##### Your Code Here #####
'''Hoàn thành code để thực hiện cấu hình Adam optimizer cho các tham số của
model với tốc độ học là 0.0001
...
optimizer =
#####
```

```
if early_stopper.early_stop(avg_vloss):
    ##### Your Code Here ##### Q5
    ''' Hoàn thành đoạn code bên dưới để print ra epoch hiện tại và
    minimum watched metric và thoát loop
    ...
#####

```

```
def forward(self, x):
    ##### Your Code Here ##### Q2
    ''' Hoàn thành code để thực hiện forward dự đoán cùi chỉ với input x.
    Thực hiện flatten x
    Pass x vừa flatten vào linear_relu_stack
    Return logits (outputs từ layer cuối cùng)
    ...
#####

```

```
##### Your Code Here ##### Q7
'''Hoàn thành code bên dưới để predict class của cùi chỉ và update accuracy
với kết quả predict và true labels
...
preds =
#####

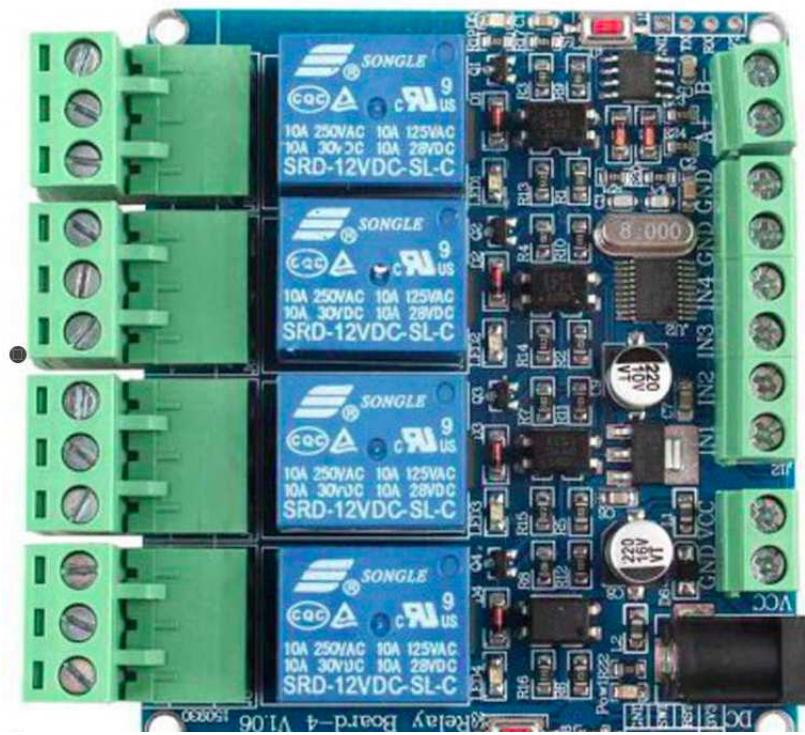
```

```
##### Your Code Here ##### Q8
'''Hoàn thành code để thực hiện khởi tạo NeuralNetwork model đã xây dựng ở trên,
khởi tạo hàm loss sử dụng CrossEntropyLoss và khởi tạo early stopper với patience
là 30 và min_delta là 0.01
...
model =
loss_function =
early_stopper =
#####

```

```
#####
# Your Code Here #####
''' Hoàn thành code để thực hiện reset gradients và dự đoán class của inputs
...
preds =
#####
```

```
#####
# Your Code Here #####
''' Hoàn thành code để thực hiện tính loss dựa vào kết quả dự đoán và labels, sau đó thực hiện backward và update parameters thông qua optimizer
...
loss =
#####
```





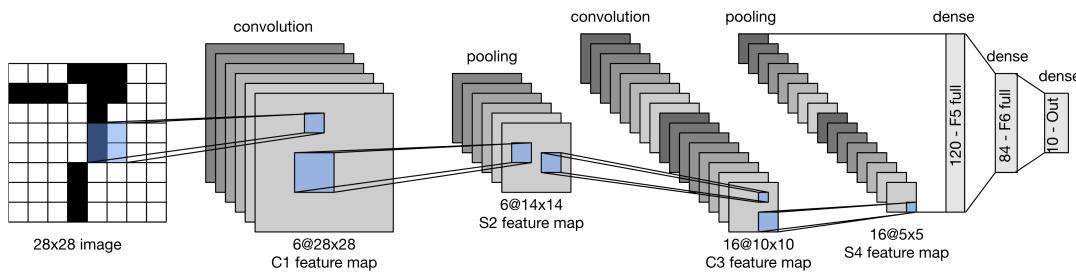
Phân VIII

Module 8: Những kiến trúc deep learning

Chương 34

Convolution Neural Network

34.1 Giới thiệu



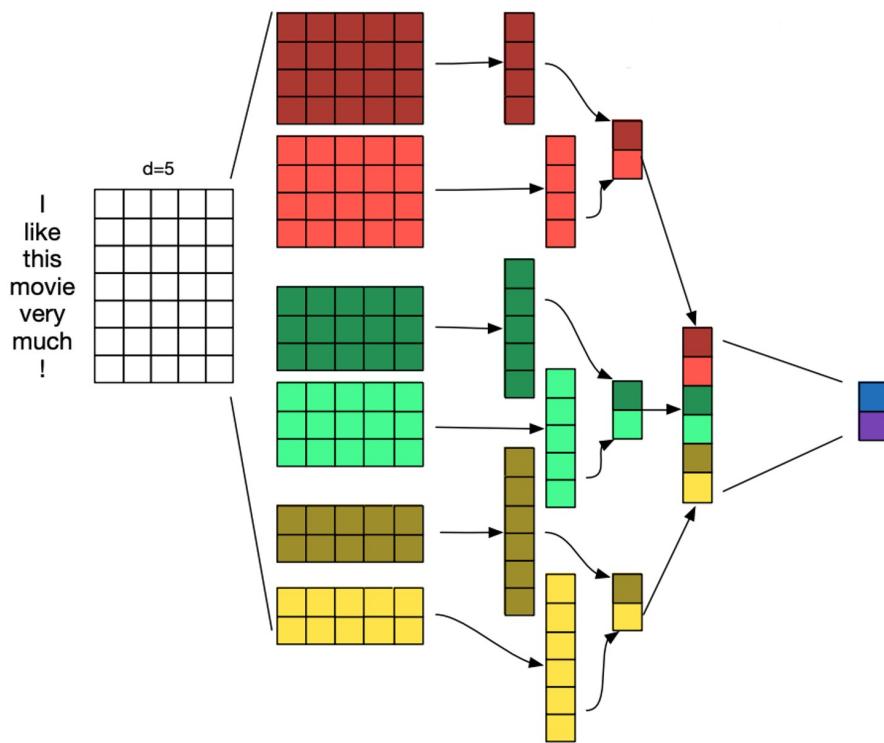
Hình 34.1: Mô hình LeNet cho bài toán phân loại ảnh trên bộ dữ liệu MNIST.

Trong xử lý ảnh, mỗi ảnh có hàng ngàn pixels, mỗi pixel được xem như 1 feature, vậy nếu như một bức ảnh có kích thước $1000 * 1000$, thì sẽ có 1.000.000 features. Với normal feed-forward neural networks, mỗi layer là full-connected với previous input layer. Trong normal feed-forward neural network, với mỗi layer, có 1.000.000 pixels, mỗi pixel lại kết nối full-connected với 1.000.000 pixels ở layer trước, tức sẽ có 10^{12} tham số. Đây là còn số quá lớn để có thể tính được vào thời điểm đó, bởi vì với mô hình có nhiều tham số, sẽ dễ bị overfitted và cần lượng lớn data cho việc training, ngoài ra còn cần nhiều memory và năng lực tính toán cho việc training và prediction.

Vì vậy, sự ra đời CNN giúp xây dựng các model giải quyết hiệu quả với dữ liệu ảnh. Có 2 đặc tính của image hình thành nên cách hoạt động của CNN trên image, đó là feature localization và feature independence of location.

Feature localization: mỗi pixel hoặc feature có liên quan với các pixel quanh nó.

Feature Independence of location: mỗi feature dù nó có nằm ở đâu trong bức ảnh, thì nó vẫn mang giá trị của feature đó. CNN xử lý vấn đề có quá nhiều tham số với Shared parameters (feature independence of location) của Locally connected networks (feature localization), được gọi là Convolution



Hình 34.2: Mô hình TextCNN cho bài toán phân loại văn bản trên bộ dữ liệu NTC-SCV.

Net.

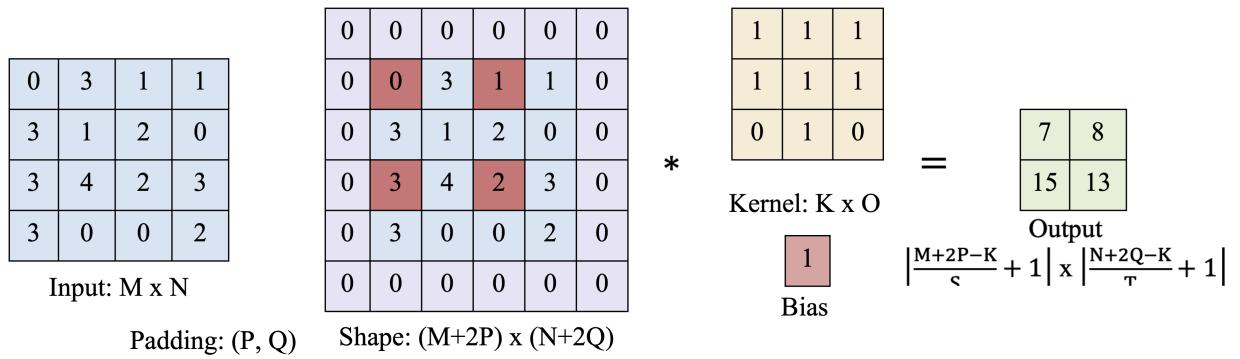
Locally connected layer: Trong hidden layer đầu tiên, mỗi node sẽ kết nối tới một cụm nhỏ pixels của input image chứ không phải toàn bộ image, gọi là small portion. Theo cách này, ta sẽ có ít kết nối hơn, vì thế ít tham số hơn giữa input và hidden layer đầu tiên.

Shared parameters: Có nhưng khu vực, mà việc tìm ra feature là giống nhau về cách làm, vì vậy ta có thể dùng chung bộ parameter, trong hình trên là phía phải bên trên và phía trái bên dưới. Tức ta chia sẻ bộ parameter giữa những vị trí khác nhau trong bức ảnh.

CNN có 3 thành phần chính:

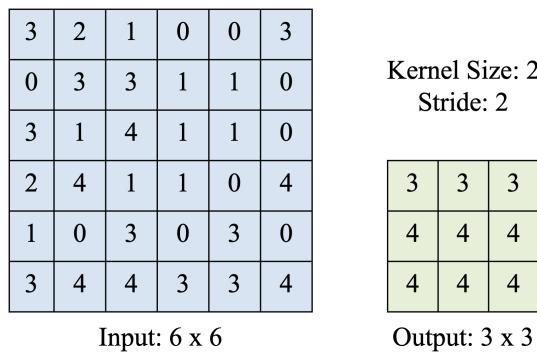
- Convolution layer: gồm Filter, Padding và Stride. Filter là locally connected

network, mỗi filter (kernel) sẽ học được nhiều feature trong images. Ma trận ảnh đầu vào có thể sẽ được padding thêm các giá trị padding index vào ngoài các hàng và các cột. Mỗi filter sẽ di chuyển quanh bức ảnh với bước nhảy được cấu hình trước là Stride.

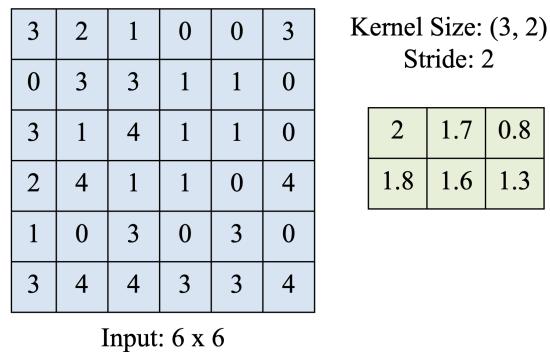


Hình 34.3: Convolutional Operation, Filter, Padding, Stride.

- Pooling layer: bao gồm max-pooling và average-pooling layer tương ứng. Max-pooling layer chọn giá trị lớn nhất từ cửa sổ tính toán, còn Average-pooling layer sẽ tính giá trị trung bình của các giá trị mỗi kernel.

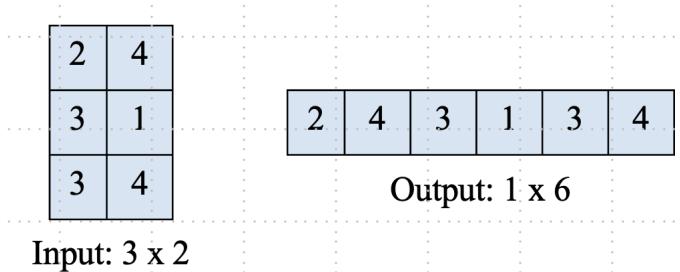


Hình 34.4: Max Pooling.



Hình 34.5: Average Pooling.

- Fully-connected layer: flatten convolution layer cuối cùng duỗi các ma trận nhiều chiều trong ảnh và fully connect neuron cho output layer.

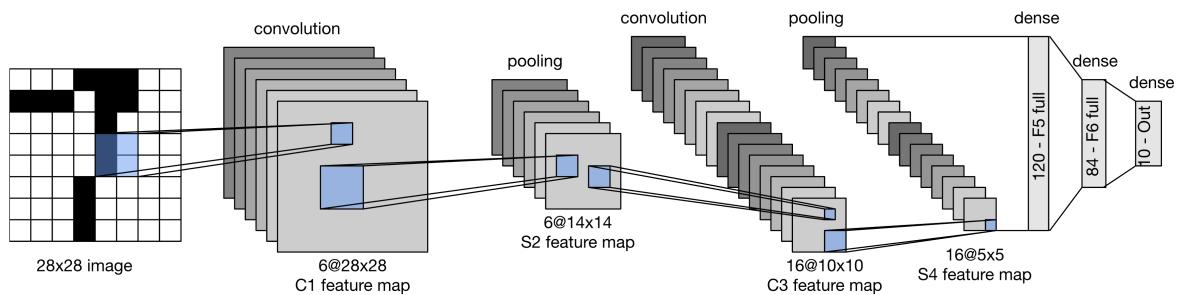


Hình 34.6: Flatten Layer.

34.2 Thực hành

2.1. Phân loại hình ảnh

Phần này chúng ta xây dựng mô hình phân loại cho dữ liệu ảnh và văn bản sử dụng mạng CNN. Một trong những mô hình mạng CNN đầu tiên và nổi bật đó là NeNet được mô tả như hình sau:



Hình 34.7: LeNet Model.

2.1.1. Bộ dữ liệu MNIST

Đầu tiên, mô hình phân loại văn bản sử dụng LeNet được xây dựng trên tập dữ liệu phân loại chữ số MNIST, với 70.000 ảnh và nhãn tương ứng thuộc 10 classes từ 0 đến 9.

1. Tải về bộ dữ liệu

```

1 # Import libs
2 import os
3 import random
4 import numpy as np
5
6 import torch
7 import torch.nn as nn
8 import torch.optim as optim
9 import torch.nn.functional as F
10 import torch.utils.data as data
11
12 import torchvision.transforms as transforms
13 import torchvision.datasets as datasets
14

```

```
15 from torchsummary import summary
16
17 import matplotlib.pyplot as plt
18 from PIL import Image
19
20 ROOT = './data'
21 train_data = datasets.MNIST(
22     root=ROOT,
23     train=True,
24     download=True
25 )
26 test_data = datasets.MNIST(
27     root=ROOT,
28     train=False,
29     download=True
30 )
```

2. Tiên xử lý dữ liệu

- Chia tỉ lệ các tập training: validation = 0.9 : 0.1
- Chuẩn hoá dữ liệu và chuyển sang tensor sử dụng torchvision.transform

```
1 # split training: validation = 0.9 : 0.1
2 VALID_RATIO = 0.9
3
4 n_train_examples = int(len(train_data) * VALID_RATIO)
5 n_valid_examples = len(train_data) - n_train_examples
6
7 train_data, valid_data = data.random_split(
8     train_data,
9     [n_train_examples, n_valid_examples]
10 )
11
12 # compute mean and std for normalization
13 mean = train_data.dataset.data.float().mean() / 255
14 std = train_data.dataset.data.float().std() / 255
15
16 train_transforms = transforms.Compose([
17     transforms.ToTensor(),
18     transforms.Normalize(mean=[mean], std=[std])
19 ])
20 test_transforms = transforms.Compose([
21     transforms.ToTensor(),
22     transforms.Normalize(mean=[mean], std=[std])
23 ])
24
```

```
25 train_data.dataset.transform = train_transforms
26 valid_data.dataset.transform = test_transforms
27
28 # Create dataloader
29
30 BATCH_SIZE = 256
31
32 train_dataloader = data.DataLoader(
33     train_data,
34     shuffle=True,
35     batch_size=BATCH_SIZE
36 )
37
38 valid_dataloader = data.DataLoader(
39     valid_data,
40     batch_size=BATCH_SIZE
41 )
```

3. Xây dựng mô hình LeNet

```
1 class LeNetClassifier(nn.Module):
2     def __init__(self, num_classes):
3         super().__init__()
4         self.conv1 = nn.Conv2d(
5             in_channels=1, out_channels=6, kernel_size=5,
6             padding='same',
7             )
8         self.avgpool1 = nn.AvgPool2d(kernel_size=2)
9         self.conv2 = nn.Conv2d(in_channels=6, out_channels
10 =16, kernel_size=5)
11         self.avgpool2 = nn.AvgPool2d(kernel_size=2)
12         self.flatten = nn.Flatten()
13         self.fc_1 = nn.Linear(16 * 5 * 5, 120)
14         self.fc_2 = nn.Linear(120, 84)
15         self.fc_3 = nn.Linear(84, num_classes)
16
17     def forward(self, inputs):
18         outputs = self.conv1(inputs)
19         outputs = self.avgpool1(outputs)
20         outputs = F.relu(outputs)
21         outputs = self.conv2(outputs)
22         outputs = self.avgpool2(outputs)
23         outputs = F.relu(outputs)
24         outputs = self.flatten(outputs)
25         outputs = self.fc_1(outputs)
26         outputs = self.fc_2(outputs)
```

```
25     outputs = self.fc_3(outputs)
26     return outputs
```

4. Huấn luyện mô hình

```
1 # Training function
2 def train(model, optimizer, criterion, train_dataloader,
3           device, epoch=0, log_interval=50):
4     model.train()
5     total_acc, total_count = 0, 0
6     losses = []
7     start_time = time.time()
8
9     for idx, (inputs, labels) in enumerate(train_dataloader):
10        inputs = inputs.to(device)
11        labels = labels.to(device)
12
13        optimizer.zero_grad()
14
15        predictions = model(inputs)
16
17        # compute loss
18        loss = criterion(predictions, labels)
19        losses.append(loss.item())
20
21        # backward
22        loss.backward()
23        torch.nn.utils.clip_grad_norm_(model.parameters(),
24                                       0.1)
25        optimizer.step()
26        total_acc += (predictions.argmax(1) == labels).sum().item()
27        total_count += labels.size(0)
28
29        if idx % log_interval == 0 and idx > 0:
30            elapsed = time.time() - start_time
31            print(
32                "| epoch {:3d} | {:5d}/{:5d} batches "
33                "| accuracy {:.8.3f}{}".format(
34                    epoch, idx, len(train_dataloader),
35                    total_acc / total_count
36                )
37            )
38            total_acc, total_count = 0, 0
39            start_time = time.time()
40
41    epoch_acc = total_acc / total_count
```

```

38     epoch_loss = sum(losses) / len(losses)
39     return epoch_acc, epoch_loss
40
41 # Evaluation function
42 def evaluate(model, criterion, valid_dataloader):
43     model.eval()
44     total_acc, total_count = 0, 0
45     losses = []
46
47     with torch.no_grad():
48         for idx, (inputs, labels) in enumerate(
49             valid_dataloader):
50             inputs = inputs.to(device)
51             labels = labels.to(device)
52
53             predictions = model(inputs)
54
55             loss = criterion(predictions, labels)
56             losses.append(loss.item())
57
58             total_acc += (predictions.argmax(1) == labels).
59             sum().item()
60             total_count += labels.size(0)
61
62     epoch_acc = total_acc / total_count
63     epoch_loss = sum(losses) / len(losses)
64     return epoch_acc, epoch_loss

```

- Training:

```

1 num_classes = len(train_data.dataset.classes)
2
3 device = torch.device('cuda' if torch.cuda.is_available()
4                       else 'cpu')
5
6 lenet_model = LeNetClassifier(num_classes)
7 lenet_model.to(device)
8
9 criterion = torch.nn.CrossEntropyLoss()
10
11 optimizer = optim.Adam(lenet_model.parameters())
12
13 num_epochs = 10
14 save_model = './model'
15
16 train_accs, train_losses = [], []
17 eval_accs, eval_losses = [], []

```

```

16 best_loss_eval = 100
17
18 for epoch in range(1, num_epochs+1):
19     epoch_start_time = time.time()
20     # Training
21     train_acc, train_loss = train(lenet_model, optimizer,
22         criterion, train_dataloader, device, epoch)
23     train_accs.append(train_acc)
24     train_losses.append(train_loss)
25
26     # Evaluation
27     eval_acc, eval_loss = evaluate(lenet_model, criterion,
28         valid_dataloader)
29     eval_accs.append(eval_acc)
30     eval_losses.append(eval_loss)
31
32     # Save best model
33     if eval_loss < best_loss_eval:
34         torch.save(lenet_model.state_dict(), save_model + '/lenet_model.pt')
35
36     # Print loss, acc end epoch
37     print("-" * 59)
38     print(
39         "| End of epoch {:3d} | Time: {:.2f}s | Train Accuracy {:.3f} | Train Loss {:.3f} "
40         "| Valid Accuracy {:.3f} | Valid Loss {:.3f} ".
41         format(
42             epoch, time.time() - epoch_start_time, train_acc,
43             train_loss, eval_acc, eval_loss
44         )
45     )
46     print("-" * 59)
47
48     # Load best model
49     lenet_model.load_state_dict(torch.load(save_model + '/lenet_model.pt'))
50     lenet_model.eval()

```

- Accuracy và Loss:

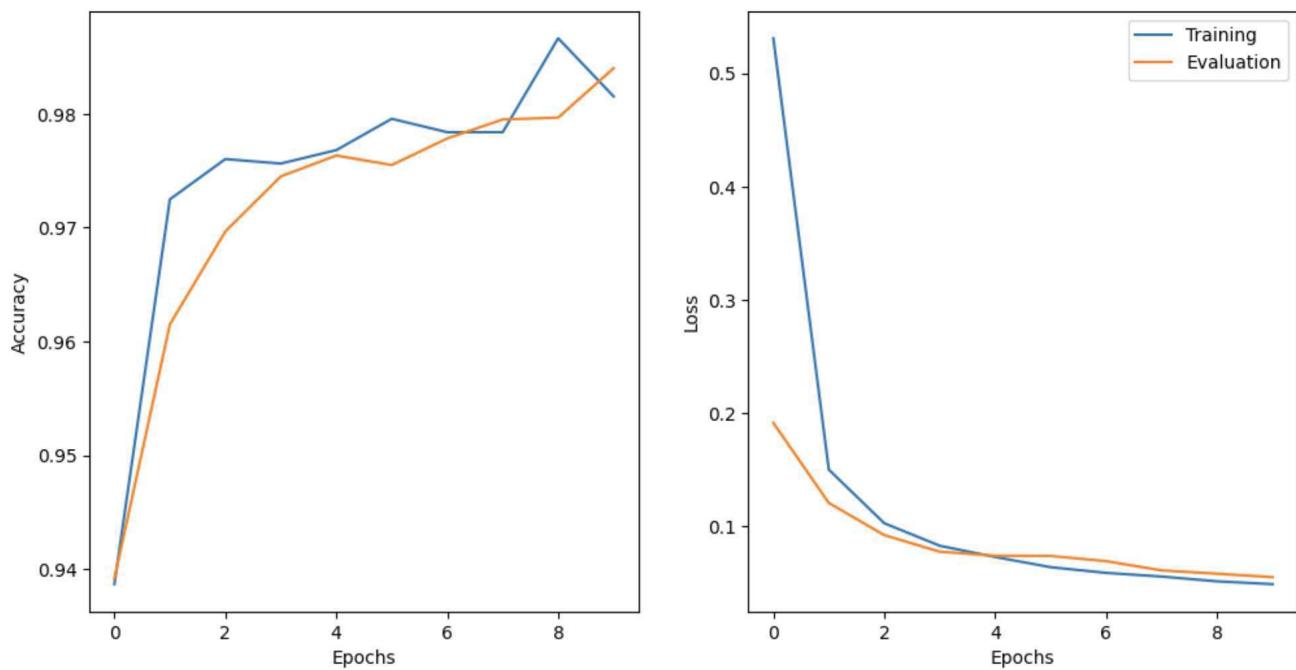
5. Đánh giá mô hình trên tập test

- Đánh giá độ chính xác trên tập test

```

1 test_data.transform = test_transforms
2 test_dataloader = data.DataLoader(

```



Hình 34.8: Accracy và Loss trên tập training và validation.

```

3     test_data,
4     batch_size=BATCH_SIZE
5 )
6 test_acc, test_loss = evaluate(lenet_model, criterion,
    test_dataloader)
7 test_acc, test_loss

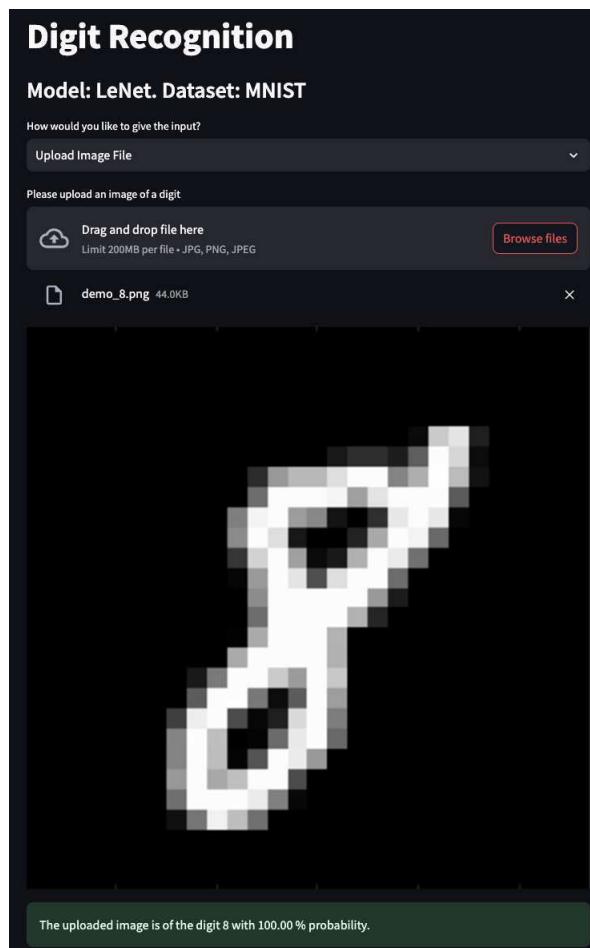
```

- Kết quả độ chính xác trên tập test là 98

6. Triển khai mô hình

Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:

1. [Link Streamlit](#)
2. [Github](#)
3. Giao diện:



Hình 34.9: Giao diện ứng dụng.

2.1.2. Bộ dữ liệu Cassava Leaf Disease

Trong phần này chúng ta xây dựng mô hình phân loại cho bộ dữ liệu Cassava Leaf Disease với các ảnh được gán nhãn và phân loại vào 5 lớp.

1. Tải về bộ dữ liệu

```
1 !wget --no-check-certificate https://storage.googleapis.com/
      emcassavadata/cassavaleafdata.zip \
2           -O /content/cassavaleafdata.zip
3 !unzip /content/cassavaleafdata.zip
4
```



Hình 34.10: Bộ dữ liệu Cassava Leaf Disease.

```
5 # Import libs
6 import os
7 import random
8 import numpy as np
9
10 import torch
```

```
11 import torch.nn as nn
12 import torch.optim as optim
13 import torch.nn.functional as F
14 import torch.utils.data as data
15
16 import torchvision.transforms as transforms
17 import torchvision.datasets as datasets
18
19 from torchsummary import summary
20
21 import matplotlib.pyplot as plt
22 from PIL import Image
```

2. Tiết xử lý dữ liệu

```
1 data_paths = {
2     'train': './train',
3     'valid': './validation',
4     'test': './test'
5 }
6
7 # load image from path
8 def loader(path):
9     return Image.open(path)
10
11 img_size = 150
12 train_transforms = transforms.Compose([
13     transforms.Resize((150, 150)),
14     transforms.ToTensor(),
15 ])
16
17 train_data = datasets.ImageFolder(
18     root=data_paths['train'],
19     loader=loader,
20     transform=train_transforms
21 )
22 valid_data = datasets.ImageFolder(
23     root=data_paths['valid'],
24     transform=train_transforms
25 )
26 test_data = datasets.ImageFolder(
27     root=data_paths['test'],
28     transform=train_transforms
29 )
```

3. Xây dựng mô hình

```
1  class LeNetClassifier(nn.Module):
2      def __init__(self, num_classes):
3          super().__init__()
4          self.conv1 = nn.Conv2d(
5              in_channels=3, out_channels=6, kernel_size=5,
6              padding='same',
7              )
8          self.avgpool1 = nn.AvgPool2d(kernel_size=2)
9          self.conv2 = nn.Conv2d(in_channels=6, out_channels
10             =16, kernel_size=5)
11          self.avgpool2 = nn.AvgPool2d(kernel_size=2)
12          self.flatten = nn.Flatten()
13          self.fc_1 = nn.Linear(16 * 35 * 35, 120)
14          self.fc_2 = nn.Linear(120, 84)
15          self.fc_3 = nn.Linear(84, num_classes)
16
17      def forward(self, inputs):
18          outputs = self.conv1(inputs)
19          outputs = self.avgpool1(outputs)
20          outputs = F.relu(outputs)
21          outputs = self.conv2(outputs)
22          outputs = self.avgpool2(outputs)
23          outputs = F.relu(outputs)
24          outputs = self.flatten(outputs)
25          outputs = self.fc_1(outputs)
26          outputs = self.fc_2(outputs)
27          outputs = self.fc_3(outputs)
28
29      return outputs
```

4. Huấn luyện mô hình

```
1 # Training function
2 def train(model, optimizer, criterion, train_dataloader,
3           device, epoch=0, log_interval=50):
4     model.train()
5     total_acc, total_count = 0, 0
6     losses = []
7     start_time = time.time()
8
9     for idx, (inputs, labels) in enumerate(train_dataloader):
10         inputs = inputs.to(device)
11         labels = labels.to(device)
12
13         optimizer.zero_grad()
```

```
14     predictions = model(inputs)
15
16     # compute loss
17     loss = criterion(predictions, labels)
18     losses.append(loss.item())
19
20     # backward
21     loss.backward()
22     torch.nn.utils.clip_grad_norm_(model.parameters(),
23         0.1)
24     optimizer.step()
25     total_acc += (predictions.argmax(1) == labels).sum().item()
26     total_count += labels.size(0)
27     if idx % log_interval == 0 and idx > 0:
28         elapsed = time.time() - start_time
29         print(
30             "| epoch {:3d} | {:5d}/{:5d} batches "
31             "| accuracy {:.8.3f}".format(
32                 epoch, idx, len(train_dataloader),
33                 total_acc / total_count
34             )
35         )
36         total_acc, total_count = 0, 0
37         start_time = time.time()
38
39     epoch_acc = total_acc / total_count
40     epoch_loss = sum(losses) / len(losses)
41     return epoch_acc, epoch_loss
42
43 # Evaluation function
44 def evaluate(model, criterion, valid_dataloader):
45     model.eval()
46     total_acc, total_count = 0, 0
47     losses = []
48
49     with torch.no_grad():
50         for idx, (inputs, labels) in enumerate(
51             valid_dataloader):
52             inputs = inputs.to(device)
53             labels = labels.to(device)
54             predictions = model(inputs)
55             loss = criterion(predictions, labels)
```

```
55         losses.append(loss.item())
56
57         total_acc += (predictions.argmax(1) == labels).
58             sum().item()
59         total_count += labels.size(0)
60
61     epoch_acc = total_acc / total_count
62     epoch_loss = sum(losses) / len(losses)
63     return epoch_acc, epoch_loss
```

- Huấn luyện mô hình:

```
1 num_classes = len(train_data.classes)
2
3 device = torch.device('cuda' if torch.cuda.is_available()
4                         else 'cpu')
5
6 lenet_model = LeNetClassifier(num_classes)
7 lenet_model.to(device)
8
9 criterion = torch.nn.CrossEntropyLoss()
10 learning_rate = 2e-4
11 optimizer = optim.Adam(lenet_model.parameters(),
12                         learning_rate)
13
14 num_epochs = 10
15 save_model = './model'
16
17 best_loss_eval = 100
18
19 for epoch in range(1, num_epochs+1):
20     epoch_start_time = time.time()
21     # Training
22     train_acc, train_loss = train(lenet_model, optimizer,
23                                   criterion, train_dataloader, device, epoch, log_interval
24                                   =10)
25     train_accs.append(train_acc)
26     train_losses.append(train_loss)
27
28     # Evaluation
29     eval_acc, eval_loss = evaluate(lenet_model, criterion,
30                                    valid_dataloader)
31     eval_accs.append(eval_acc)
32     eval_losses.append(eval_loss)
```

```
30
31     # Save best model
32     if eval_loss < best_loss_eval:
33         torch.save(lenet_model.state_dict(), save_model + '/
lenet_model.pt')
34
35     # Print loss, acc end epoch
36     print("-" * 59)
37     print(
38         "| End of epoch {:3d} | Time: {:.5f}s | Train
Accuracy {:.3f} | Train Loss {:.3f} "
39         "| Valid Accuracy {:.3f} | Valid Loss {:.3f} ".
format(
40             epoch, time.time() - epoch_start_time, train_acc,
41             train_loss, eval_acc, eval_loss
42         )
43     )
44     print("-" * 59)
45
46     # Load best model
47     lenet_model.load_state_dict(torch.load(save_model + '/
lenet_model.pt'))
48     lenet_model.eval()
```

5. Đánh giá mô hình

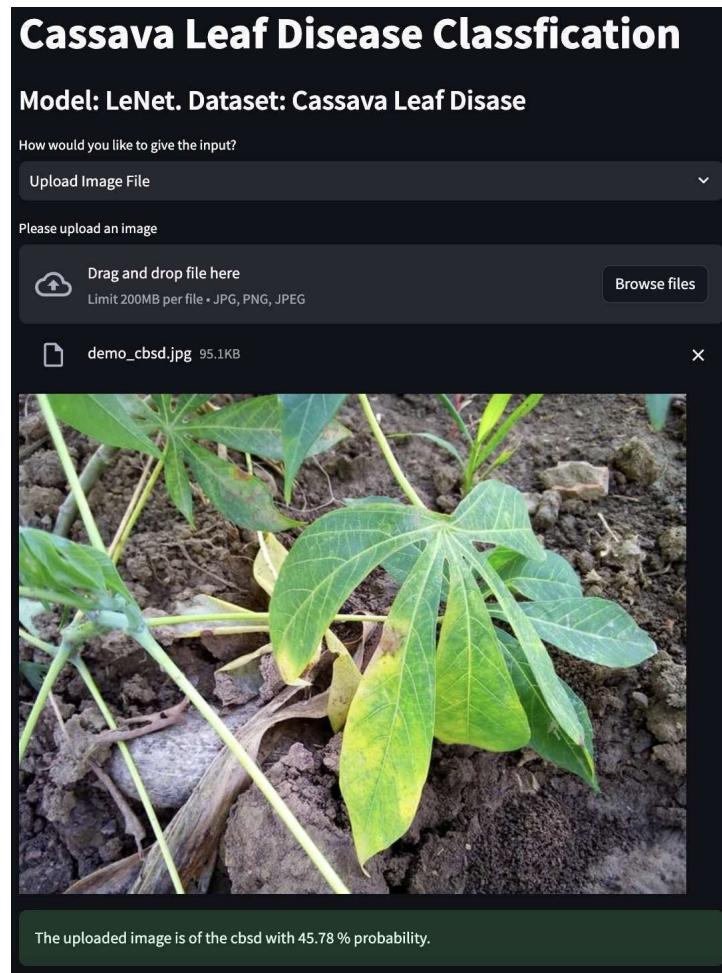
- Kết quả trên tập test: 56

```
1 test_acc, test_loss = evaluate(lenet_model, criterion,
        test_dataloader)
2 test_acc, test_loss
```

6. Triển khai mô hình

Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:

1. [Link Streamlit](#)
2. [Github](#)
3. Giao diện:



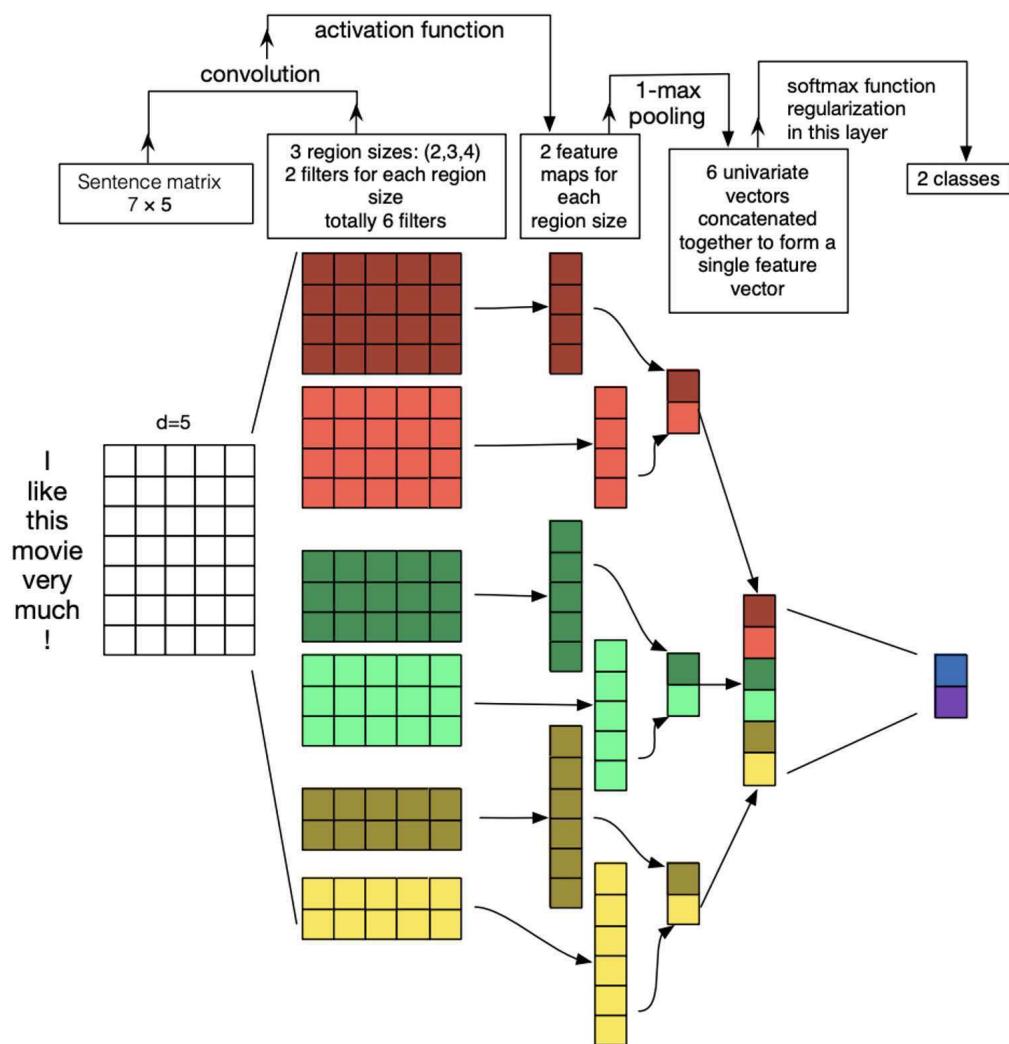
Hình 34.11: Giao diện ứng dụng.

2.2. Phân loại văn bản

Trong phần này chúng ta xây dựng mô hình phân loại văn bản sử dụng kiến trúc mạng TextCNN trên bộ dữ liệu NTC-SCV:

1. Tải về bộ dữ liệu

```
1 !git clone https://github.com/congnggia0609/ntc-scv.git
2 !unzip ./ntc-scv/data/data_test.zip -d ./data
3 !unzip ./ntc-scv/data/data_train.zip -d ./data
4 !rm -rf ./ntc-scv
```



Hình 34.12: TextCNN Model.

```

5
6 # load data from paths to dataframe
7 import os
8 import pandas as pd
9
10 def load_data_from_path(folder_path):
11     examples = []
12     for label in os.listdir(folder_path):

```

```

13     full_path = os.path.join(folder_path, label)
14     for file_name in os.listdir(full_path):
15         file_path = os.path.join(full_path, file_name)
16         with open(file_path, "r", encoding="utf-8") as f:
17             lines = f.readlines()
18             sentence = " ".join(lines)
19             if label == "neg":
20                 label = 0
21             if label == "pos":
22                 label = 1
23             data = {
24                 'sentence': sentence,
25                 'label': label
26             }
27             examples.append(data)
28     return pd.DataFrame(examples)
29
30 folder_paths = {
31     'train': './data/data_train/train',
32     'valid': './data/data_train/test',
33     'test': './data/data_test/test'
34 }
35
36 train_df = load_data_from_path(folder_paths['train'])
37 valid_df = load_data_from_path(folder_paths['valid'])
38 test_df = load_data_from_path(folder_paths['test'])

```

2. Tiền xử lý dữ liệu

Với bộ dữ liệu NTC-SCV có một số bình luận viết bằng tiếng anh hoặc tiếng pháp và chứa các thẻ HTML, đường dẫn URLs. Tiền xử lý dữ liệu gồm 2 bước:

- Xoá bỏ những bình luận không phải tiếng việt

Sử dụng thư viện langid được mô tả như hình sau:

```

1 # Install library
2 !pip install langid
3
4 from langid.langid import LanguageIdentifier, model
5 def identify_vn(df):
6     identifier = LanguageIdentifier.from_modelstring(
7         model, norm_probs=True)
8     not_vi_idx = set()
9     THRESHOLD = 0.9

```

```
9     for idx, row in df.iterrows():
10         score = identifier.classify(row["sentence"])
11         if score[0] != "vi" or (score[0] == "vi" and
12             score[1] <= THRESHOLD):
13             not_vi_idx.add(idx)
14     vi_df = df[~df.index.isin(not_vi_idx)]
15     not_vi_df = df[df.index.isin(not_vi_idx)]
16     return vi_df, not_vi_df
17 train_df_vn, train_df_other = identify_vn(train_df)
```

- Làm sạch dữ liệu

Các bước tiền làm sạch liệu:

- Xoá bỏ thẻ HTML, đường dẫn URL
 - Xoá bỏ dấu câu, số
 - Xoá bỏ các ký tự đặc biệt, emoticons,....
 - Chuẩn hoá khoảng trắng
 - Chuyển sang viết thường

```
1 import re
2 import string
3
4 def preprocess_text(text):
5
6     url_pattern = re.compile(r'https?://\s+\www\.\s+')
7     text = url_pattern.sub(" ", text)
8
9     html_pattern = re.compile(r'<[^>]+>')
10    text = html_pattern.sub(" ", text)
11
12    replace_chars = list(string.punctuation + string.
13 digits)
14    for char in replace_chars:
15        text = text.replace(char, " ")
16
17    emoji_pattern = re.compile("["
18        u"\U0001F600-\U0001F64F" # emoticons
19        u"\U0001F300-\U0001F5FF" # symbols & pictographs
20        u"\U0001F680-\U0001F6FF" # transport & map
21    symbols
22        u"\U0001F1E0-\U0001F1FF" # flags (iOS)
```

```

21         u"\U0001F1F2-\U0001F1F4" # Macau flag
22         u"\U0001F1E6-\U0001F1FF" # flags
23         u"\U0001F600-\U0001F64F"
24         u"\U00002702-\U000027B0"
25         u"\U000024C2-\U0001F251"
26         u"\U0001f926-\U0001f937"
27         u"\U0001F1F2"
28         u"\U0001F1F4"
29         u"\U0001F620"
30         u"\u200d"
31         u"\u2640-\u2642"
32         "+" , flags=re.UNICODE)
33     text = emoji_pattern.sub(r" ", text)
34
35     text = " ".join(text.split())
36
37     return text.lower()
38
39 train_df_vn['preprocess_sentence'] = [
40     preprocess_text(row['sentence']) for index, row in
41     train_df_vn.iterrows()
42 ]
43 valid_df['preprocess_sentence'] = [
44     preprocess_text(row['sentence']) for index, row in
45     valid_df.iterrows()
46 ]
47 test_df['preprocess_sentence'] = [
48     preprocess_text(row['sentence']) for index, row in
49     test_df.iterrows()
50 ]

```

3. Biểu diễn văn bản thành vector

Để biểu diễn dữ liệu văn bản thành các đặc trưng (vectors), chúng ta sử dụng thư viện torchtext.

```

1 !pip install -q torchtext==0.16.0
2
3 # word-based tokenizer
4 from torchtext.data.utils import get_tokenizer
5 tokenizer = get_tokenizer("basic_english")
6
7 # create iter dataset
8 def yield_tokens(sentences, tokenizer):
9     for sentence in sentences:
10         yield tokenizer(sentence)

```

```
11
12 # build vocabulary
13 from torchtext.vocab import build_vocab_from_iterator
14
15 vocab_size = 10000
16 vocabulary = build_vocab_from_iterator(
17     yield_tokens(train_df_v1['preprocess_sentence']),
18     tokenizer,
19     max_tokens=vocab_size,
20     specials=["<pad>", "<unk>"]
21 )
22 vocabulary.set_default_index(vocabulary["<unk>"])
23
24 # convert iter into torchtext dataset
25 def prepare_dataset(df):
26     for index, row in df.iterrows():
27         sentence = row['preprocess_sentence']
28         encoded_sentence = vocabulary(tokenizer(sentence))
29         label = row['label']
30         yield encoded_sentence, label
31
32 train_dataset = prepare_dataset(train_df_v1)
33 train_dataset = to_map_style_dataset(train_dataset)
34
35 valid_dataset = prepare_dataset(valid_df)
36 valid_dataset = to_map_style_dataset(valid_dataset)
```

```
1 import torch
2 from torch.utils.data import DataLoader
3
4 def collate_batch(batch):
5     # create inputs, offsets, labels for batch
6     encoded_sentences, labels = [], []
7     for encoded_sentence, label in batch:
8         labels.append(label)
9         encoded_sentence = torch.tensor(encoded_sentence,
10                                         dtype=torch.int64)
11         encoded_sentences.append(encoded_sentence)
12
13     labels = torch.tensor(labels, dtype=torch.int64)
14     encoded_sentences = pad_sequence(
15         encoded_sentences,
16         padding_value=vocabulary["<pad>"]
17     )
```

```
18     return encoded_sentences, labels
19
20 batch_size = 128
21 train_dataloader = DataLoader(
22     train_dataset,
23     batch_size=batch_size,
24     shuffle=True,
25     collate_fn=collate_batch
26 )
27 valid_dataloader = DataLoader(
28     valid_dataset,
29     batch_size=batch_size,
30     shuffle=False,
31     collate_fn=collate_batch
32 )
```

4. Xây dựng mô hình TextCNN

```
1 class TextCNN(nn.Module):
2     def __init__(self,
3                  vocab_size, embedding_dim, kernel_sizes, num_filters,
4                  num_classes):
5         super(TextCNN, self).__init__()
6
7         self.vocab_size = vocab_size
8         self.embedding_dim = embedding_dim
9         self.kernel_sizes = kernel_sizes
10        self.num_filters = num_filters
11        self.num_classes = num_classes
12        self.embedding = nn.Embedding(vocab_size,
13                                      embedding_dim, padding_idx=0)
13        self.conv = nn.ModuleList([
14            nn.Conv1d(
15                in_channels=embedding_dim,
16                out_channels=num_filters,
17                kernel_size=k,
18                stride=1
19            ) for k in kernel_sizes])
20        self.fc = nn.Linear(len(kernel_sizes) * num_filters,
21                           num_classes)
21
22     def forward(self, x):
23         batch_size, sequence_length = x.shape
24         x = self.embedding(x.T).transpose(1, 2)
25         x = [F.relu(conv(x)) for conv in self.conv]
```

```
26         x = [F.max_pool1d(c, c.size(-1)).squeeze(dim=-1) for
27         c in x]
28         x = torch.cat(x, dim=1)
29         x = self.fc(x)
30     return x
```

5. Huấn luyện mô hình

```
1 # Training
2 import time
3
4 def train(model, optimizer, criterion, train_dataloader,
5           device, epoch=0, log_interval=50):
6     model.train()
7     total_acc, total_count = 0, 0
8     losses = []
9     start_time = time.time()
10
11     for idx, (inputs, labels) in enumerate(train_dataloader):
12         inputs = inputs.to(device)
13         labels = labels.to(device)
14
15         # zero grad
16         optimizer.zero_grad()
17
18         # predictions
19         predictions = model(inputs)
20
21         # compute loss
22         loss = criterion(predictions, labels)
23         losses.append(loss.item())
24
25         # backward
26         loss.backward()
27         optimizer.step()
28         total_acc += (predictions.argmax(1) == labels).sum().item()
29         total_count += labels.size(0)
30
31         if idx % log_interval == 0 and idx > 0:
32             elapsed = time.time() - start_time
33             print(
34                 "| epoch {:3d} | {:5d}/{:5d} batches "
35                 "| accuracy {:.8.3f}".format(
36                     epoch, idx, len(train_dataloader),
37                     total_acc / total_count
38                 )
39             )
```

```

36             )
37             total_acc, total_count = 0, 0
38             start_time = time.time()
39
40             epoch_acc = total_acc / total_count
41             epoch_loss = sum(losses) / len(losses)
42             return epoch_acc, epoch_loss
43
44 # Evaluation
45 def evaluate(model, criterion, valid_dataloader):
46     model.eval()
47     total_acc, total_count = 0, 0
48     losses = []
49
50     with torch.no_grad():
51         for idx, (inputs, labels) in enumerate(
52             valid_dataloader):
53             inputs = inputs.to(device)
54             labels = labels.to(device)
55             # predictions
56             predictions = model(inputs)
57
58             # compute loss
59             loss = criterion(predictions, labels)
60             losses.append(loss.item())
61
62             total_acc += (predictions.argmax(1) == labels).
63             sum().item()
64             total_count += labels.size(0)
65
66             epoch_acc = total_acc / total_count
67             epoch_loss = sum(losses) / len(losses)
68             return epoch_acc, epoch_loss

```

- Huấn luyện mô hình

```

1 num_class = 2
2 vocab_size = len(vocabulary)
3 embedding_dim = 100
4 device = torch.device("cuda" if torch.cuda.is_available()
5 else "cpu")
6
6 model = TextCNN(
7     vocab_size=vocab_size,
8     embedding_dim=embedding_dim,
9     kernel_sizes=[3, 4, 5],

```

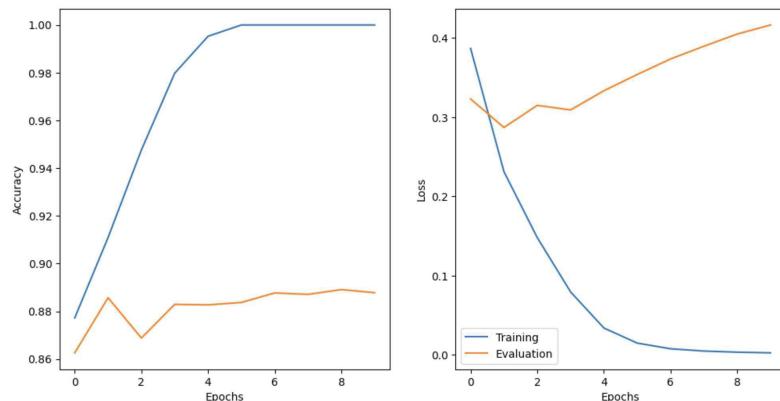
```
10     num_filters=100,
11     num_classes=2
12 )
13 model.to(device)
14
15 criterion = torch.nn.CrossEntropyLoss()
16 optimizer = torch.optim.Adam(model.parameters())
17
18 num_epochs = 10
19 save_model = './model'
20
21 train_accs, train_losses = [], []
22 eval_accs, eval_losses = [], []
23 best_loss_eval = 100
24
25 for epoch in range(1, num_epochs+1):
26     epoch_start_time = time.time()
27     # Training
28     train_acc, train_loss = train(model, optimizer, criterion,
29         , train_dataloader, device, epoch)
30     train_accs.append(train_acc)
31     train_losses.append(train_loss)
32
33     # Evaluation
34     eval_acc, eval_loss = evaluate(model, criterion,
35         valid_dataloader)
36     eval_accs.append(eval_acc)
37     eval_losses.append(eval_loss)
38
39     # Save best model
40     if eval_loss < best_loss_eval:
41         torch.save(model.state_dict(), save_model + '/text_cnn_model.pt')
42
43     # Print loss, acc end epoch
44     print("-" * 59)
45     print(
46         "| End of epoch {:3d} | Time: {:.5f}s | Train Accuracy {:.3f} | Train Loss {:.3f} "
47         "| Valid Accuracy {:.3f} | Valid Loss {:.3f} ".
48         format(
49             epoch, time.time() - epoch_start_time, train_acc,
50             train_loss, eval_acc, eval_loss
51         )
52     )
```

```

49     print("-" * 59)
50
51     # Load best model
52     model.load_state_dict(torch.load(save_model + '/text_cnn_model.pt'))
53     model.eval()

```

- Kết quả huấn luyện mô hình:



Hình 34.13: Accuracy và Loss cho tập training và validation.

6. Đánh giá mô hình

```

1 test_dataset = prepare_dataset(test_df)
2 test_dataset = to_map_style_dataset(test_dataset)
3
4 test_dataloader = DataLoader(
5     test_dataset,
6     batch_size=batch_size,
7     shuffle=False,
8     collate_fn=collate_batch
9 )
10
11 test_acc, test_loss = evaluate(model, criterion,
12     valid_dataloader)
12 test_acc, test_loss

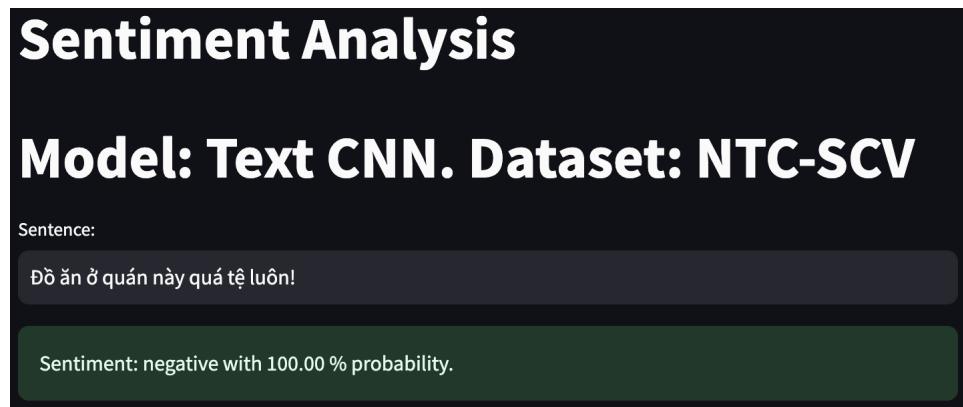
```

- Độ chính xác trên tập test là 88.78

7. Triển khai mô hình

Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:

1. [Link Streamlit](#)
2. [Github](#)
3. Giao diện:



Hình 34.14: Giao diện ứng dụng.

34.3 Câu hỏi trắc nghiệm

Câu hỏi 23 Phép toán sau đây không thuộc lớp tích chập (Convolutional Layer)?

- a) Covolutional Operation
- b) Padding
- c) Stride
- d) Max Pooling

Câu hỏi 24 Lớp Pooling có tác dụng?

- a) Tăng kích thước ảnh
- b) Giảm kích thước ảnh
- c) Cả hai đáp án trên đều đúng
- d) Cả hai đáp án trên đều sai

Trả lời câu hỏi 3 và 4 dựa vào ma trận input và kernel được biểu diễn như hình sau:

Hình 34.15: Phép convolution với kernel size (2, 3).

Câu hỏi 25 Phép tính convolutional với kernel trên cho kết quả có kích thước là?

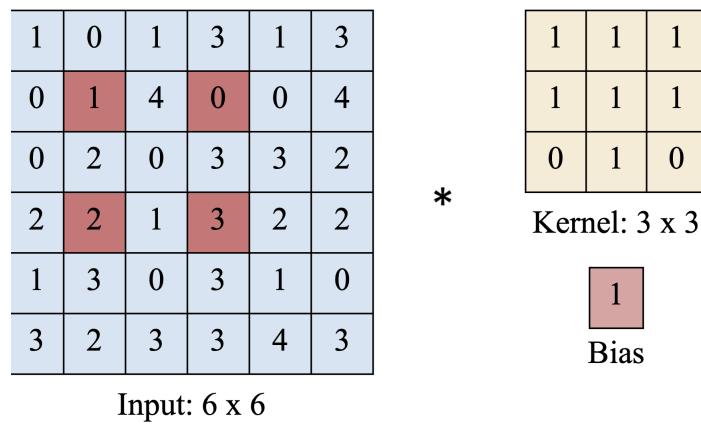
- a) 4 x 3

- b) 5×4
- c) 6×5
- d) 5×6

Câu hỏi 26 Kết quả giá trị cuối cùng theo cả hàng và cột của ví dụ trên (tương ứng là phép tính tại vị trí màu hồng) là?

- a) 4
- b) 8
- c) 16
- d) 32

Câu hỏi 27 Cho ma trận input và kernel size: 3, stride: 2



Hình 34.16: Phép convolution với kernel size: 3, stride: 3.

Kết quả của phép tính trên có kích thước là?

- a) 4×4
- b) 3×3
- c) 2×2
- d) 1×1

Câu hỏi 28 Chọn đáp án đúng cho phép max pooling sau đây?

3	2	1	0	0	3
0	3	3	1	1	0
3	1	4	1	1	0
2	4	1	1	0	4
1	0	3	0	3	0
3	4	4	3	3	4

Kernel Size: 2
Stride: 2

3	3	3
4	5	4
4	4	4

a

3	3	3
4	4	5
4	4	4

b

3	3	3
4	4	4
4	4	4

c

3	3	3
4	4	4
5	4	4

d

Hình 34.17: Phép max pooling với kernel size: 2 và stride: 2.

Câu hỏi 29 Bộ dữ liệu được sử dụng cho phân loại hình ảnh trong bài là?

- a) CIFAR10
- b) CIFAR100
- c) ImageNet
- d) MNIST

Câu hỏi 30 Bộ dữ liệu được sử dụng cho phân loại văn bản được sử dụng trong bài là?

- a) C4
- b) MNIST
- c) NTC-SCV
- d) Tweets

Câu hỏi 31 Mô hình phân loại hình ảnh được sử dụng trong bài là?

- a) LeNet
- b) VGG
- c) GoogleNet
- d) RCNN

Câu hỏi 32 Mô hình phân loại văn bản được sử dụng trong bài là?

- a) RNN
- b) TextCNN
- c) Neural Network
- d) Logistic Regression

34.4 Phụ lục

1. **Hint:** Các file code gợi ý có thể được tải về [tại đây](#)
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

Phần	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Nắm vững kiến thức về bài toán phân loại hình ảnh - Kiến trúc mạng LeNet - Các thành phần quan trọng xây dựng mô hình LeNet như: Lớp tích chập, lớp pooling, lớp linear 	<ul style="list-style-type: none"> - Xây dựng mô hình phân loại hình ảnh sử dụng mạng LeNet trên bộ dữ liệu MNIST và Cassava Leaf Disease
2.	<ul style="list-style-type: none"> - Hiểu về mô hình phân loại văn bản sử dụng TextCNN - Các lớp xử lý cho văn bản như Embedding thông qua thư viện torchtext 	<ul style="list-style-type: none"> - Xây dựng mô hình phân loại văn bản sử dụng TextCNN - Bộ dữ liệu phân loại văn bản NTC-SCV

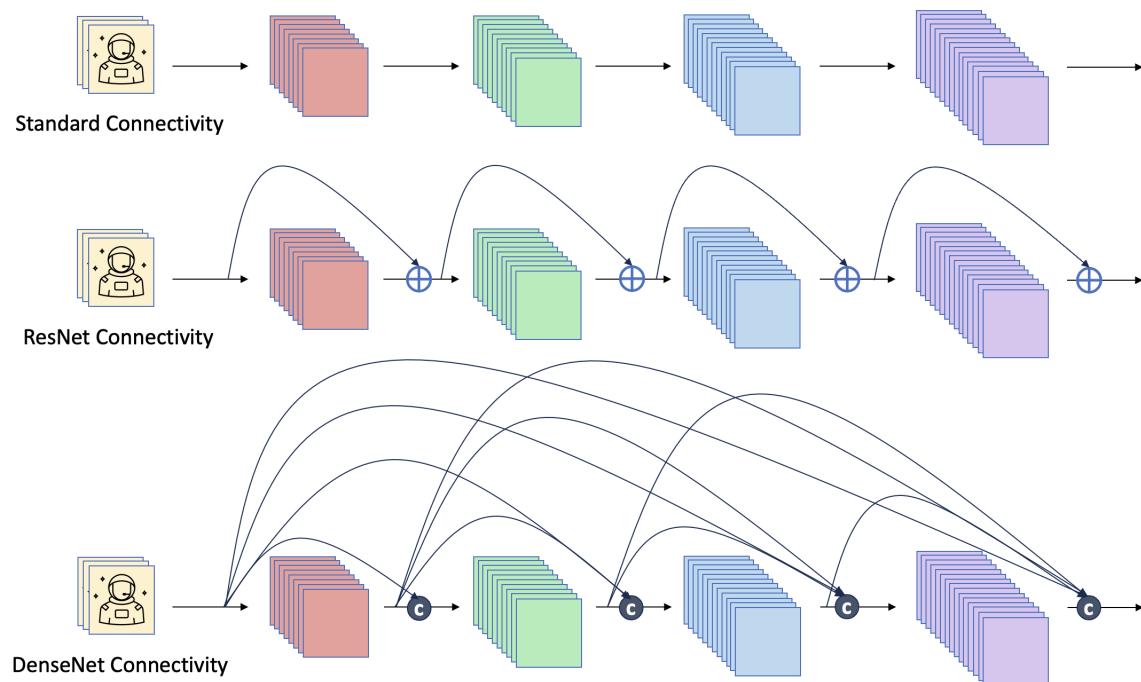
- *Hết* -

Chương 35

Khảo sát và đánh giá tất cả các mô hình CNN nổi bật

35.1 Giới thiệu

Convolutional Neural Networks (CNNs) là một nhánh trong deep neural networks, thường được dùng để giải quyết các bài toán liên quan đến dữ liệu ảnh. Các mạng CNNs với đặc điểm là các lớp tích chập (convolutional layers) để trích xuất đặc trưng từ ảnh, lớp pooling dùng để giảm kích thước ảnh nhằm tăng hiệu suất tính toán và các lớp fully-connected thực hiện phân loại dựa trên các đặc trưng đã trích xuất.



Hình 35.1: Minh họa một số kiểu kết nối giữa các lớp trong CNN.

Trong bài tập này ở phần lập trình, chúng ta sẽ thực hành cài đặt từ đầu quá trình xây dựng hai mô hình phân loại ảnh sử dụng kiến trúc ResNet và DenseNet, áp dụng vào giải quyết hai bài toán phân loại ảnh đa lớp là Weather Image Classification và Scenes Classification. Đồng thời, ôn tập một số lý thuyết về CNNs thông qua bài tập trắc nghiệm.

35.2 Bài tập

35.2.1 Phần lập trình

- Weather Classification

1. **Tải bộ dữ liệu:** Các bạn tải bộ dữ liệu **img_cls_weather_dataset.zip** trong đường dẫn thuộc mục Datasets tại phần 60.4
2. **Import các thư viện cần thiết:** Trong bài tập này, chúng ta sẽ sử dụng thư viện PyTorch để xây dựng và huấn luyện mô hình deep learning. Thêm vào đó, vì làm việc liên quan đến dữ liệu ảnh, chúng ta sẽ sử dụng thư viện PIL để xử lý:

```
1 import torch
2 import torch.nn as nn
3 import os
4 import random
5 import numpy as np
6 import pandas as pd
7 import matplotlib.pyplot as plt
8
9 from PIL import Image
10 from torch.utils.data import Dataset, DataLoader
11 from sklearn.model_selection import train_test_split
```

3. **Cố định giá trị ngẫu nhiên:** Để có thể tái tạo lại cùng một kết quả mô hình, chúng ta sẽ cố định cùng một giá trị ngẫu nhiên (seed) cho các thư viện có chứa các hàm tạo giá trị ngẫu nhiên:

```
1 def set_seed(seed):
2     random.seed(seed)
3     np.random.seed(seed)
4     torch.manual_seed(seed)
5     torch.cuda.manual_seed(seed)
6     torch.cuda.manual_seed_all(seed)
7     torch.backends.cudnn.deterministic = True
8     torch.backends.cudnn.benchmark = False
9
10 seed = 59
11 set_seed(seed)
```

4. **Đọc dữ liệu:** Sau khi tải và giải nén bộ dữ liệu, chúng ta sẽ được một thư mục chứa dữ liệu như sau:



Hình 35.2: Cấu trúc cây thư mục trong bộ dữ liệu ảnh thời tiết.

Để thuận tiện trong việc xây dựng PyTorch datasets, chúng ta sẽ ghi nhận thông tin về các classes, đường dẫn đến tất cả các ảnh cũng như label tương ứng như sau. Nhận thấy tên của các folder con trong thư mục weather-dataset/dataset cũng là tên class. Vì vậy, chúng ta sẽ đọc tên các folder này và đưa vào một dictionary như sau:

```

1 root_dir = 'weather-dataset/dataset'
2 img_paths = []
3 labels = []
4 classes = {
5     label_idx: class_name \
6         for label_idx, class_name in enumerate(
7             sorted(os.listdir(root_dir))
8         )
9 }
  
```

Sau đó, ta đọc toàn bộ đường dẫn của các ảnh trong bộ dữ liệu cũng như label tương ứng:

```

1 img_paths = []
2 labels = []
3 for label_idx, class_name in classes.items():
  
```

```

4     class_dir = os.path.join(root_dir, class_name)
5     for img_filename in os.listdir(class_dir):
6         img_path = os.path.join(class_dir,
7             img_filename)
8         img_paths.append(img_path)
9         labels.append(label_idx)

```

5. **Chia bộ dữ liệu train, val, test:** Với danh sách đường dẫn ảnh và label, chúng ta sẽ chia thành ba bộ dữ liệu train, val, test sử dụng hàm `train_test_split()` của thư viện scikit-learn như sau:

```

1 val_size = 0.2
2 test_size = 0.125
3 is_shuffle = True
4
5 X_train, X_val, y_train, y_val = train_test_split(
6     img_paths, labels,
7     test_size=val_size,
8     random_state=seed,
9     shuffle=is_shuffle
10 )
11
12 X_train, X_test, y_train, y_test = train_test_split(
13     X_train, y_train,
14     test_size=test_size,
15     random_state=seed,
16     shuffle=is_shuffle
17 )

```

6. **Xây dựng class pytorch datasets:** Chúng ta xây dựng class datasets cho bộ dữ liệu weather như sau:

```

1 class WeatherDataset(Dataset):
2     def __init__(
3         self,
4         X, y,
5         transform=None
6     ):
7         self.transform = transform
8         self.img_paths = X
9         self.labels = y
10
11     def __len__(self):
12         return len(self.img_paths)
13
14     def __getitem__(self, idx):

```

```

15         img_path = self.img_paths[idx]
16         img = Image.open(img_path).convert("RGB")
17
18     if self.transform:
19         img = self.transform(img)
20
21     return img, self.labels[idx]

```

7. **Xây dựng hàm tiền xử lý ảnh (transform):** Để đảm bảo dữ liệu ảnh đầu vào được đồng bộ về kích thước và giá trị, chúng ta tự định nghĩa hàm `transform` để tiền xử lý ảnh đầu vào như sau (không sử dụng thư viện `torchvision.transforms`):

```

1 def transform(img, img_size=(224, 224)):
2     img = img.resize(img_size)
3     img = np.array(img) [..., :3]
4     img = torch.tensor(img).permute(2, 0, 1).float()
5     normalized_img = img / 255.0
6
7     return normalized_img

```

Các kỹ thuật được áp dụng: resize ảnh, đổi về tensor và chuẩn hóa giá trị pixel về khoảng (0, 1).

8. **Khai báo datasets object cho ba bộ train, val, test:** Với class `WeatherDataset` và hàm chuẩn hóa ảnh, ta tạo ba object datasets tương ứng như sau:

```

1 train_dataset = WeatherDataset(
2     X_train, y_train,
3     transform=transform
4 )
5 val_dataset = WeatherDataset(
6     X_val, y_val,
7     transform=transform
8 )
9 test_dataset = WeatherDataset(
10    X_test, y_test,
11    transform=transform
12 )

```

9. **Khai báo dataloader:** Với ba object datasets trên, ta khai báo giá trị batch size và tạo dataloader như sau:

```

1 train_batch_size = 512
2 test_batch_size = 8

```

```

3
4 train_loader = DataLoader(
5     train_dataset,
6     batch_size=train_batch_size,
7     shuffle=True
8 )
9 val_loader = DataLoader(
10    val_dataset,
11    batch_size=test_batch_size,
12    shuffle=False
13 )
14 test_loader = DataLoader(
15    test_dataset,
16    batch_size=test_batch_size,
17    shuffle=False
18 )

```

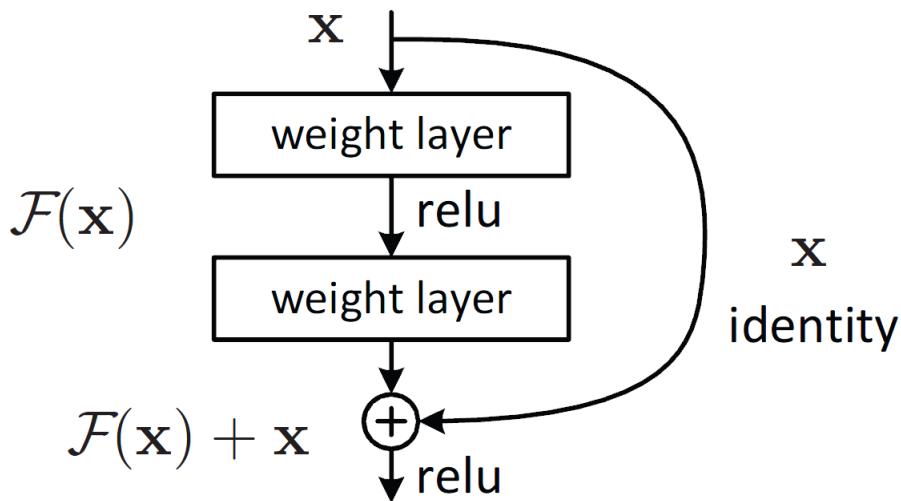
10. **Xây dựng model:** Trong phần này, chúng ta sẽ xây dựng class cho model deep learning với kiến trúc ResNet. Thông tin tổng quan về kiến trúc ResNet được thể hiện ở bảng sau:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112			7×7, 64, stride 2		
conv2_x	56×56	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1			average pool, 1000-d fc, softmax		
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Hình 35.3: Thông tin chi tiết của từng layer trong kiến trúc ResNet. Nguồn ảnh: [link](#).

Đầu tiên, chúng ta sẽ xây dựng class Residual Block, đây là một

thành phần đặc biệt của kiến trúc ResNet so với các mạng CNNs khác, có mô phỏng như hình sau:



Hình 35.4: Minh họa residual connection. Nguồn ảnh: [link](#).

Trong PyTorch, ta triển khai Residual Block như sau:

```

1 class ResidualBlock(nn.Module):
2     def __init__(self, in_channels, out_channels,
3                  stride=1):
4         super(ResidualBlock, self).__init__()
5         self.conv1 = nn.Conv2d(in_channels,
6                             out_channels, kernel_size=3, stride=stride,
7                             padding=1)
8         self.batch_norm1 = nn.BatchNorm2d(
9             out_channels)
10        self.conv2 = nn.Conv2d(out_channels,
11                             out_channels, kernel_size=3, stride=1, padding=1)
12        self.batch_norm2 = nn.BatchNorm2d(
13            out_channels)
14
15        self.downsample = nn.Sequential()
16        if stride != 1 or in_channels != out_channels
17        :
18            self.downsample = nn.Sequential(
19                nn.Conv2d(in_channels, out_channels,
20                         kernel_size=1, stride=stride),
21                nn.BatchNorm2d(out_channels))

```

```

14         )
15         self.relu = nn.ReLU()
16
17     def forward(self, x):
18         shortcut = x.clone()
19         x = self.conv1(x)
20         x = self.batch_norm1(x)
21         x = self.relu(x)
22         x = self.conv2(x)
23         x = self.batch_norm2(x)
24         x += self.downsample(shortcut)
25         x = self.relu(x)
26
27     return x

```

Với ResidualBlock, ta triển khai toàn bộ kiến trúc ResNet như sau:

```

1 class ResNet(nn.Module):
2     def __init__(self, residual_block, n_blocks_lst,
3                  n_classes):
4         super(ResNet, self).__init__()
5         self.conv1 = nn.Conv2d(3, 64, kernel_size=7,
6                           stride=2, padding=3)
7         self.batch_norm1 = nn.BatchNorm2d(64)
8         self.relu = nn.ReLU()
9         self.maxpool = nn.MaxPool2d(kernel_size=3,
10                           stride=2, padding=1)
11        self.conv2 = self.create_layer(residual_block,
12                           64, 64, n_blocks_lst[0], 1)
13        self.conv3 = self.create_layer(residual_block,
14                           64, 128, n_blocks_lst[1], 2)
15        self.conv4 = self.create_layer(residual_block,
16                           128, 256, n_blocks_lst[2], 2)
17        self.conv5 = self.create_layer(residual_block,
18                           256, 512, n_blocks_lst[3], 2)
19        self.avgpool = nn.AdaptiveAvgPool2d(1)
20        self.flatten = nn.Flatten()
21        self.fc1 = nn.Linear(512, n_classes)
22
23    def create_layer(self, residual_block,
24                    in_channels, out_channels, n_blocks, stride):
25        blocks = []
26        first_block = residual_block(in_channels,
27                                     out_channels, stride)
28        blocks.append(first_block)

```

```

20
21     for idx in range(1, n_blocks):
22         block = residual_block(out_channels,
23                               out_channels, stride=1)
24         blocks.append(block)
25
26     block_sequential = nn.Sequential(*blocks)
27
28     return block_sequential
29
30 def forward(self, x):
31     x = self.conv1(x)
32     x = self.batch_norm1(x)
33     x = self.maxpool(x)
34     x = self.relu(x)
35     x = self.conv2(x)
36     x = self.conv3(x)
37     x = self.conv4(x)
38     x = self.conv5(x)
39     x = self.avgpool(x)
40     x = self.flatten(x)
41     x = self.fc1(x)
42
43     return x

```

Cuối cùng, khai báo model ResNet bằng đoạn code sau:

```

1 n_classes = len(list(classes.keys()))
2 device = 'cuda' if torch.cuda.is_available() else 'cpu'
3
4 model = ResNet(
5     residual_block=ResidualBlock,
6     n_blocks_lst=[2, 2, 2, 2],
7     n_classes=n_classes
8 ).to(device)

```

11. Xây dựng hàm đánh giá model: Ta xây dựng hàm đánh giá model với đầu vào là model, bộ dữ liệu đánh giá và hàm loss. Hàm này sẽ trả về giá trị loss và accuracy của model trên tập dữ liệu đầu vào:

```

1 def evaluate(model, dataloader, criterion, device):
2     model.eval()
3     correct = 0

```

```

4     total = 0
5     losses = []
6     with torch.no_grad():
7         for inputs, labels in dataloader:
8             inputs, labels = inputs.to(device),
9             labels.to(device)
10            outputs = model(inputs)
11            loss = criterion(outputs, labels)
12            losses.append(loss.item())
13            _, predicted = torch.max(outputs.data, 1)
14            total += labels.size(0)
15            correct += (predicted == labels).sum().
16            item()
17
18    loss = sum(losses) / len(losses)
19    acc = correct / total
20
21    return loss, acc

```

12. Xây dựng hàm huấn luyện model: Ta triển khai xây dựng hàm huấn luyện mô hình như sau:

```

1 def fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     device,
8     epochs
9 ):
10    train_losses = []
11    val_losses = []
12
13    for epoch in range(epochs):
14        batch_train_losses = []
15
16        model.train()
17        for idx, (inputs, labels) in enumerate(
18            train_loader):
19            inputs, labels = inputs.to(device),
20            labels.to(device)
21
22            optimizer.zero_grad()
23            outputs = model(inputs)
24            loss = criterion(outputs, labels)

```

```

23         loss.backward()
24         optimizer.step()
25
26         batch_train_losses.append(loss.item())
27
28         train_loss = sum(batch_train_losses) / len(
batch_train_losses)
29         train_losses.append(train_loss)
30
31         val_loss, val_acc = evaluate(
32             model, val_loader,
33             criterion, device
34         )
35         val_losses.append(val_loss)
36
37         print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal loss: {val_loss:.4f}')
38
39     return train_losses, val_losses

```

13. **Khai báo hàm loss và thuật toán tối ưu hóa:** Với bài toán phân loại ảnh, ta sử dụng hàm loss CrossEntropyLoss và thuật toán tối ưu hóa Stochastic Gradient Descent (SGD). Ngoài ra, ta cũng khai báo giá trị learning rate và số epochs:

```

1 lr = 1e-2
2 epochs = 25
3
4 criterion = nn.CrossEntropyLoss()
5 optimizer = torch.optim.SGD(
6     model.parameters(),
7     lr=lr
8 )

```

14. **Thực hiện huấn luyện:** Với tất cả các tham số đầu vào đã sẵn sàng, ta gọi hàm fit() để bắt đầu quá trình huấn luyện mô hình ResNet:

```

1 train_losses, val_losses = fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     device,

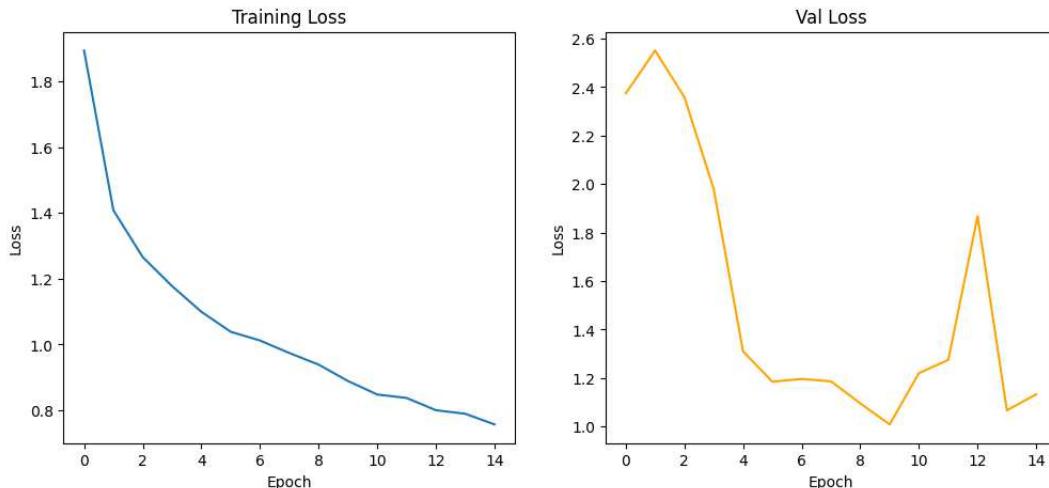
```

```

8     epochs
9 )

```

Với danh sách giá trị loss qua từng epoch trên tập train và val, ta có thể trực quan hóa như sau:



Hình 35.5: Trực quan hóa kết quả huấn luyện của mô hình trên tập train và tập val theo giá trị loss.

15. **Đánh giá mô hình:** Ta gọi hàm `evaluate()` để đánh giá performance của model trên hai tập val và test như sau:

```

1 val_loss, val_acc = evaluate(
2     model,
3     val_loader,
4     criterion,
5     device
6 )
7 test_loss, test_acc = evaluate(
8     model,
9     test_loader,
10    criterion,
11    device
12 )
13 print('Evaluation on val/test dataset')
14 print('Val accuracy: ', val_acc)
15 print('Test accuracy: ', test_acc)

```

- Scenes Classification

1. **Tải bộ dữ liệu:** Các bạn tải bộ dữ liệu **img_cls_scenes_classification.zip** trong đường dẫn thuộc mục Datasets tại phần 60.4

2. **Import các thư viện cần thiết:**

```
1 import torch
2 import torch.nn as nn
3 import os
4 import random
5 import numpy as np
6 import pandas as pd
7 import matplotlib.pyplot as plt
8
9 from PIL import Image
10 from torch.utils.data import Dataset, DataLoader
11 from sklearn.model_selection import train_test_split
```

3. **Cố định giá trị ngẫu nhiên:**

```
1 def set_seed(seed):
2     random.seed(seed)
3     np.random.seed(seed)
4     torch.manual_seed(seed)
5     torch.cuda.manual_seed(seed)
6     torch.cuda.manual_seed_all(seed)
7     torch.backends.cudnn.deterministic = True
8     torch.backends.cudnn.benchmark = False
9
10 seed = 59
11 set_seed(seed)
```

4. **Đọc dữ liệu:** Sau khi tải và giải nén bộ dữ liệu, chúng ta sẽ được một thư mục chứa dữ liệu như sau:



Hình 35.6: Cấu trúc cây thư mục trong bộ dữ liệu ảnh phong cảnh.

So với bộ dữ liệu ở bài trước, ta đã được chia sẵn hai bộ dữ liệu train và val. Tuy nhiên, để đồng bộ, chúng ta sẽ vẫn đi theo hướng code cũ. Đầu tiên, ta vẫn đọc danh sách classes như sau:

```

1 root_dir = 'scenes_classification'
2 train_dir = os.path.join(root_dir, 'train')
3 test_dir = os.path.join(root_dir, 'val')
4
5 classes = {
6     label_idx: class_name \
7         for label_idx, class_name in enumerate(
8             sorted(os.listdir(train_dir))
9         )
10    }
  
```

Tiếp đến, ta đọc lên toàn bộ các đường dẫn ảnh cũng như label tương ứng. Tuy nhiên, ta sẽ coi thư mục val là tập test của bộ dữ liệu và sẽ khai báo danh sách riêng cho bộ này. Như vậy, ta có code như sau:

```

1 X_train = []
2 y_train = []
3 X_test = []
4 y_test = []
  
```

```

5
6 for dataset_path in [train_dir, test_dir]:
7     for label_idx, class_name in classes.items():
8         class_dir = os.path.join(dataset_path,
9             class_name)
10        for img_filename in os.listdir(class_dir):
11            img_path = os.path.join(class_dir,
12                img_filename)
13            if 'train' in dataset_path:
14                X_train.append(img_path)
15                y_train.append(label_idx)
16            else:
17                X_test.append(img_path)
18                y_test.append(label_idx)

```

5. **Chia bộ dữ liệu train, val, test:** Vì đã có sẵn bộ dữ liệu train và test, ta chỉ việc chia thêm cho tập val từ tập train như sau:

```

1 seed = 0
2 val_size = 0.2
3 is_shuffle = True
4
5 X_train, X_val, y_train, y_val = train_test_split(
6     X_train, y_train,
7     test_size=val_size,
8     random_state=seed,
9     shuffle=is_shuffle
10 )

```

6. **Xây dựng class pytorch datasets:** Tương tự ở bài trước, ta xây dựng pytorch dataset cho bộ dữ liệu Scenes như sau:

```

1 class ScenesDataset(Dataset):
2     def __init__(self,
3                  X,
4                  y,
5                  transform=None):
6         self.transform = transform
7         self.img_paths = X
8         self.labels = y
9
10    def __len__(self):
11        return len(self.img_paths)
12
13    def __getitem__(self, idx):

```

```

15         img_path = self.img_paths[idx]
16         img = Image.open(img_path).convert("RGB")
17
18     if self.transform:
19         img = self.transform(img)
20
21     return img, self.labels[idx]

```

7. **Xây dựng hàm tiền xử lý ảnh (transforms):** Tương tự như bài trước, ta xây dựng hàm tiền xử lý ảnh như sau:

```

1 def transform(img, img_size=(224, 224)):
2     img = img.resize(img_size)
3     img = np.array(img)[..., :3]
4     img = torch.tensor(img).permute(2, 0, 1).float()
5     normalized_img = img / 255.0
6
7     return normalized_img

```

8. **Khai báo datasets object cho ba bộ train, val, test:** Tương tự như bài trên, ta khai báo ba object datasets tương ứng cho ba bộ dữ liệu train, val, test như sau:

```

1 train_dataset = ScenesDataset(
2     X_train, y_train,
3     transform=transform
4 )
5 val_dataset = ScenesDataset(
6     X_val, y_val,
7     transform=transform
8 )
9 test_dataset = ScenesDataset(
10    X_test, y_test,
11    transform=transform
12 )

```

9. **Khai báo dataloader:** Lưu ý, với colab, các bạn nên cài đặt train batch size = 64 để có thể train được bài này.

```

1 train_batch_size = 64
2 test_batch_size = 8
3
4 train_loader = DataLoader(
5     train_dataset,
6     batch_size=train_batch_size,
7     shuffle=True

```

```

8 )
9 val_loader = DataLoader(
10     val_dataset,
11     batch_size=test_batch_size,
12     shuffle=False
13 )
14 test_loader = DataLoader(
15     test_dataset,
16     batch_size=test_batch_size,
17     shuffle=False
18 )

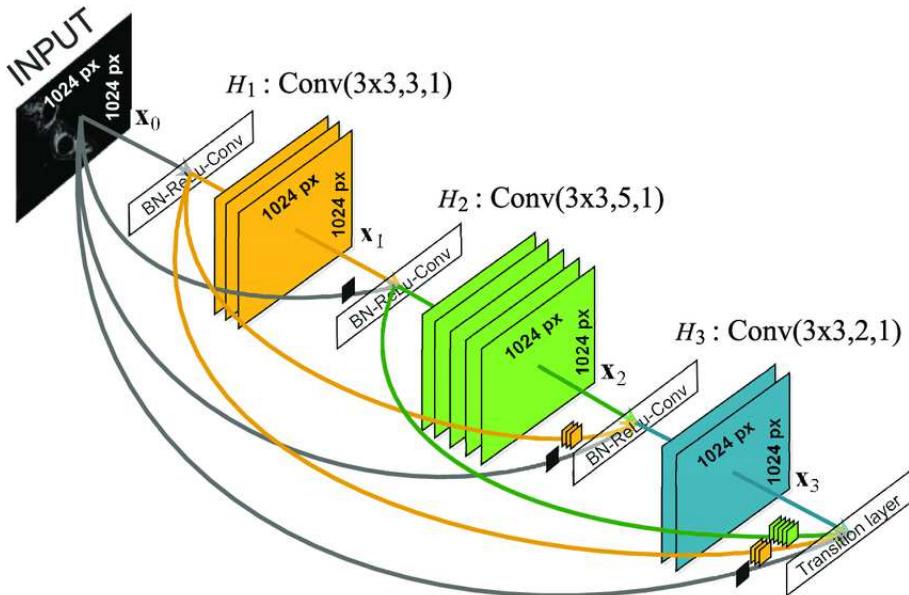
```

10. **Xây dựng model:** Trong phần này, chúng ta sẽ xây dựng class cho model deep learning với kiến trúc DenseNet. Thông tin tổng quan về kiến trúc DenseNet được thể hiện ở bảng sau:

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112		7×7 conv, stride 2		
Pooling	56×56		3×3 max pool, stride 2		
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56			1×1 conv	
	28×28			2×2 average pool, stride 2	
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28			1×1 conv	
	14×14			2×2 average pool, stride 2	
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14			1×1 conv	
	7×7			2×2 average pool, stride 2	
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1		7×7 global average pool		1000D fully-connected, softmax

Hình 35.7: Thông tin chi tiết của từng layer trong kiến trúc DenseNet. Nguồn ảnh: [link](#).

Đầu tiên, chúng ta sẽ xây dựng class Dense Block, có ảnh mô phỏng như hình dưới đây:



Hình 35.8: Ảnh mô phỏng kiến trúc mạng DenseNet. Nguồn ảnh: [link](#).

```

1  class BottleneckBlock(nn.Module):
2      def __init__(self, in_channels, growth_rate):
3          super(BottleneckBlock, self).__init__()
4          self.bn1 = nn.BatchNorm2d(in_channels)
5          self.conv1 = nn.Conv2d(in_channels, 4 * growth_rate, kernel_size=1, bias=False)
6          self.bn2 = nn.BatchNorm2d(4 * growth_rate)
7          self.conv2 = nn.Conv2d(4 * growth_rate, growth_rate, kernel_size=3, padding=1, bias=False)
8          self.relu = nn.ReLU()
9
10     def forward(self, x):
11         res = x.clone().detach()
12         x = self.bn1(x)
13         x = self.relu(x)
14         x = self.conv1(x)
15         x = self.bn2(x)
16         x = self.relu(x)
17         x = self.conv2(x)
18         x = torch.cat([res, x], 1)
19
20     return x

```

```

21
22 class DenseBlock(nn.Module):
23     def __init__(self, num_layers, in_channels,
24                  growth_rate):
25         super(DenseBlock, self).__init__()
26         layers = []
27         for i in range(num_layers):
28             layers.append(BottleneckBlock(in_channels
29                                         + i * growth_rate, growth_rate))
30         self.block = nn.Sequential(*layers)
31
32     def forward(self, x):
33         return self.block(x)

```

Với DenseBlock, ta triển khai toàn bộ kiến trúc DenseNet như sau:

```

1  class DenseNet(nn.Module):
2      def __init__(self, num_blocks, growth_rate,
3                   num_classes):
4          super(DenseNet, self).__init__()
5          self.conv1 = nn.Conv2d(3, 2 * growth_rate,
6                               kernel_size=7, padding=3, stride=2, bias=False)
7          self.bn1 = nn.BatchNorm2d(2 * growth_rate)
8          self.pool1 = nn.MaxPool2d(kernel_size=3,
9                                   stride=2, padding=1)
10
11         self.dense_blocks = nn.ModuleList()
12         in_channels = 2 * growth_rate
13         for i, num_layers in enumerate(num_blocks):
14             self.dense_blocks.append(DenseBlock(
15                 num_layers, in_channels, growth_rate))
16             in_channels += num_layers * growth_rate
17             if i != len(num_blocks) - 1:
18                 out_channels = in_channels // 2
19                 self.dense_blocks.append(nn.
20                     Sequential(
21                         nn.BatchNorm2d(in_channels),
22                         nn.Conv2d(in_channels,
23                                 out_channels, kernel_size=1, bias=False),
24                         nn.AvgPool2d(kernel_size=2,
25                                     stride=2)
26                     ))
27                 in_channels = out_channels
28
29         self.bn2 = nn.BatchNorm2d(in_channels)

```

```

23         self.pool2 = nn.AvgPool2d(kernel_size=7)
24         self.relu = nn.ReLU()
25         self.fc = nn.Linear(in_channels, num_classes)
26
27     def forward(self, x):
28         x = self.conv1(x)
29         x = self.bn1(x)
30         x = self.relu(x)
31         x = self.pool1(x)
32
33         for block in self.dense_blocks:
34             x = block(x)
35
36         x = self.bn2(x)
37         x = self.relu(x)
38         x = self.pool2(x)
39         x = x.view(x.size(0), -1)
40         x = self.fc(x)
41
42     return x

```

Cuối cùng, khai báo model DenseNet bằng đoạn code sau (ở đây ta sẽ sử dụng phiên bản DenseNet-121):

```

1 n_classes = len(list(classes.keys()))
2 device = 'cuda' if torch.cuda.is_available() else 'cpu'
3
4 model = DenseNet(
5     [6, 12, 24, 16],
6     growth_rate=32,
7     num_classes=n_classes
8 ).to(device)

```

11. Khai báo hàm loss và thuật toán huấn luyện: Với bài toán là phân loại ảnh, ta cũng sẽ sử dụng hàm loss CrossEntropy và Stochastic Gradient:

```

1 lr = 1e-2
2 epochs = 15
3
4 criterion = nn.CrossEntropyLoss()
5 optimizer = torch.optim.SGD(
6     model.parameters(),
7     lr=lr
8 )

```

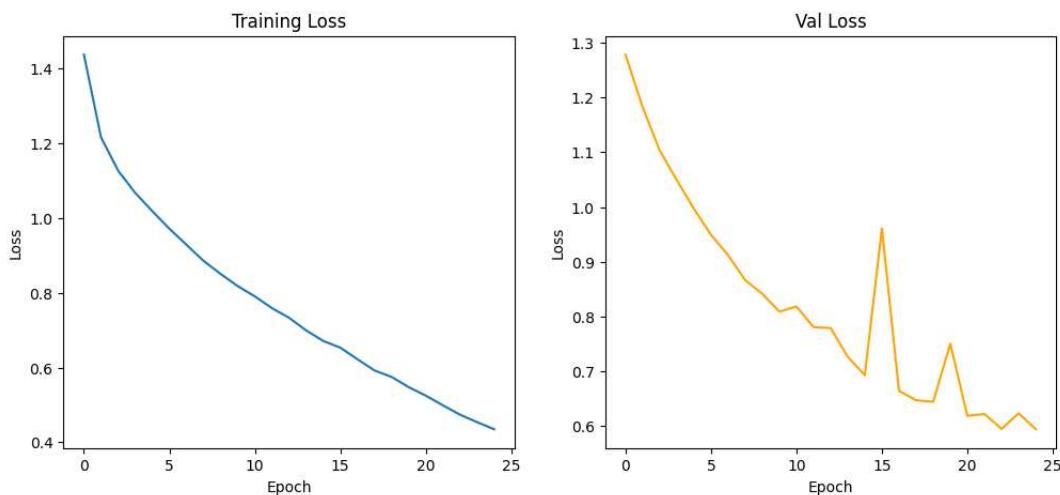
12. **Thực hiện huấn luyện:** Sử dụng hàm evaluate() và hàm fit() đã triển khai trong bài trước, chúng ta sẽ huấn luyện model DenseNet như sau:

```

1 train_losses, val_losses = fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     device,
8     epochs
9 )

```

Với danh sách giá trị loss qua từng epoch trên tập train và val, ta có thể trực quan hóa như sau:



Hình 35.9: Trực quan hóa kết quả huấn luyện của mô hình trên tập train và tập val theo giá trị loss.

13. **Đánh giá model:** Ta gọi hàm evaluate() để đánh giá performance của model trên hai tập val và test như sau:

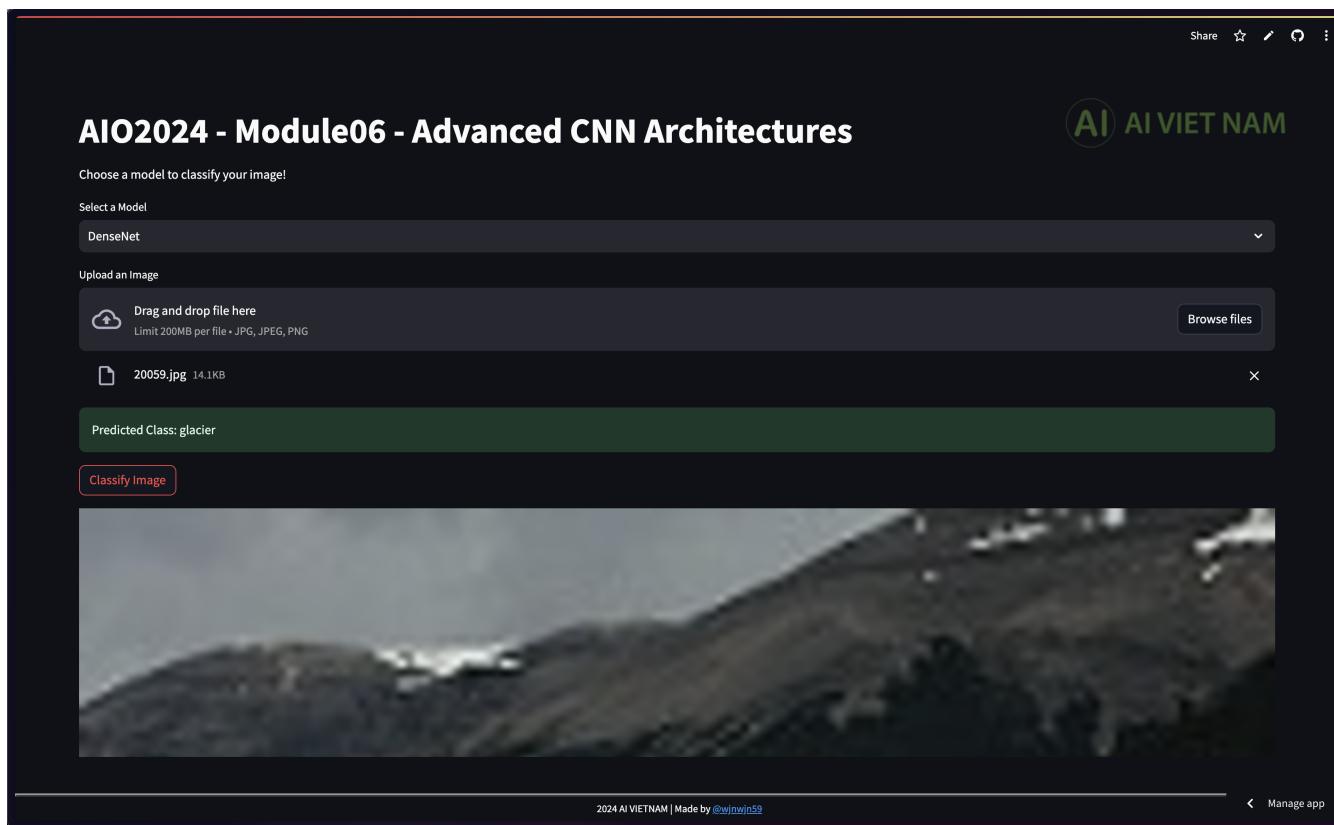
```

1 val_loss, val_acc = evaluate(
2     model,
3     val_loader,
4     criterion,

```

```
5     device
6 )
7 test_loss, test_acc = evaluate(
8     model,
9     test_loader,
10    criterion,
11    device
12 )
13
14 print('Evaluation on val/test dataset')
15 print('Val accuracy: ', val_acc)
16 print('Test accuracy: ', test_acc)
```

- **Triển khai mô hình:** Với hai mô hình về phân loại ảnh thời tiết và phân loại ảnh phong cảnh đã huấn luyện được ở phía trên, chúng ta có thể triển khai lên streamlit như ảnh sau (thông tin về code cài đặt và link web streamlit được đề cập tại phần 60.4):



Hình 35.10: Ảnh giao diện của web demo.

35.2.2 Phân trắc nghiệm

1. Cho đoạn code như hình bên dưới, các bạn khai báo sử dụng BatchNormalization của pytorch với các tham số mặc định. Khi thực thi chương trình, kết quả in ra màn hình là?

```

1 import torch
2 import torch.nn as nn
3 import numpy as np
4
5 data = np.array([[1, 6], [3, 4]])
6 data = torch.tensor(data, dtype=torch.float32)
7
8 bnorm = nn.BatchNorm2d(1)
9 data = data.unsqueeze(0)
10 with torch.no_grad():
11     output = bnorm(data)
12     print(output)

```

- a) tensor([[[[-1.3867, 1.3867], [-0.2773, 0.2773]]]]).
- b) tensor([[[[-1.3842, 1.3998], [-0.2573, 0.2417]]]]).
- c) tensor([[[[-1.3802, 1.3906], [-0.2891, 0.2787]]]]).
- d) tensor([[[[-1.3732, 1.4073], [-0.2750, 0.2409]]]]).

2. Cho đoạn code như hình bên dưới, các bạn sử dụng phép toán concatenate (torch.cat) của pytorch với axis=0, để concatenate 2 feature map lại với nhau. Khi thực thi chương trình, kết quả in ra màn hình là?

```

1 import torch
2
3 a = torch.tensor([[1, 2], [3, 4]])
4 b = torch.tensor([[1, 2], [3, 4]])
5
6 a = a.reshape(1, 2, 2)
7 b = b.reshape(1, 2, 2)
8
9
10 c = torch.cat((a, b))
11 print(c)

```

- a) tensor([[5, 6], [7, 8]], [[9, 10], [11, 12]]].
- b) tensor([[13, 14], [15, 16]], [[17, 18], [19, 20]]].

- c) `tensor([[[1, 2], [3, 4]], [[1, 2], [3, 4]]]).`
- d) `tensor([[[21, 22], [23, 24]], [[25, 26], [27, 28]]]).`
3. Hạn chế nào sau đây trong deep neural networks truyền thống đã được ResNet cải thiện?
- a) Vanishing Gradient.
 - b) Underfitting với mạng kích thước nhỏ.
 - c) Overfitting với dữ liệu lớn.
 - d) Chi phí tính toán cao.
4. Khi thực thi đoạn code sau đây, kết quả in ra màn hình là?

```
1 import torch
2 import torch.nn as nn
3
4 seed = 1
5 torch.manual_seed(seed)
6 input_tensor = torch.tensor([[[[1.0, 2.0], [3.0, 4.0]]]])
7
8 conv_layer = nn.Conv2d(in_channels=1, out_channels=1,
9 kernel_size=1)
9 conv_output = conv_layer(input_tensor)
10
11 with torch.no_grad():
12     output = conv_output + input_tensor
13     print(output)
```

- a) `tensor([[[[0.1345, 1.2345], [2.1345, 3.1345]]]]).`
 - b) `tensor([[[[-1.5678, -0.5678], [0.4322, 1.4322]]]]).`
 - c) `tensor([[[[2.5432, 3.5432], [4.5432, 5.5432]]]]).`
 - d) `tensor([[[[1.0739, 2.5891], [4.1044, 5.6197]]]]).`
5. Điều nào sau đây là sự khác biệt giữa ResNet-18 và ResNet-34?
- a) ResNet-34 sử dụng DenseBlock thay vì ResidualBlock.
 - b) ResNet-18 sử dụng loại convolutional layer khác.
 - c) Hàm kích hoạt trong mỗi layer.
 - d) Số lượng layers.

6. Trong ResNet, để thực hiện skip connection add, channel của 2 feature map phải bằng nhau. Khi đó, tác giả paper ResNet đã đề xuất phương pháp gì để thực hiện skip connection khi input và output của một block có số lượng channel khác nhau?
 - a) Zero-padding.
 - b) Convolution layer 1x1.
 - c) Tất cả đều sai.
 - d) A và B đúng.
7. Đặc điểm nào sau đây là của kiến trúc DenseNet?
 - a) Có sử dụng Residual Connections giữa các layer.
 - b) Sử dụng layer max-pooling giữa các layer.
 - c) Ứng dụng 1x1 convolution.
 - d) Có Direct Connection từ bất kì layer nào đến tất cả các layer đứng sau.
8. Điều nào sau đây là lợi điểm của kiến trúc DenseNet?
 - a) Tiêu tốn nhiều tài nguyên tính toán hơn các kiến trúc CNNs khác.
 - b) Hạn chế tình trạng Vanish Gradient.
 - c) Loại bỏ convolutional layer.
 - d) Loại bỏ max-pooling layer.

35.3 Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
2. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
3. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần bài tập, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
4. **Triển khai streamlit:** Các bạn có thể truy cập vào web streamlit và link GitHub tại [đây](#).
5. **Rubric:**

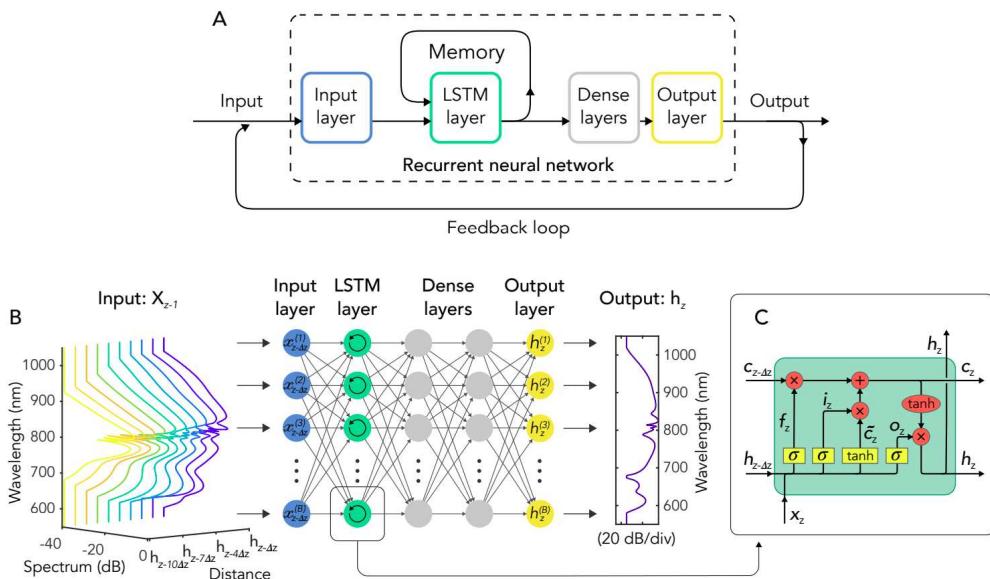
Mục	Kiến Thức	Đánh Giá
II.A	<ul style="list-style-type: none"> - Kiến thức về việc cài đặt chương trình huấn luyện mô hình ResNet từ đầu sử dụng PyTorch. - Kiến thức về các thành phần trong kiến trúc ResNet. - Kỹ thuật lập trình PyTorch. 	<ul style="list-style-type: none"> - Năm được cách cài đặt mô hình ResNet cho bài toán phân loại ảnh từ đầu sử dụng PyTorch. - Năm được các hàm thành phần liên quan đến ResNet.
II.B	<ul style="list-style-type: none"> - Kiến thức về việc cài đặt chương trình huấn luyện mô hình DenseNet từ đầu sử dụng PyTorch. - Kiến thức về các thành phần trong kiến trúc DenseNet. - Kỹ thuật lập trình PyTorch. 	<ul style="list-style-type: none"> - Năm được cách cài đặt mô hình DenseNet cho bài toán phân loại ảnh từ đầu sử dụng PyTorch. - Năm được các hàm thành phần liên quan đến DenseNet.
III.	<ul style="list-style-type: none"> - Lý thuyết về ResNet và DenseNet. - Code về PyTorch. 	<ul style="list-style-type: none"> - Năm được lý thuyết cơ bản về ResNet và DenseNet.

- Hết -

Chương 36

Recurrent Neural Networks

36.1 Giới thiệu



Data Cho đến nay, các kiến thức mà chúng ta đã học về các mô hình Machine Learning, chẳng hạn như Linear Regression, Logistic Regression, v.v., thường được áp dụng cho dữ liệu dạng bảng (tabular data), trong đó mỗi hàng đại diện cho một mẫu (sample) và mỗi cột đại diện cho một thuộc tính (attribute). Khi chuyển sang các mô hình như Multi-Layer Perceptron (MLP) hoặc Convolutional Neural Network (CNN), chúng ta làm việc với dữ liệu ảnh (image data), nơi đầu vào (input) là các pixel biểu diễn giá trị tại từng tọa độ trên hình ảnh.

Điểm chung giữa dữ liệu dạng bảng và dữ liệu ảnh là chúng đều có số lượng đặc trưng (features) rõ ràng và cụ thể, chẳng hạn như số lượng thuộc tính trong dữ liệu bảng hoặc số lượng pixel của một bức ảnh (ví dụ: MNIST dataset chứa các hình ảnh có kích thước 28 x 28 pixel). Đồng thời, các mô

hình được thiết kế cho các tác vụ regression hoặc classification cho những dữ liệu trên thường nhận input là các features của một sample data và đưa ra dự đoán là một output đơn lẻ (ví dụ: phân loại hình ảnh chó hoặc mèo, dự đoán giá nhà, chẩn đoán bệnh, v.v.).

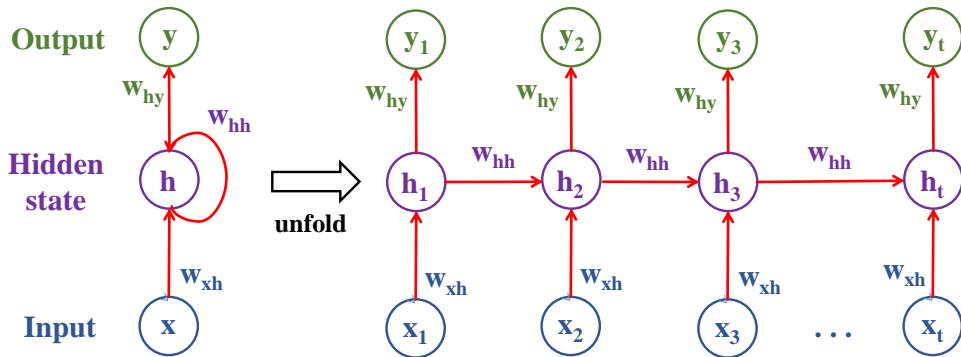
Tuy nhiên, có rất nhiều tác vụ học khác mà chúng ta sẽ phải làm việc với các data có độ dài linh hoạt như text, audio, time series. Đồng thời các models thiết kế cho các tác vụ trên cũng yêu cầu sau khi nhận vào input là sequential data, output dự đoán cũng sẽ có dạng sequential. Sẽ có nhiều sự thắc mắc về việc như thế nào là data có fixed length và flexible length, khi mà ảnh cũng sẽ có nhiều ảnh lớn nhỏ, tabular data cũng sẽ có nhiều bảng với số lượng attribute khác nhau. Để hiểu rõ, chúng ta cần phân rõ sự khác biệt giữa chúng:

- **Image data:** Ảnh có rất nhiều size khác nhau như 28x28, 32x32, 128x128, ... nhưng nhìn chung chúng ta đều có thể resize về một size nhất định phù hợp với cấu trúc mô hình.
- **Tabular data:** Đối với cùng một mẫu, như chẩn đoán bệnh cho bệnh nhân, có thể sẽ có bệnh nhân có số lượng thông tin nhiều hơn hẳn bệnh nhân khác, nhưng chung quy chúng ta đều phải quy chúng về 1 bảng (thực hiện feature engineering) và thực hiện dự đoán trên cùng một số lượng features nhất định.
- **Sequential data:** Dữ liệu dạng chuỗi lại có tính chất rất khác so với 2 cái trên, khi mà từng mẫu có độ dài khác nhau, ví dụ như mỗi câu trong văn bản sẽ có độ dài khác nhau, việc chuẩn hóa (như padding, truncate) sẽ dẫn tới mất ngữ nghĩa hoặc tạo ra nhiễu.

Vì vậy, để xử lý cho data dạng time sequence, chúng ta cần mô hình riêng biệt xử lý hiệu quả cho chúng, sẽ được tìm hiểu ở phần tiếp theo.

Mô hình Recurrent Neural Networks (RNNs):

Recurrent Neural Networks (RNNs) là một mạng neural nhân tạo, được thiết kế đặc biệt để xử lý dữ liệu tuần tự (sequential data), ứng dụng rộng rãi đối với các tác vụ như Time Series Prediction, Speech Recognition and Synthesis. Điểm đặc trưng của RNN nằm ở khả năng ghi nhớ thông tin từ các bước trước đó bằng cách sử dụng **trạng thái ẩn** (hidden states). Tại mỗi thời điểm t , trạng thái h_t được cập nhật dựa trên trạng thái của bước trước đó h_{t-1} và dữ liệu đầu vào hiện tại x_t . Từ trạng thái ẩn này, RNN có



thể đưa ra dự đoán tương ứng y_t , giúp mô hình học được mối quan hệ phụ thuộc trong chuỗi dữ liệu một cách hiệu quả.

RNN trong xử lý ngôn ngữ tự nhiên (NLP) Mô hình RNN là một công cụ mạnh mẽ trong việc xử lý ngôn ngữ tự nhiên (NLP), đặc biệt đối với các bài toán yêu cầu học mối quan hệ tuần tự giữa các từ trong câu. Với dữ liệu văn bản, mỗi từ trong chuỗi được biểu diễn bằng vector (word embedding), và RNN có thể mô hình hóa ngữ cảnh của từ dựa trên trạng thái ẩn h_t tại mỗi bước. Một ví dụ điển hình như sử dụng RNN trong bài toán dịch máy (machine translation), RNN có thể học cách mỗi từ phụ thuộc vào các từ trước đó, từ đó đưa ra kết quả chính xác hơn. Một trong những ưu điểm nổi bật là khả năng nắm bắt ngữ nghĩa của câu nhờ vào việc ghi nhớ trạng thái qua thời gian. Tuy nhiên, để xử lý các câu dài và mối quan hệ xa, các biến thể của RNN như LSTM và GRU thường được ưu tiên.

RNN trong bài toán Time Series Prediction (Forecasting) RNN cũng rất phù hợp để phân tích dữ liệu chuỗi thời gian, nơi các mẫu dữ liệu được ghi nhận tuần tự theo thời gian (ví dụ: giá cổ phiếu, nhiệt độ, lưu lượng mạng). Trong bài toán này, RNN sử dụng các hidden state để học mối quan hệ giữa các bước thời gian, giúp dự đoán các giá trị tương lai dựa trên các thông tin lịch sử. Thông thường, input của bài toán sẽ là các giá trị trong cửa sổ thời gian trước đó $\{y_{t-1}, y_{t-2}, \dots, y_{t-n}\}$, output là giá trị tương lai tại thời điểm t (hoặc nhiều thời điểm tiếp theo nếu dự báo nhiều bước). RNN có thể nắm bắt các xu hướng ngắn hạn và dài hạn trong dữ liệu, nhưng tương

tự bài toán text, nó cũng gặp khó khăn khi làm việc với chuỗi dài, yêu cầu các biến thể như LSTM hoặc GRU để tối ưu hóa.

Trong bài tập này ở phần lập trình, chúng ta sẽ thực hành xây dựng mô hình Classification và Regression có tích hợp các layer RNN, áp dụng vào giải quyết hai bài toán là Financial News Sentiment Analysis và Temperature Forecasting. Đồng thời, ôn tập một số lý thuyết về RNNs thông qua bài tập trắc nghiệm.

36.2 Bài tập

36.2.1 Phần lập trình

- **Problem 01: Sentiment Analysis for Financial News** Trong bài tập này, chúng ta sẽ xây dựng một mô hình về Text Classification dùng để phân loại tình hình tin tức tài chính là tích cực (positive), tiêu cực (negative) hay trung lập (neutral) dựa trên một đoạn văn có nội dung về tài chính cho trước. Các bạn sẽ thực hiện theo hướng dẫn sau:

1. Import các thư viện cần thiết:

```
1 import torch
2 import torch.nn as nn
3
4 seed = 1
5 torch.manual_seed(seed)
6
7 import os
8 import numpy as np
9 import pandas as pd
10 import matplotlib.pyplot as plt
11 import re
12 import nltk
13 import unidecode
14
15 nltk.download('stopwords')
16 from nltk.corpus import stopwords
17 from nltk.stem.porter import PorterStemmer
18
19 from torch.utils.data import Dataset, DataLoader
20 from sklearn.model_selection import train_test_split
```

2. **Tải bộ dữ liệu:** Các bạn tải bộ dữ liệu tại [đây](#). Ở đây, ta sẽ chỉ quan tâm đến file **all_data.csv**. Dưới đây là thông tin 4 hàng đầu tiên của bảng dữ liệu:

Index	sentiment	content
0	neutral	According to Gran , the company has no plans to move all production to although that is where the company is growing .
1	neutral	Technopolis plans to develop in stages an area of no less than 100,000 square meters in order to host companies working in computer technologies and telecommunications , the statement said .
2	negative	The international electronic industry company Elcoteq has laid off tens of workers from its Tallinn facility ; contrary to earlier layoffs the company contracted ranks of its office workers , the daily Postimees reported .
3	positive	With the new production plant the company would increase its capacity to meet the expected increase in demand and would improve the use of raw materials and therefore increase the production profitability .

Dựa vào bảng dữ liệu trên, ta xác định được rằng mô hình ta xây dựng sẽ sử dụng thông tin từ cột **content** để dự đoán nhãn tại cột **sentiment**.

3. **Đọc bộ dữ liệu:** Sử dụng thư viện pandas, các bạn đọc bộ dữ liệu lên như sau, lưu ý cần phải cài đặt tham số encoding='ISO-8859-1' để đưa về định dạng đúng như đoạn code như sau:

```

1 dataset_path = 'dataset/all-data.csv'
2 headers = ['sentiment', 'content']
3 df = pd.read_csv(
4     dataset_path,
5     names=headers,
6     encoding='ISO-8859-1'
7 )

```

Từ đây, ta có thể xác định được danh sách các class của bài toán bằng phương thức unique() của Pandas, đồng thời ta cũng sẽ đổi class dạng string thành số nguyên đại diện cho ID của chúng:

```

1 classes = {
2     class_name: idx for idx, class_name in enumerate(
3         df['sentiment'].unique().tolist()
4     )
5     df['sentiment'] = df['sentiment'].apply(lambda x:
6         classes[x])

```

4. **Tiền xử lý dữ liệu:** Để tận dụng hàm apply của pandas, ta sẽ tiền xử lý văn bản cột **content** ngay tại đây. Đầu tiên, định nghĩa

hàm chuẩn hóa, nhận tham số đầu vào là 1 text và trả về 1 text đã được chuẩn hóa:

```

1 english_stop_words = stopwords.words('english')
2 stemmer = PorterStemmer()
3
4 def text_normalize(text):
5     text = text.lower()
6     text = unidecode.unidecode(text)
7     text = text.strip()
8     text = re.sub(r'[^w\s]', ' ', text)
9     text = ' '.join([word for word in text.split(' ')
10                     if word not in english_stop_words])
11    text = ' '.join([stemmer.stem(word) for word in
12                      text.split(' ')])
13
14    return text

```

Các kỹ thuật chuẩn hóa được áp dụng trong bài bao gồm: Chuyển chữ viết thường (Lowercasing), Xóa dấu câu (Punctuation Removal), Xóa stopwords (Stopwords Removal), Stemming.

Sau đó, áp dụng hàm text_normalize() vào cột **content**:

```

1 df['content'] = df['content'].apply(lambda x:
2                                         text_normalize(x))

```

5. **Xây dựng bộ từ vựng:** Để huấn luyện dữ liệu văn bản, ta cần chuyển đổi các văn bản thành vector. Phương thức đơn giản nhất, ta sẽ thu thập toàn bộ các từ trong bộ dữ liệu thành một tập từ vựng, mỗi từ có một ID riêng. Từ đó, với một văn bản đầu, ta quy đổi sang ID tương ứng, như vậy sẽ được một vector số nguyên. Đoạn code tạo bộ từ vựng được thực hiện như sau:

```

1 vocab = []
2 for sentence in df['content'].tolist():
3     tokens = sentence.split()
4     for token in tokens:
5         if token not in vocab:
6             vocab.append(token)
7
8 vocab.append('UNK')
9 vocab.append('PAD')
10 word_to_idx = {word: idx for idx, word in enumerate(
11                         vocab)}
11 vocab_size = len(vocab)

```

Trong đó, '**UNK**' đại diện cho các từ không thuộc bộ từ vựng và '**PAD**' đại diện cho các ô trống được thêm vào để thỏa mãn độ dài tối thiểu của một văn bản mà ta quy định về sau.

Với bộ từ vựng trên, ta xây dựng hàm `transform()` dùng để chuyển đổi văn bản thành danh sách các số nguyên như sau:

```

1 def transform(text, word_to_idx, max_seq_len):
2     tokens = []
3     for w in text.split():
4         try:
5             w_ids = word_to_idx[w]
6         except:
7             w_ids = word_to_idx['UNK']
8         tokens.append(w_ids)
9
10    if len(tokens) < max_seq_len:
11        tokens += [word_to_idx['PAD']] * (max_seq_len
12 - len(tokens))
13    elif len(tokens) > max_seq_len:
14        tokens = tokens[:max_seq_len]
15
16    return tokens

```

Trong đó, `word_to_idx` đại diện cho bộ từ vựng và `max_seq_len` là một hằng số quy ước độ dài của các văn bản, đảm bảo các văn bản có cùng độ dài.

6. **Chia bộ dữ liệu train, val, test:** Ta dùng hàm `train_test_split` của thư viện scikit-learn để chia bộ dữ liệu thành 3 bộ train/val/test theo tỷ lệ 7:2:1:

```

1 val_size = 0.2
2 test_size = 0.125
3 is_shuffle = True
4 texts = df['content'].tolist()
5 labels = df['sentiment'].tolist()
6
7 X_train, X_val, y_train, y_val = train_test_split(
8     texts, labels,
9     test_size=val_size,
10    random_state=seed,
11    shuffle=is_shuffle
12 )
13
14 X_train, X_test, y_train, y_test = train_test_split(

```

```
15     X_train, y_train,
16     test_size=val_size,
17     random_state=seed,
18     shuffle=is_shuffle
19 )
```

7. **Xây dựng pytorch datasets:** Ta triển khai class tên FinancialNews với đầu vào gồm cặp dữ liệu đầu vào X - y, bộ từ điển cũng như hàm transform như sau:

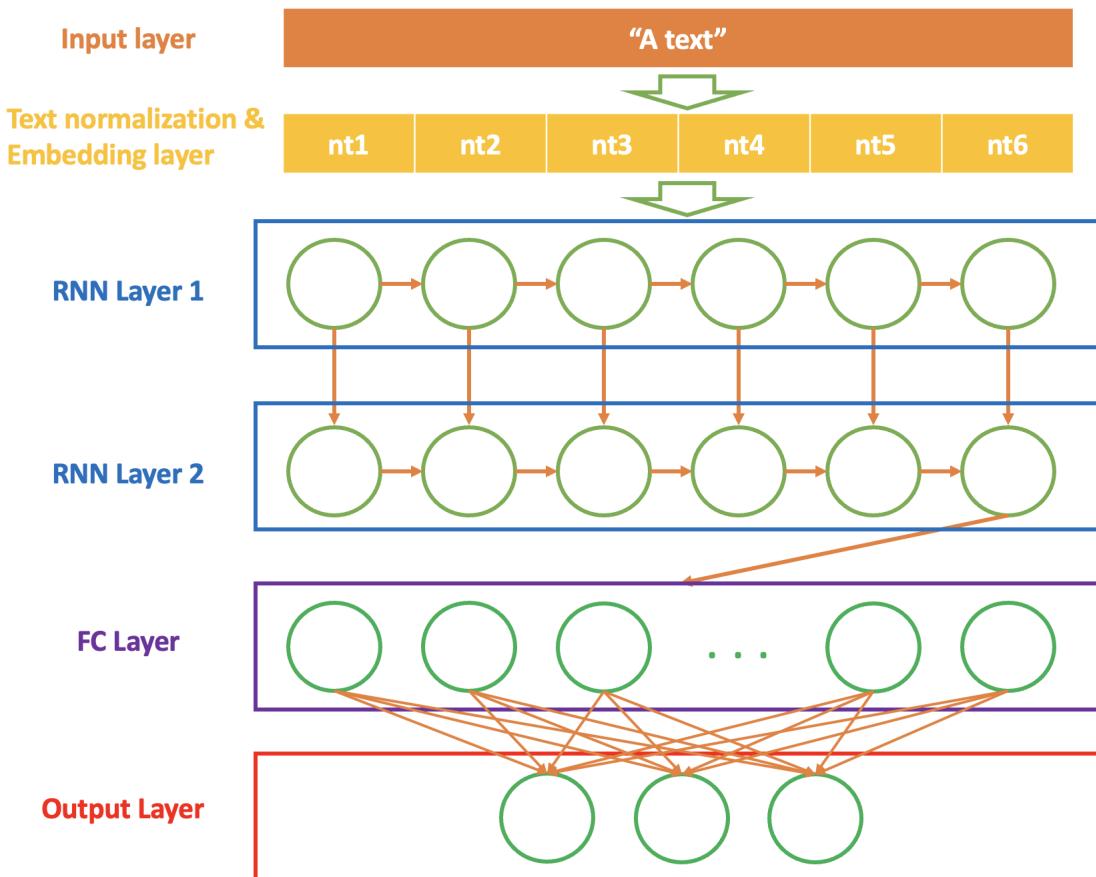
```
1 class FinancialNews(Dataset):
2     def __init__(
3         self,
4         X, y,
5         word_to_idx,
6         max_seq_len,
7         transform=None
8     ):
9         self.texts = X
10        self.labels = y
11        self.word_to_idx = word_to_idx
12        self.max_seq_len = max_seq_len
13        self.transform = transform
14
15    def __len__(self):
16        return len(self.texts)
17
18    def __getitem__(self, idx):
19        text = self.texts[idx]
20        label = self.labels[idx]
21
22        if self.transform:
23            text = self.transform(
24                text,
25                self.word_to_idx,
26                self.max_seq_len
27            )
28        text = torch.tensor(text)
29
30    return text, label
```

8. Khai báo dataloader:

```
1 max_seq_len = 32
2
3 train_dataset = FinancialNews(
```

```
4     X_train, y_train,
5     word_to_idx=word_to_idx,
6     max_seq_len=max_seq_len,
7     transform=transform
8 )
9 val_dataset = FinancialNews(
10    X_val, y_val,
11    word_to_idx=word_to_idx,
12    max_seq_len=max_seq_len,
13    transform=transform
14 )
15 test_dataset = FinancialNews(
16    X_test, y_test,
17    word_to_idx=word_to_idx,
18    max_seq_len=max_seq_len,
19    transform=transform
20 )
21
22 train_batch_size = 128
23 test_batch_size = 8
24
25 train_loader = DataLoader(
26    train_dataset,
27    batch_size=train_batch_size,
28    shuffle=True
29 )
30 val_loader = DataLoader(
31    val_dataset,
32    batch_size=test_batch_size,
33    shuffle=False
34 )
35 test_loader = DataLoader(
36    test_dataset,
37    batch_size=test_batch_size,
38    shuffle=False
39 )
```

9. **Xây dựng mô hình:** Chúng ta sẽ xây dựng phân loại cảm xúc với 2 layer RNNs. Vector tại hidden state cuối cùng của RNN layer thứ 2 sẽ được đưa vào FC layer để thực hiện dự đoán. Các bạn có thể quan sát rõ hơn thông qua hình dưới đây:



Hình 36.1: Ảnh minh họa kiến trúc của mô hình

Như vậy, code cài đặt như sau:

```

1 class SentimentClassifier(nn.Module):
2     def __init__(self, vocab_size, embedding_dim,
3                  hidden_size, n_layers, n_classes,
4                  dropout_prob):
5         super(SentimentClassifier, self).__init__()
6         self.embedding = nn.Embedding(vocab_size, embedding_dim)
7         self.rnn = nn.RNN(embedding_dim, hidden_size,
8                          n_layers, batch_first=True)
9         self.norm = nn.LayerNorm(hidden_size)
10

```

```

11         self.dropout = nn.Dropout(dropout_prob)
12         self.fc1 = nn.Linear(hidden_size, 16)
13         self.relu = nn.ReLU()
14         self.fc2 = nn.Linear(16, n_classes)
15
16     def forward(self, x):
17         x = self.embedding(x)
18         x, hn = self.rnn(x)
19         x = x[:, -1, :]
20         x = self.norm(x)
21         x = self.dropout(x)
22         x = self.fc1(x)
23         x = self.relu(x)
24         x = self.fc2(x)
25
26     return x

```

Ta định nghĩa class với tên SentimentClassifier, nhận tham số đầu vào gồm: kích thước bộ từ vựng (vocab_size), kích thước không gian vector của mỗi từ trong chuỗi đầu vào (embedding_dim), kích thước không gian vector của hidden state (hidden_size), số lượng RNN layer (n_layers), số class dự đoán của bài toán (n_classes) và tỉ lệ dropout (dropout_prob).

Với class trên, ta khai báo mô hình SentimentClassifier:

```

1 n_classes = len(list(classes.keys()))
2 embedding_dim = 64
3 hidden_size = 64
4 n_layers = 2
5 dropout_prob = 0.2
6 device = 'cuda' if torch.cuda.is_available() else 'cpu'
7
8 model = SentimentClassifier(
9     vocab_size=vocab_size,
10    embedding_dim=embedding_dim,
11    hidden_size=hidden_size,
12    n_layers=n_layers,
13    n_classes=n_classes,
14    dropout_prob=dropout_prob
15 ).to(device)

```

10. **Cài đặt hàm loss và optimizer:** Ta sử dụng hàm loss crossentropy cho bài toán phân loại và Adam optimizer.

```
1 lr = 1e-4
2 epochs = 50
3
4 criterion = nn.CrossEntropyLoss()
5 optimizer = torch.optim.Adam(
6     model.parameters(),
7     lr=lr
8 )
```

11. **Thực hiện huấn luyện:** Ta định nghĩa hàm fit() và evaluate() dùng để huấn luyện và đánh giá mô hình như sau:

```
1 def fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     device,
8     epochs
9 ):
10    train_losses = []
11    val_losses = []
12
13    for epoch in range(epochs):
14        batch_train_losses = []
15
16        model.train()
17        for idx, (inputs, labels) in enumerate(
18            train_loader):
19            inputs, labels = inputs.to(device),
20            labels.to(device)
21
22            optimizer.zero_grad()
23            outputs = model(inputs)
24            loss = criterion(outputs, labels)
25            loss.backward()
26            optimizer.step()
27
28            batch_train_losses.append(loss.item())
29
30            train_loss = sum(batch_train_losses) / len(
31                batch_train_losses)
32            train_losses.append(train_loss)
33
34            val_loss, val_acc = evaluate(
```

```

32             model, val_loader,
33             criterion, device
34         )
35         val_losses.append(val_loss)
36
37         print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal loss: {val_loss:.4f}')
38
39     return train_losses, val_losses

1 def evaluate(model, dataloader, criterion, device):
2     model.eval()
3     correct = 0
4     total = 0
5     losses = []
6     with torch.no_grad():
7         for inputs, labels in dataloader:
8             inputs, labels = inputs.to(device),
labels.to(device)
9             outputs = model(inputs)
10            loss = criterion(outputs, labels)
11            losses.append(loss.item())
12            _, predicted = torch.max(outputs.data, 1)
13            total += labels.size(0)
14            correct += (predicted == labels).sum().
item()
15
16    loss = sum(losses) / len(losses)
17    acc = correct / total
18
19    return loss, acc

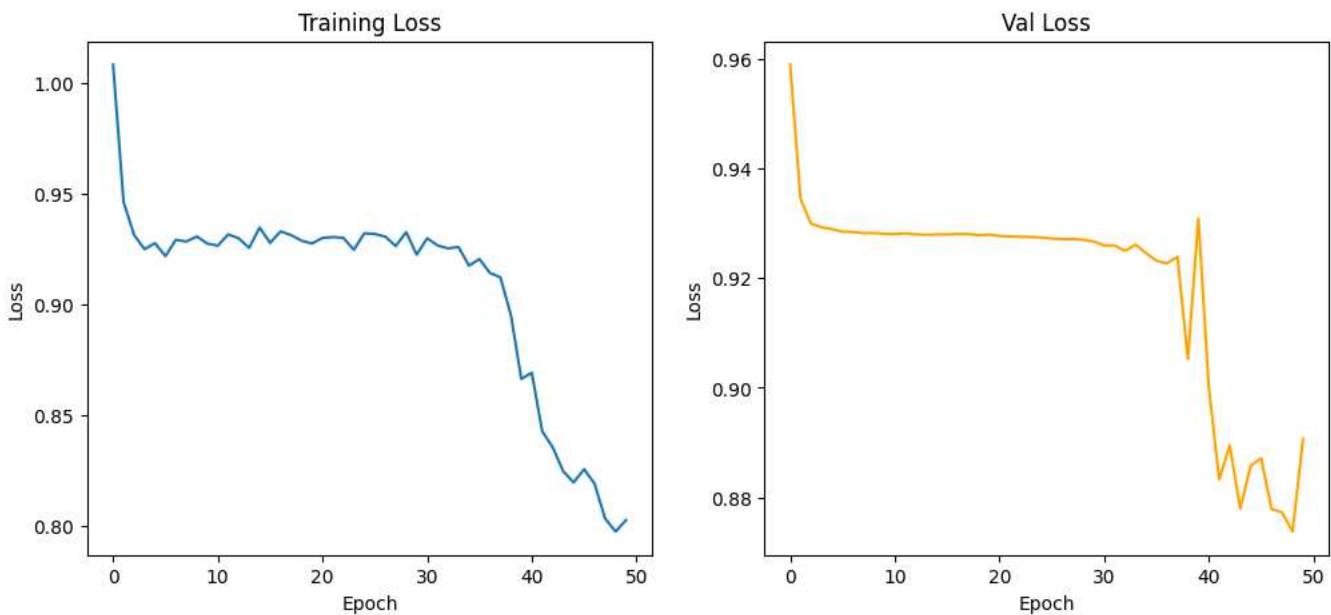
```

Tổng hợp tất cả các dữ kiện trên, ta tiến hành huấn luyện mô hình SentimentClassifier:

```

1 train_losses, val_losses = fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     device,
8     epochs
9 )

```



Hình 36.2: Kết quả huấn luyện mô hình SentimentClassifier

12. **Đánh giá mô hình:** Sử dụng hàm evaluate() đã xây dựng ở trên, ta đánh giá mô hình trên hai tập val và test như sau:

```

1 val_loss, val_acc = evaluate(
2     model,
3     val_loader,
4     criterion,
5     device
6 )
7 test_loss, test_acc = evaluate(
8     model,
9     test_loader,
10    criterion,
11    device
12 )
13
14 print('Evaluation on val/test dataset')
15 print('Val accuracy: ', val_acc)
16 print('Test accuracy: ', test_acc)

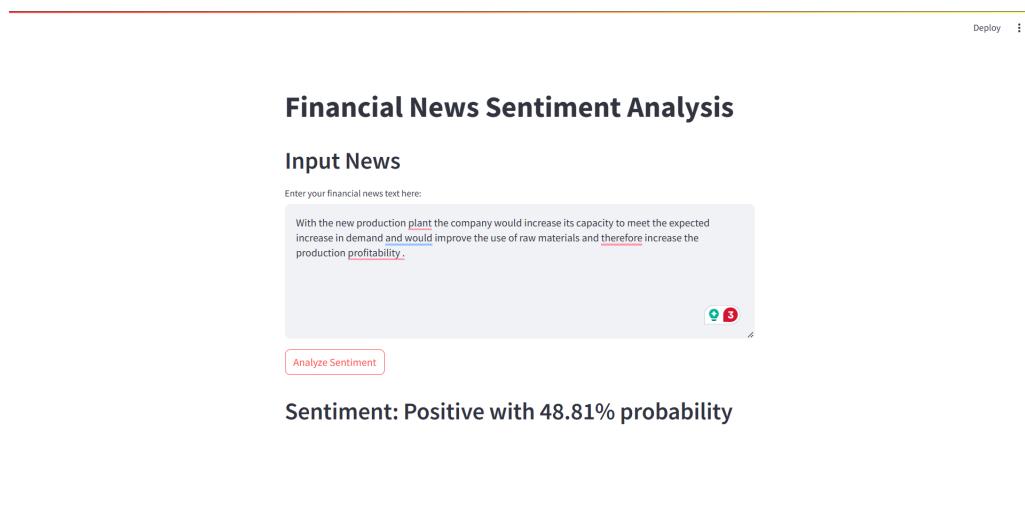
```

Lưu ý: Dựa vào phần code mẫu trên (dùng RNN), các bạn hãy thực hiện bài tập với các version bao gồm LSTM và BiLSTM (các

bạn tham khảo cách sử dụng LSTM/BiLSTM trong pytorch tại [đây](#)) với các config về tham số không đổi.

13. **Triển khai mô hình:** Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:

- [Link Streamlit](#)
- [Github](#)
- Giao Diện



Hình 36.3: Giao diện ứng dụng

• Problem 02: Hourly Temperature Forecasting

Trong bài tập này, chúng ta sẽ xây dựng một mô hình về Times Series dùng để dự đoán nhiệt độ trong 1 giờ tiếp theo dựa trên nhiệt độ của 6 giờ trước đó. Các bạn sẽ thực hiện theo hướng dẫn sau:

1. Import các thư viện cần thiết:

```

1 import torch
2 import torch.nn as nn
3
4 seed = 1
5 torch.manual_seed(seed)
6
7 import numpy as np

```

```

8 import pandas as pd
9 import matplotlib.pyplot as plt
10
11 from torch.utils.data import Dataset, DataLoader

```

2. **Tải bộ dữ liệu:** Các bạn tải bộ dữ liệu tại [đây](#). Dưới đây là thông tin 4 hàng đầu tiên của bảng dữ liệu:

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	Data Summary
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Partly cloudy throughout
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Partly cloudy throughout
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Partly cloudy throughout
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Partly cloudy throughout

3. Chuẩn bị dữ liệu:

(a) Đọc dữ liệu từ file .csv:

```

1 dataset_filepath = 'dataset/weatherHistory.csv'
2 df = pd.read_csv(dataset_filepath)

```

Trong bài tập lần này, vì mô hình chỉ dự đoán nhiệt độ (Temperature (C)) nên ta sẽ loại bỏ đi các cột không cần thiết trong DataFrame trước khi đưa vào tiền xử lý:

```

1 univariate_df = df['Temperature (C)']
2 univariate_df.index = df['Formatted Date']

```

Như vậy, dữ liệu bảng của chúng ta sẽ có dạng:

Formatted Date	Temperature (C)
2006-04-01 00:00:00.000 +0200	9.472222
2006-04-01 01:00:00.000 +0200	9.355556
2006-04-01 02:00:00.000 +0200	9.377778
2006-04-01 03:00:00.000 +0200	8.288889

Bảng 36.1: 4 hàng đầu tiên của bảng dữ liệu sau khi đã xóa đi các cột không cần thiết

- (b) **Xây dựng hàm tạo cặp X, y:** Cũng như bất kì các bài toán thuộc nhánh học có giám sát trên dữ liệu dạng bảng khác, ta cần xác định X, y của bài toán sao cho phù hợp. Thông thường, X sẽ là các đặc trưng và y sẽ là nhãn tương ứng cho các đặc trưng X (thường được định nghĩa rõ ràng trong bảng dữ liệu). Tuy nhiên ở bài toán này, ta không có các nhãn cụ thể theo như đề bài đưa ra. Chính vì vậy, cần phải thực hiện chỉnh sửa Input/Output của dữ liệu sao cho phù hợp để ta có thể đưa vào mô hình.

Với việc đề bài yêu cầu sử dụng thông tin nhiệt độ của 6 giờ trước để dự đoán nhiệt độ của 1 giờ tiếp theo, ta có thể xác định được rằng X sẽ là nhiệt độ của 6 giờ, y là nhiệt độ của giờ tiếp theo, định nghĩa này có thể được thể hiện như hình dưới đây:

Time	Temperature (C)
2006-04-01 00:00:00.000 +0200	9.472222
2006-04-01 01:00:00.000 +0200	9.355556
2006-04-01 02:00:00.000 +0200	9.377778
2006-04-01 03:00:00.000 +0200	8.288889
2006-04-01 04:00:00.000 +0200	8.755556
2006-04-01 05:00:00.000 +0200	9.222222
2006-04-01 06:00:00.000 +0200	7.733333

Hình 36.4: Cách xác định X, y dựa trên 7 mẫu của bảng dữ liệu cho trước

Từ đây, có thể thấy khi ta duyệt qua từng hàng trên bảng dữ liệu, ta có thể tạo ra các cặp X, y theo đúng ý của đề bài yêu cầu, nhờ đó có thể sử dụng để đưa vào huấn luyện mô hình.

Kỹ thuật này được gọi là **Windowing** (coi các cặp X, y như các "cửa sổ"). Ta sẽ thực hiện triển khai hàm windowing này như sau:

```

1 input_size = 6
2 label_size = 1
3 offset = 1
4
5 def slicing_window(df, df_start_idx, df_end_idx,
6                     input_size, label_size, offset):
7     features = []
8     labels = []
9
10    window_size = input_size + offset
11
12    if df_end_idx == None:
13        df_end_idx = len(df) - window_size
14
15    for idx in range(df_start_idx, df_end_idx):
16        feature_end_idx = idx + input_size
17        label_start_idx = idx + window_size -
18        label_size
19
20        feature = df[idx:feature_end_idx]
21        label = df[label_start_idx:(idx+
22        window_size)]
23
24        features.append(feature)
25        labels.append(label)
26
27    features = np.expand_dims(np.array(features),
28                             -1)
29    labels = np.array(labels)
30
31    return features, labels

```

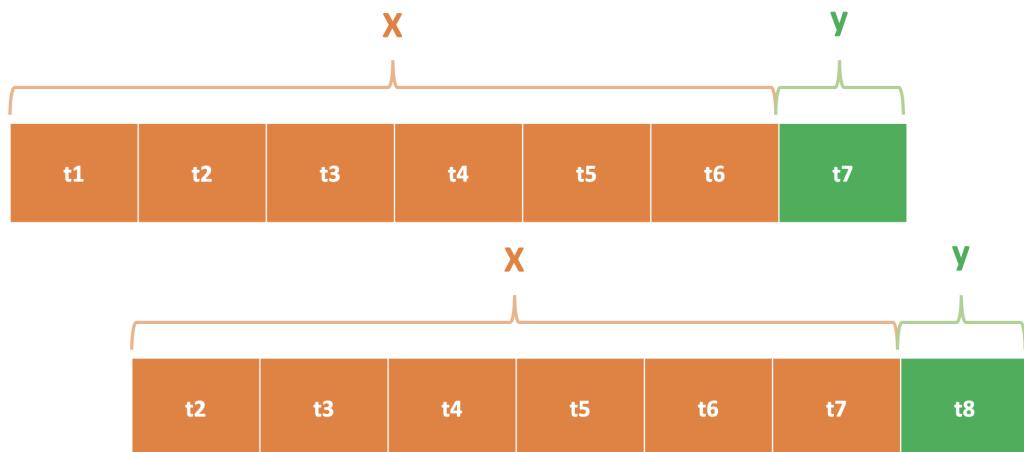
Ý nghĩa của các tham số đầu vào là:

- **df**: DataFrame của bảng dữ liệu.
- **df_start_idx**: Chỉ mục bắt đầu thực hiện "windowing" trong bảng dữ liệu.
- **df_end_idx**: Chỉ mục kết thúc thực hiện "windowing" trong bảng dữ liệu.
- **input_size**: Kích thước (số thời gian) của X.
- **label_size**: Kích thước (số thời gian) của y.

- **offset:** Khoảng cách về thời gian giữa X và y.
Đối với yêu cầu đề bài, có thể dễ dàng xác định được rằng `input_size = 6`, `output_size = 1` và `offset = 1`.

Trong đó:

- **Dòng 6, 7, 9, 11, 12:** Tạo hai list dùng để chứa các mẫu dữ liệu X, y. Sau đó thực hiện tính kích thước của cửa sổ = `input_size + offset`.
- **Dòng 14, 15, 16, 18, 19, 21, 22:** Bắt đầu duyệt qua từng mẫu dữ liệu theo chỉ mục trong khoảng (`df_start_idx`, `df_end_idx`), ta sẽ thực hiện tìm X và y và thêm vào danh sách chứa features, labels đã khai báo phía trên.
- **Dòng 24, 25, 27:** Cuối cùng, ta chuyển đổi 2 list thành `np.ndarray` và sử dụng chúng làm kết quả trả về của hàm:



Hình 36.5: Kỹ thuật Windowing: duyệt qua từng chỉ mục trong bảng dữ liệu để xác định cặp X, y

4. **Chia bộ dữ liệu train, val, test:** Sử dụng hàm `slicing_window()` đã định nghĩa ở trên, ta tiến hành chia ba bộ dữ liệu train, val, test như sau:

```

1 dataset_length = len(univariate_df)
2 train_size = 0.7

```

```
3 val_size = 0.2
4 train_end_idx = int(train_size * dataset_length)
5 val_end_idx = int(val_size * dataset_length) +
    train_end_idx
6
7 X_train, y_train = slicing_window(
8     univariate_df,
9     df_start_idx=0,
10    df_end_idx=train_end_idx,
11    input_size=input_size,
12    label_size=label_size,
13    offset=offset
14 )
15
16 X_val, y_val = slicing_window(
17     univariate_df,
18     df_start_idx=train_end_idx,
19     df_end_idx=val_end_idx,
20     input_size=input_size,
21     label_size=label_size,
22     offset=offset
23 )
24
25 X_test, y_test = slicing_window(
26     univariate_df,
27     df_start_idx=val_end_idx,
28     df_end_idx=None,
29     input_size=input_size,
30     label_size=label_size,
31     offset=offset
32 )
```

Ở đây, ta chia bộ dữ liệu theo tỉ lệ 7/2/1, tương ứng cho train/val/test.

5. Xây dựng pytorch datasets:

```
1 class WeatherForecast(Dataset):
2     def __init__(self,
3                  X,
4                  y,
5                  transform=None):
6         self.X = X
7         self.y = y
8         self.transform = transform
9
10    def __len__(self):
```

```
12         return len(self.X)
13
14     def __getitem__(self, idx):
15         X = self.X[idx]
16         y = self.y[idx]
17
18         if self.transform:
19             X = self.transform(X)
20
21         X = torch.tensor(X, dtype=torch.float32)
22         y = torch.tensor(y, dtype=torch.float32)
23
24     return X, y
```

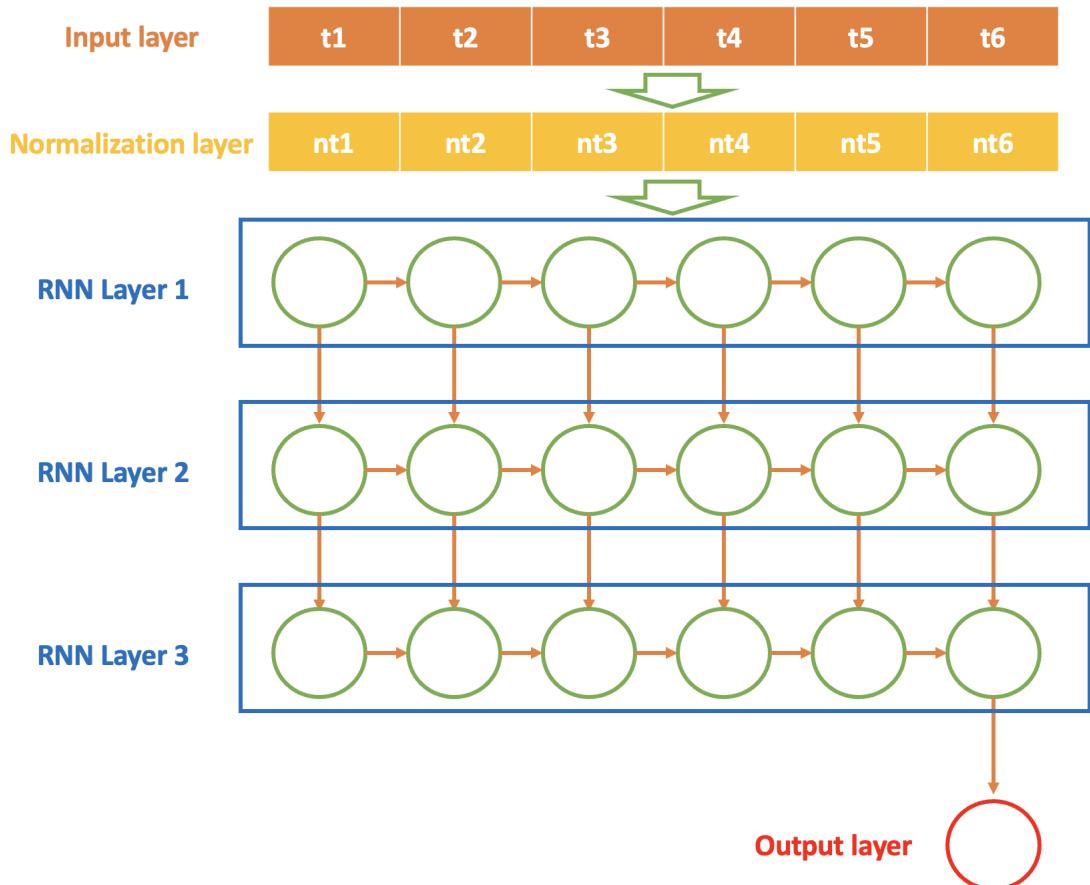
6. Khai báo dataloader:

```
1 train_dataset = WeatherForecast(
2     X_train, y_train
3 )
4 val_dataset = WeatherForecast(
5     X_val, y_val
6 )
7 test_dataset = WeatherForecast(
8     X_test, y_test
9 )
10
11 train_batch_size = 128
12 test_batch_size = 8
13
14 train_loader = DataLoader(
15     train_dataset,
16     batch_size=train_batch_size,
17     shuffle=True
18 )
19 val_loader = DataLoader(
20     val_dataset,
21     batch_size=test_batch_size,
22     shuffle=False
23 )
24 test_loader = DataLoader(
25     test_dataset,
26     batch_size=test_batch_size,
27     shuffle=False
28 )
```

7. Xây dựng mô hình:

Ta xây dựng class WeatherForecastor dùng

để dự đoán nhiệt độ ứng dụng 3 lớp RNN, vector hidden state cuối cùng của lớp RNN thứ 3 sẽ được đưa vào lớp FC để thực hiện dự đoán. Các bạn có thể quan sát rõ hơn ở hình sau:



Hình 36.6: Ảnh minh họa kiến trúc của mô hình

```

1 class WeatherForecaster(nn.Module):
2     def __init__(self, embedding_dim, hidden_size,
3                  n_layers, dropout_prob):
4         super(WeatherForecaster, self).__init__()
5         self.rnn = nn.RNN(embedding_dim, hidden_size,
6                          n_layers, batch_first=True)
7         self.norm = nn.LayerNorm(hidden_size)
8

```

```

9
10         self.dropout = nn.Dropout(dropout_prob)
11         self.fc = nn.Linear(hidden_size, 1)
12
13     def forward(self, x):
14         x, hn = self.rnn(x)
15         x = x[:, -1, :]
16         x = self.norm(x)
17         x = self.dropout(x)
18         x = self.fc(x)
19
20     return x

```

Sử dụng class đã định nghĩa ở trên, ta khai báo mô hình WeatherForecaster như sau:

```

1 embedding_dim = 1
2 hidden_size = 8
3 n_layers = 3
4 dropout_prob = 0.2
5 device = 'cuda' if torch.cuda.is_available() else 'cpu'
6
7 model = WeatherForecaster(
8     embedding_dim=embedding_dim,
9     hidden_size=hidden_size,
10    n_layers=n_layers,
11    dropout_prob=dropout_prob
12 ).to(device)

```

8. **Cài đặt hàm loss và optimizer:** Vì bài toán dự đoán nhiệt độ thời tiết này dưới dạng là bài Regression, ta sẽ sử dụng hàm loss là MSE và optimizer là Adam.

```

1 lr = 1e-3
2 epochs = 50
3
4 criterion = nn.MSELoss()
5 optimizer = torch.optim.Adam(
6     model.parameters(),
7     lr=lr
8 )

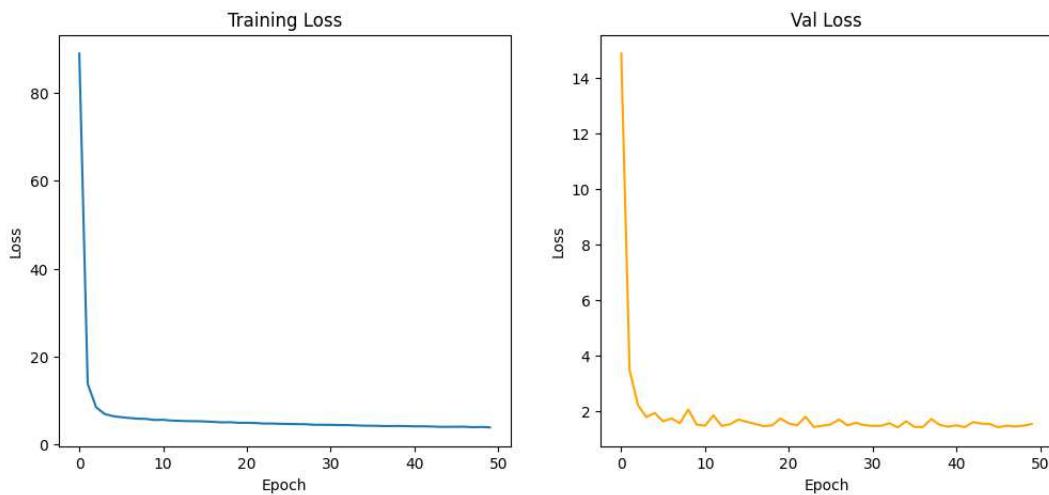
```

9. **Thực hiện huấn luyện mô hình:** Ta sử dụng hàm fit() và evaluate() đã định nghĩa ở bài Sentiment Analysis trước, thực hiện huấn luyện mô hình dự báo nhiệt độ như sau:

```

1 train_losses, val_losses = fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     device,
8     epochs
9 )

```



Hình 36.7: Trực quan hóa kết quả loss trong quá trình huấn luyện mô hình

10. **Đánh giá mô hình:** Ta sử dụng hàm evaluate() để đánh giá mô hình đã huấn luyện trên tập val và test như sau:

```

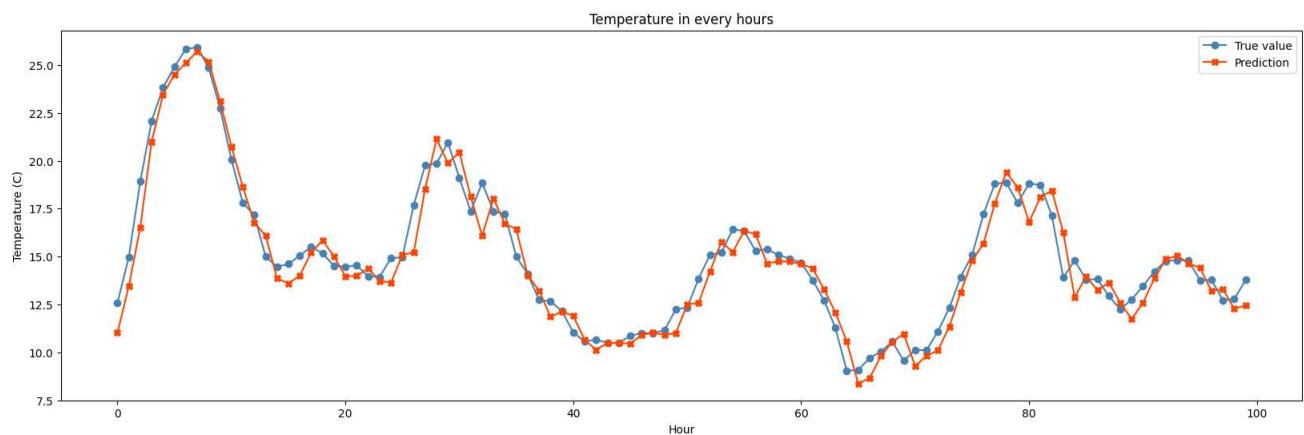
1 val_loss = evaluate(
2     model,
3     val_loader,
4     criterion,
5     device
6 )
7 test_loss = evaluate(
8     model,
9     test_loader,
10    criterion,
11    device
12 )
13

```

```
14 print('Evaluation on val/test dataset')
15 print('Val loss: ', val_loss)
16 print('Test loss: ', test_loss)
```

Bên cạnh đó, đối với bài toán time series, ta cũng có thể plot dự đoán của mô hình so với dữ liệu thực tế trong một khoảng thời gian như sau:

```
1 def plot_difference(y, pred):
2     plt.figure(figsize=(20, 6))
3     times = range(len(y))
4     y_to_plot = y.flatten()
5     pred_to_plot = pred.flatten()
6
7     plt.plot(times, y_to_plot, color='steelblue',
8             marker='o', label='True value')
9     plt.plot(times, pred_to_plot, color='orangered',
10            marker='X', label='Prediction')
11
12    plt.title('Temperature in every hours')
13    plt.xlabel('Hour')
14    plt.ylabel('Temperature (C)')
15    plt.legend()
16    plt.show()
17
18 inputs = torch.tensor(X_test[:100], dtype=torch.
19                      float32).to(device)
20 model.eval()
21 with torch.no_grad():
22     outputs = model(inputs).detach().cpu().numpy()
23 plot_difference(y_test[:100], outputs)
```



Hình 36.8: Kết quả dự đoán so với kết quả thực tế trong một khoảng thời gian

Lưu ý: Dựa vào phần code mẫu trên (dùng RNN), các bạn hãy thực hiện bài tập với các version bao gồm LSTM và BiLSTM (các bạn tham khảo cách sử dụng LSTM/BiLSTM trong pytorch tại [đây](#)) với các config về tham số không đổi.

- **Triển khai mô hình:** Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:

- [Link Streamlit](#)
- [Github](#)
- Giao Diện

Deploy :

Hourly Temperature Forecasting

Example: 24.86, 22.75, 20.07, 17.81, 17.16, 15.01 Target: 14.47

Enter 6 temperatures separated by commas:

24.86, 22.75, 20.07, 17.81, 17.16, 15.01

Predict

Predicted Temperature for the Next Hour:

Predicted Temperature: 13.67°C

Hình 36.9: Giao diện ứng dụng

36.2.2 Phản trắc nghiệm

1. Đặc điểm của dữ liệu dạng chuỗi (sequential data) là gì?
 - (A). Mỗi mẫu có độ dài cố định
 - (B). Dữ liệu có tính chất không thay đổi theo thời gian
 - (C). Mỗi mẫu có độ dài khác nhau và có thể mất ngữ nghĩa khi chuẩn hóa
 - (D). Mỗi mẫu có số lượng thuộc tính cố định
2. Trong phần xử lý dữ liệu của **Problem 01** tại sao cần thêm token '**UNK**' và '**PAD**' vào bộ từ vựng?
 - (A). Token '**UNK**' dùng để thay thế các từ không có trong bộ từ vựng, còn token '**PAD**' giúp chuẩn hóa độ dài của các chuỗi đầu vào.
 - (B). Token '**UNK**' dùng để giảm độ dài của văn bản, còn token '**PAD**' dùng để tạo ra các từ mới.
 - (C). Token '**UNK**' dùng để xóa bỏ các từ không quan trọng, còn token '**PAD**' để thay thế từ khóa.
 - (D). Token '**UNK**' và '**PAD**' đều dùng để cải thiện hiệu suất mô hình học máy mà không thay đổi dữ liệu đầu vào.
3. "input_size + offset" trong phần Hourly Temperature Forecasting thể hiện điều gì?
 - (A). Số lượng dự đoán cần thiết cho mỗi bước tính toán
 - (B). Kích thước cửa sổ lấy cặp dữ liệu
 - (C). Khoảng cách giữa các mẫu dữ liệu trong bộ dữ liệu
 - (D). Kích thước của dữ liệu đầu vào
4. Mô hình RNN có bao nhiêu bộ tham số (không kể bias)?
 - (A). Có 1 bộ tham số
 - (B). Không xác định, tùy vào số lượng time step của mô hình

RNN

- (C). Có 3 bộ tham số
- (D). Có 2 bộ tham số

5. Nhược điểm của mô hình RNN là?

- (A). Mô hình RNN chỉ có thể xử lý dữ liệu chuỗi ngắn.
- (B). RNN không thể học các mối quan hệ giữa các bước thời gian gần nhau.
- (C). RNN không thể áp dụng cho các bài toán phân loại.
- (D). Mô hình RNN gặp phải vấn đề vanishing gradient và không thể học các mối quan hệ dài hạn hiệu quả.

6. Cho đoạn code sau

```
1 import torch
2 import torch.nn as nn
3
4 input_tensor = torch.randn(5, 10, 10)
5
6 lstm = nn.LSTM(input_size=10, hidden_size=20, num_layers=2,
7     batch_first=True)
7 bilstm = nn.LSTM(input_size=10, hidden_size=20, num_layers=2,
8     batch_first=True, bidirectional=True)
8
9 output_lstm, (hidden_state_lstm, cell_state_lstm) = lstm(
10    input_tensor)
10
11 output_bilstm, (hidden_state_bilstm, cell_state_bilstm) =
12    bilstm(input_tensor)
12
13 print(output_lstm.shape)
14 print(output_bilstm.shape)
```

Khi đưa một tensor có kích thước cố định (5, 10, 10) cho các lớp LSTM và BiLSTM được định nghĩa như code trên, kích thước đầu ra của mỗi lớp sẽ khác nhau như thế nào (xét theo biến trong hàm print)?

- (A). LSTM sẽ tạo ra đầu ra với kích thước (5, 10, 20) và BiLSTM với kích thước (5, 10, 400).

- (B). Cả LSTM và BiLSTM đều output ra tensor có shape là (5, 10, 20).
- (C). Output của BiLSTM có số hidden state gấp đôi so với LSTM.
- (D). BiLSTM sẽ tạo ra đầu ra với số sequence length nhỏ hơn so với LSTM do xử lý hai chiều.

7. Đâu là điểm mạnh của LSTM so với RNN?

- (A). LSTM có thể xử lý tốt với chuỗi dài
- (B). LSTM có tốc độ xử lý nhanh hơn
- (C). LSTM yêu cầu ít dữ liệu huấn luyện hơn
- (D). LSTM yêu cầu tài nguyên tính toán ít hơn

8. Dòng code ‘`x = x[:, -1, :]`’ ở trong class WeatherForecaster có ý nghĩa là gì?

- (A). Lấy giá trị output của tất cả những time step trong chuỗi
- (B). Lấy giá trị output của time step cuối cùng của chuỗi
- (C). Lấy batch đầu tiên của dữ liệu
- (D). Lấy giá trị hidden state cuối cùng của mô hình RNN

9. Đâu không phải là một ứng dụng chính của RNN?

- (A). Text Classification
- (B). Stock Trading Prediction
- (C). Image Classification
- (D). Machine Translation

10. Ý nghĩa chính của hidden state trong RNN là?

- (A). Dùng để lưu trữ thông tin của các thời điểm trước
- (B). Dùng để biểu diễn trọng số của input
- (C). Dùng để tăng tốc độ tính toán của RNN so với CNN
- (D). Dùng để chuẩn hóa dữ liệu đầu vào

36.3 Phụ Lục

1. **Hint:** Các file code template có thể tải tại [Link](#)
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể tải tại [Link](#) (**Lưu ý** Sáng thứ 3 khi hết deadline phần bài tập ad mới copy các nội dung bài giải nêu trên vào đường dẫn)
3. **Rubric:**

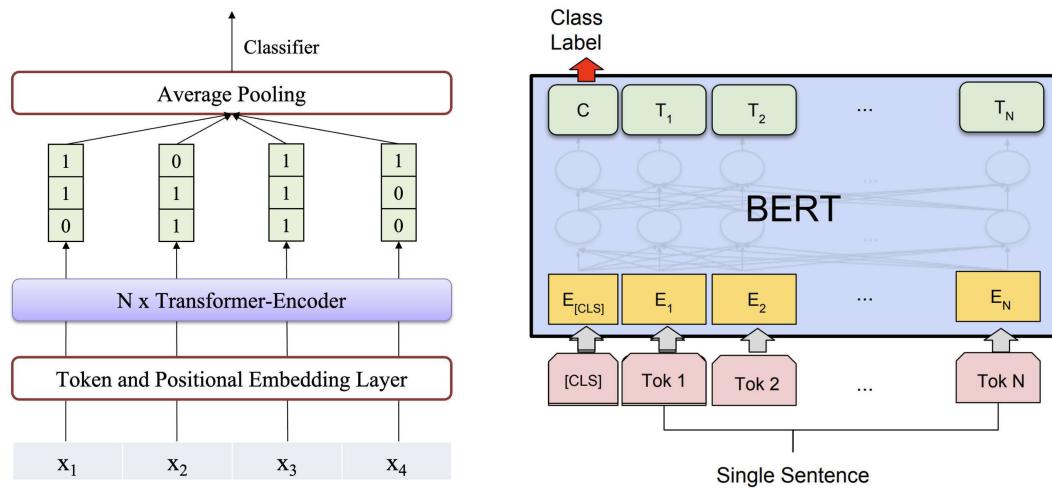
Recurrent Neural Networks Exercise - Rubric		
Câu	Kiến Thức	Đánh Giá
II.A. 1, 2	<ul style="list-style-type: none"> - Kiến thức về lập trình pytorch. - Quy trình cơ bản trong việc xây dựng mô hình mạng nơ-ron sử dụng các layer thuộc họ RNN. - Cách xử lý dữ liệu văn bản và dữ liệu time series. 	<ul style="list-style-type: none"> - Biết cách ứng dụng kỹ thuật lập trình pytorch để xây dựng một mô hình phân loại văn bản và mô hình time series.
II.B. 1, 2, 3	<ul style="list-style-type: none"> - Kiến thức cơ bản về RNN và LSTM. 	<ul style="list-style-type: none"> - Hiểu được các tính chất cơ bản hai mô hình RNN và LSTM.
II.B. 4, 5, 6	<ul style="list-style-type: none"> - Kiến thức cơ bản về RNN và LSTM. 	<ul style="list-style-type: none"> - Hiểu được các tính chất cơ bản hai mô hình RNN và LSTM.
II.B. 7, 8, 9	<ul style="list-style-type: none"> - Công thức tính toán hidden state, output và backpropagation trong RNN. 	<ul style="list-style-type: none"> - Hiểu chi tiết các công thức trong RNN.

- *Hết* -

Chương 37

Mô hình Transformer

37.1 Transformer



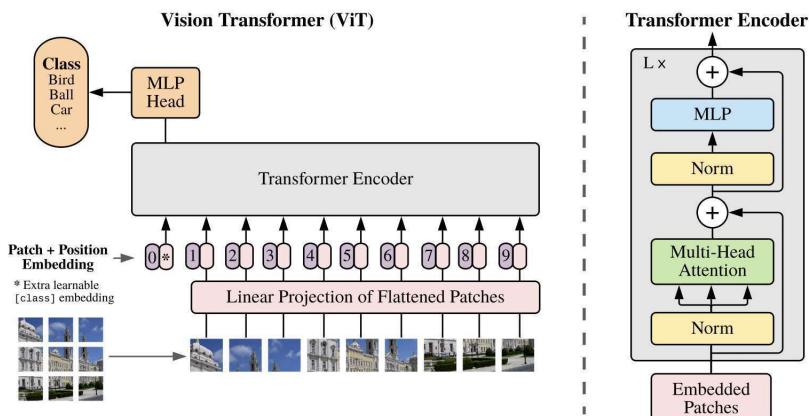
Hình 37.1: Mô hình Transformer-Encoder và BERT cho bài toán phân loại văn bản.

Mô hình Transformer ra đời với kỹ thuật cốt lõi và cơ chế Attention, đã đạt được những kết quả ấn tượng cho các bài toán về dữ liệu văn bản rồi từ đó mở rộng cho các kiểu dữ liệu như hình ảnh và âm thanh,... Trong phần này, chúng ta sẽ tìm hiểu các thành phần trong mô hình Transformer và ứng dụng cho bài toán phân loại văn bản.

37.1.1 Kiến trúc Transformer

Kiến trúc Transformer gồm các thành phần:

- Input Embedding: Biểu diễn các token đầu vào (Thường được tách bởi Subword-based Tokenization) thành các dense vector.



Hình 37.2: Mô hình Vision Transformer cho bài toán phân loại hình ảnh.

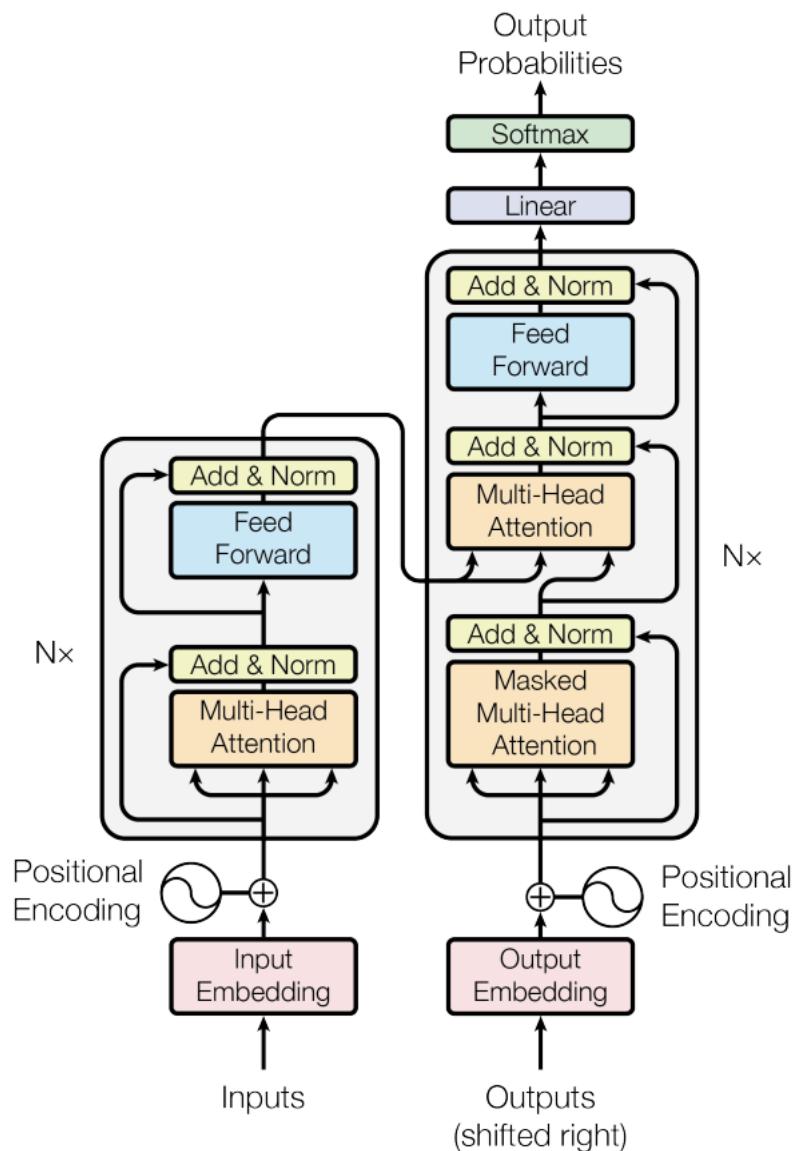
- Positional Encoding: Biểu diễn vị trí (thứ tự) của các token trong câu. Thường được tính dựa vào hàm sinusoid hoặc được học trong quá trình huấn luyện mô hình.
- Các khối encoder: Để mã hoá các tokens đầu vào thành các contextual embedding. Bao gồm: Multi-Head Attention, Add - Normalization, Feed Forward
- Các khối decoder: nhận input là các token lịch sử và trạng thái mã hoá từ encoder, giải mã dự đoán token tiếp theo. Gồm: Masked Multi-Head Attention (dựa vào token lịch sử của decoder), Multi-Head Attention (dựa vào encoder và trạng thái hiện tại decoder), Add - Normalization, Feed Forward
- Language Model Head: Projection và Softmax dự đoán token tiếp theo vs xác suất lớn nhất.

1. Input Embedding, Positional Encoding

```

1 class TokenAndPositionEmbedding(nn.Module):
2     def __init__(self, vocab_size, embed_dim, max_length,
3                  device='cpu'):
4         super().__init__()
5         self.device = device
6         self.word_emb = nn.Embedding(
            num_embeddings=vocab_size,

```



Hình 37.3: Mô hình kiến trúc Transformer.

```

7         embedding_dim=embed_dim
8     )
9     self.pos_emb = nn.Embedding(
10        num_embeddings=max_length,
11        embedding_dim=embed_dim

```

```

12
13
14     def forward(self, x):
15         N, seq_len = x.size()
16         positions = torch.arange(0, seq_len).expand(N,
17             seq_len).to(self.device)
18         output1 = self.word_emb(x)
19         output2 = self.pos_emb(positions)
20         output = output1 + output2
21         return output

```

2. Encoder

```

1 class TransformerEncoderBlock(nn.Module):
2     def __init__(self, embed_dim, num_heads, ff_dim, dropout=0.1):
3         super().__init__()
4         self.attn = nn.MultiheadAttention(
5             embed_dim=embed_dim,
6             num_heads=num_heads,
7             batch_first=True
8         )
9         self.ffn = nn.Sequential(
10             nn.Linear(in_features=embed_dim, out_features=
11                 ff_dim, bias=True),
12             nn.ReLU(),
13             nn.Linear(in_features=ff_dim, out_features=
14                 embed_dim, bias=True)
15         )
16         self.layernorm_1 = nn.LayerNorm(normalized_shape=
17             embed_dim, eps=1e-6)
18         self.layernorm_2 = nn.LayerNorm(normalized_shape=
19             embed_dim, eps=1e-6)
20         self.dropout_1 = nn.Dropout(p=dropout)
21         self.dropout_2 = nn.Dropout(p=dropout)
22
23     def forward(self, query, key, value):
24         attn_output, _ = self.attn(query, key, value)
25         attn_output = self.dropout_1(attn_output)
26         out_1 = self.layernorm_1(query + attn_output)
27         ffn_output = self.ffn(out_1)
28         ffn_output = self.dropout_2(ffn_output)
29         out_2 = self.layernorm_2(out_1 + ffn_output)
30         return out_2
31
32 class TransformerEncoder(nn.Module):

```

```

29     def __init__(self,
30                  src_vocab_size, embed_dim, max_length,
31                  num_layers, num_heads, ff_dim,
32                  dropout=0.1, device='cpu'
33                  ):
34         super().__init__()
35         self.embedding = TokenAndPositionEmbedding(
36             src_vocab_size, embed_dim, max_length, device
37         )
38         self.layers = nn.ModuleList(
39             [
40                 TransformerEncoderBlock(
41                     embed_dim, num_heads, ff_dim, dropout
42                     ) for i in range(num_layers)
43             ]
44         )
45
46     def forward(self, x):
47         output = self.embedding(x)
48         for layer in self.layers:
49             output = layer(output, output, output)
50
51     return output

```

3. Decoder

```

1 class TransformerDecoderBlock(nn.Module):
2     def __init__(self, embed_dim, num_heads, ff_dim, dropout
3 =0.1):
4         super().__init__()
5         self.attn = nn.MultiheadAttention(
6             embed_dim=embed_dim,
7             num_heads=num_heads,
8             batch_first=True
9         )
10        self.cross_attn = nn.MultiheadAttention(
11            embed_dim=embed_dim,
12            num_heads=num_heads,
13            batch_first=True
14        )
15        self.ffn = nn.Sequential(
16            nn.Linear(in_features=embed_dim, out_features=
17 ff_dim, bias=True),
18            nn.ReLU(),
19            nn.Linear(in_features=ff_dim, out_features=
20 embed_dim, bias=True)
21        )

```

```
19         self.layernorm_1 = nn.LayerNorm(normalized_shape=
20             embed_dim, eps=1e-6)
21         self.layernorm_2 = nn.LayerNorm(normalized_shape=
22             embed_dim, eps=1e-6)
23         self.layernorm_3 = nn.LayerNorm(normalized_shape=
24             embed_dim, eps=1e-6)
25         self.dropout_1 = nn.Dropout(p=dropout)
26         self.dropout_2 = nn.Dropout(p=dropout)
27         self.dropout_3 = nn.Dropout(p=dropout)
28
29     def forward(self, x, enc_output, src_mask, tgt_mask):
30         attn_output, _ = self.attn(x, x, x, attn_mask=
31             tgt_mask)
32         attn_output = self.dropout_1(attn_output)
33         out_1 = self.layernorm_1(x + attn_output)
34
35         attn_output, _ = self.cross_attn(
36             out_1, enc_output, enc_output, attn_mask=src_mask
37         )
38         attn_output = self.dropout_2(attn_output)
39         out_2 = self.layernorm_2(out_1 + attn_output)
40
41         ffn_output = self.ffn(out_2)
42         ffn_output = self.dropout_3(ffn_output)
43         out_3 = self.layernorm_3(out_2 + ffn_output)
44
45     return out_3
46
47 class TransformerDecoder(nn.Module):
48     def __init__(self,
49                  tgt_vocab_size, embed_dim, max_length, num_layers,
50                  num_heads, ff_dim,
51                  dropout=0.1, device='cpu',
52                  ):
53         super().__init__()
54         self.embedding = TokenAndPositionEmbedding(
55             tgt_vocab_size, embed_dim, max_length, device
56         )
57         self.layers = nn.ModuleList(
58             [
59                 TransformerDecoderBlock(
60                     embed_dim, num_heads, ff_dim, dropout
61                 ) for i in range(num_layers)
62             ]
63         )
64
```

```

59     def forward(self, x, enc_output, src_mask, tgt_mask):
60         output = self.embedding(x)
61         for layer in self.layers:
62             output = layer(output, enc_output, src_mask,
63                           tgt_mask)
64         return output

```

4. Transformer

```

1 class Transformer(nn.Module):
2     def __init__(self,
3                  src_vocab_size, tgt_vocab_size,
4                  embed_dim, max_length, num_layers, num_heads,
5                  ff_dim,
6                  dropout=0.1, device='cpu'
7                  ):
8         super().__init__()
9         self.device = device
10        self.encoder = TransformerEncoder(
11            src_vocab_size, embed_dim, max_length, num_layers
12            , num_heads, ff_dim
13            )
14        self.decoder = TransformerDecoder(
15            tgt_vocab_size, embed_dim, max_length, num_layers
16            , num_heads, ff_dim
17            )
18        self.fc = nn.Linear(embed_dim, tgt_vocab_size)
19
20    def generate_mask(self, src, tgt):
21        src_seq_len = src.shape[1]
22        tgt_seq_len = tgt.shape[1]
23
24        src_mask = torch.zeros(
25            (src_seq_len, src_seq_len),
26            device=self.device).type(torch.bool)
27
28        tgt_mask = (torch.triu(torch.ones(
29            (tgt_seq_len, tgt_seq_len),
30            device=self.device)
31            ) == 1).transpose(0, 1)
32        tgt_mask = tgt_mask.float().masked_fill(
33            tgt_mask == 0, float('-inf')).masked_fill(
34            tgt_mask == 1, float(0.0))
35        return src_mask, tgt_mask
36
37    def forward(self, src, tgt):
38

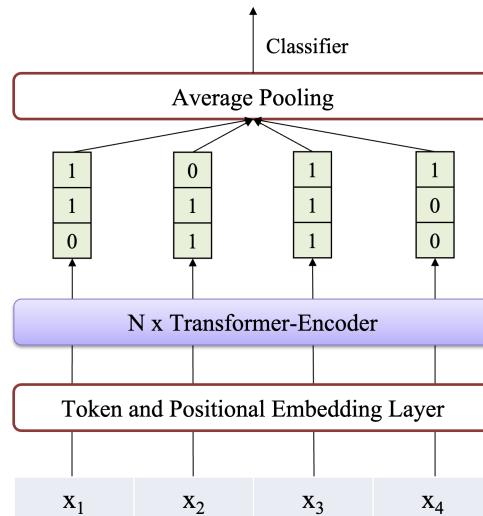
```

```
34         src_mask, tgt_mask = self.generate_mask(src, tgt)
35         enc_output = self.encoder(src)
36         dec_output = self.decoder(tgt, enc_output, src_mask,
37             tgt_mask)
38         output = self.fc(dec_output)
39         return output
```

5. Thử nghiệm

```
1 batch_size = 128
2 src_vocab_size = 1000
3 tgt_vocab_size = 2000
4 embed_dim = 200
5 max_length = 100
6 num_layers = 2
7 num_heads = 4
8 ff_dim = 256
9
10 model = Transformer(
11     src_vocab_size, tgt_vocab_size,
12     embed_dim, max_length, num_layers, num_heads, ff_dim
13 )
14
15 src = torch.randint(
16     high=2,
17     size=(batch_size, max_length),
18     dtype=torch.int64
19 )
20
21 tgt = torch.randint(
22     high=2,
23     size=(batch_size, max_length),
24     dtype=torch.int64
25 )
26
27 prediction = model(src, tgt)
28 prediction.shape # batch_size x max_length x tgt_vocab_size
```

37.1.2 Text Classification



Hình 37.4: Phân loại văn bản sử dụng Transformer-Encoder

Ở phần này, chúng ta sử dụng mô hình Transformer-Encoder cho bài toán phân loại văn bản. Kiến trúc mô hình gồm các phần:

- Input Embedding và Positional Encoding
- Các lớp Encoder
- Lớp Average Pooling: lấy trung bình biểu diễn các từ thành biểu diễn cho câu
- Classifier: Linear layer

1. Load Dataset

```

1 !pip install datasets
2
3 from datasets import load_dataset
4
5 ds = load_dataset('thainq107/ntc-scv')

```

2. Preprocessing

Áp dụng hàm tiền xử lý sau trên cột ‘sentence’ hoặc có thể bỏ qua bước tiền xử lý khi áp dụng trên cột ‘preprocessed_sentence’.

```
1 import re
2 import string
3
4 def preprocess_text(text):
5     # remove URLs https://www.
6     url_pattern = re.compile(r'https?://\s+\www\.\s+')
7     text = url_pattern.sub(r" ", text)
8
9     # remove HTML Tags: <>
10    html_pattern = re.compile(r'<[^<>]+>')
11    text = html_pattern.sub(" ", text)
12
13    # remove puncs and digits
14    replace_chars = list(string.punctuation + string.digits)
15    for char in replace_chars:
16        text = text.replace(char, " ")
17
18    # remove emoji
19    emoji_pattern = re.compile("["
20        u"\U0001F600-\U0001F64F" # emoticons
21        u"\U0001F300-\U0001F5FF" # symbols & pictographs
22        u"\U0001F680-\U0001F6FF" # transport & map symbols
23        u"\U0001F1E0-\U0001F1FF" # flags (iOS)
24        u"\U0001F1F2-\U0001F1F4" # Macau flag
25        u"\U0001F1E6-\U0001F1FF" # flags
26        u"\U0001F600-\U0001F64F"
27        u"\U00002702-\U000027B0"
28        u"\U000024C2-\U0001F251"
29        u"\U0001f926-\U0001f937"
30        u"\U0001F1F2"
31        u"\U0001F1F4"
32        u"\U0001F620"
33        u"\u200d"
34        u"\u2640-\u2642"
35        "+", flags=re.UNICODE)
36    text = emoji_pattern.sub(r" ", text)
37
38    # normalize whitespace
39    text = " ".join(text.split())
40
41    # lowercasing
42    text = text.lower()
43    return text
```

3. Representation

```
1 # !pip install -q torchtext==0.17.2 (install before import
2 #           torch)
3
4 def yield_tokens(sentences, tokenizer):
5     for sentence in sentences:
6         yield tokenizer(sentence)
7
8 # word-based tokenizer
9 from torchtext.data import get_tokenizer
10 tokenizer = get_tokenizer("basic_english")
11
12 # build vocabulary
13 from torchtext.vocab import build_vocab_from_iterator
14
15 vocab_size = 10000
16 vocabulary = build_vocab_from_iterator(
17     yield_tokens(ds['train']['preprocessed_sentence'],
18                 tokenizer),
19     max_tokens=vocab_size,
20     specials=["<pad>", "<unk>"]
21 )
22 vocabulary.set_default_index(vocabulary["<unk>"])
23
24 # convert torchtext dataset
25 from torchtext.data.functional import to_map_style_dataset
26
27 def prepare_dataset(df):
28     # create iterator for dataset: (sentence, label)
29     for row in df:
30         sentence = row['preprocessed_sentence']
31         encoded_sentence = vocabulary(tokenizer(sentence))
32         label = row['label']
33         yield encoded_sentence, label
34
35 train_dataset = prepare_dataset(ds['train'])
36 train_dataset = to_map_style_dataset(train_dataset)
37
38 valid_dataset = prepare_dataset(ds['valid'])
39 valid_dataset = to_map_style_dataset(valid_dataset)
40
41 test_dataset = prepare_dataset(ds['test'])
42 test_dataset = to_map_style_dataset(test_dataset)
```

4. Dataloader

```
1 import torch
2
3 seq_length = 100
4
5 def collate_batch(batch):
6     # create inputs, offsets, labels for batch
7     sentences, labels = list(zip(*batch))
8     encoded_sentences = [
9         sentence + ([0] * (seq_length - len(sentence))) if len(
10            sentence) < seq_length else sentence[:seq_length]
11            for sentence in sentences
12        ]
13
14     encoded_sentences = torch.tensor(encoded_sentences, dtype=
15         torch.int64)
16     labels = torch.tensor(labels)
17
18     return encoded_sentences, labels
19
20 batch_size = 128
21
22 train_dataloader = DataLoader(
23     train_dataset,
24     batch_size=batch_size,
25     shuffle=True,
26     collate_fn=collate_batch
27 )
28 valid_dataloader = DataLoader(
29     valid_dataset,
30     batch_size=batch_size,
31     shuffle=False,
32     collate_fn=collate_batch
33 )
34
35 test_dataloader = DataLoader(
36     test_dataset,
37     batch_size=batch_size,
38     shuffle=False,
39     collate_fn=collate_batch
40 )
```

5. Trainer

```
1 # train epoch
2 import time
3
4 def train_epoch(model, optimizer, criterion, train_dataloader,
5                 device, epoch=0, log_interval=50):
6     model.train()
7     total_acc, total_count = 0, 0
8     losses = []
9     start_time = time.time()
10
11    for idx, (inputs, labels) in enumerate(train_dataloader):
12        inputs = inputs.to(device)
13        labels = labels.to(device)
14
15        optimizer.zero_grad()
16
17        predictions = model(inputs)
18
19        # compute loss
20        loss = criterion(predictions, labels)
21        losses.append(loss.item())
22
23        # backward
24        loss.backward()
25        optimizer.step()
26        total_acc += (predictions.argmax(1) == labels).sum().item()
27        total_count += labels.size(0)
28        if idx % log_interval == 0 and idx > 0:
29            elapsed = time.time() - start_time
30            print(
31                "| epoch {:3d} | {:5d}/{:5d} batches "
32                "| accuracy {:.8.3f}".format(
33                    epoch, idx, len(train_dataloader),
34                    total_acc / total_count
35                )
36            )
37            total_acc, total_count = 0, 0
38            start_time = time.time()
39
40    epoch_acc = total_acc / total_count
41    epoch_loss = sum(losses) / len(losses)
42    return epoch_acc, epoch_loss
43
44 # evaluate
```

```
43 def evaluate_epoch(model, criterion, valid_dataloader, device):
44     model.eval()
45     total_acc, total_count = 0, 0
46     losses = []
47
48     with torch.no_grad():
49         for idx, (inputs, labels) in enumerate(
50             valid_dataloader):
51             inputs = inputs.to(device)
52             labels = labels.to(device)
53
54             predictions = model(inputs)
55
56             loss = criterion(predictions, labels)
57             losses.append(loss.item())
58
59             total_acc += (predictions.argmax(1) == labels).
60             sum().item()
61             total_count += labels.size(0)
62
63     epoch_acc = total_acc / total_count
64     epoch_loss = sum(losses) / len(losses)
65     return epoch_acc, epoch_loss
66
67 # train
68 def train(model, model_name, save_model, optimizer, criterion,
69           train_dataloader, valid_dataloader, num_epochs, device):
70     train_accs, train_losses = [], []
71     eval_accs, eval_losses = [], []
72     best_loss_eval = 100
73     times = []
74
75     for epoch in range(1, num_epochs+1):
76         epoch_start_time = time.time()
77         # Training
78         train_acc, train_loss = train_epoch(model, optimizer,
79             criterion, train_dataloader, device, epoch)
80         train_accs.append(train_acc)
81         train_losses.append(train_loss)
82
83         # Evaluation
84         eval_acc, eval_loss = evaluate_epoch(model, criterion,
85             valid_dataloader, device)
86         eval_accs.append(eval_acc)
```

```
82         eval_losses.append(eval_loss)
83
84     # Save best model
85     if eval_loss < best_loss_eval:
86         torch.save(model.state_dict(), save_model + f'{model_name}.pt')
87
88     times.append(time.time() - epoch_start_time)
89     # Print loss, acc end epoch
90     print("-" * 59)
91     print(
92         "| End of epoch {:3d} | Time: {:.2f}s | Train"
93         "Accuracy {:.3f} | Train Loss {:.3f} "
94         "| Valid Accuracy {:.3f} | Valid Loss {:.3f} ".
95         format(
96             epoch, time.time() - epoch_start_time,
97             train_acc, train_loss, eval_acc, eval_loss
98         )
99     )
100    print("-" * 59)
101
102    # Load best model
103    model.load_state_dict(torch.load(save_model + f'{model_name}.pt'))
104    model.eval()
105    metrics = {
106        'train_accuracy': train_accs,
107        'train_loss': train_losses,
108        'valid_accuracy': eval_accs,
109        'valid_loss': eval_losses,
110        'time': times
111    }
112    return model, metrics
113
114 # report
115 import matplotlib.pyplot as plt
116
117 def plot_result(num_epochs, train_accs, eval_accs,
118                 train_losses, eval_losses):
119     epochs = list(range(num_epochs))
120     fig, axs = plt.subplots(nrows = 1, ncols = 2 , figsize =
121                           (12,6))
122     axs[0].plot(epochs, train_accs, label = "Training")
123     axs[0].plot(epochs, eval_accs, label = "Evaluation")
124     axs[1].plot(epochs, train_losses, label = "Training")
```

```
120     axs[1].plot(epochs, eval_losses, label = "Evaluation")
121     axs[0].set_xlabel("Epochs")
122     axs[1].set_xlabel("Epochs")
123     axs[0].set_ylabel("Accuracy")
124     axs[1].set_ylabel("Loss")
125     plt.legend()
```

6. Modeling

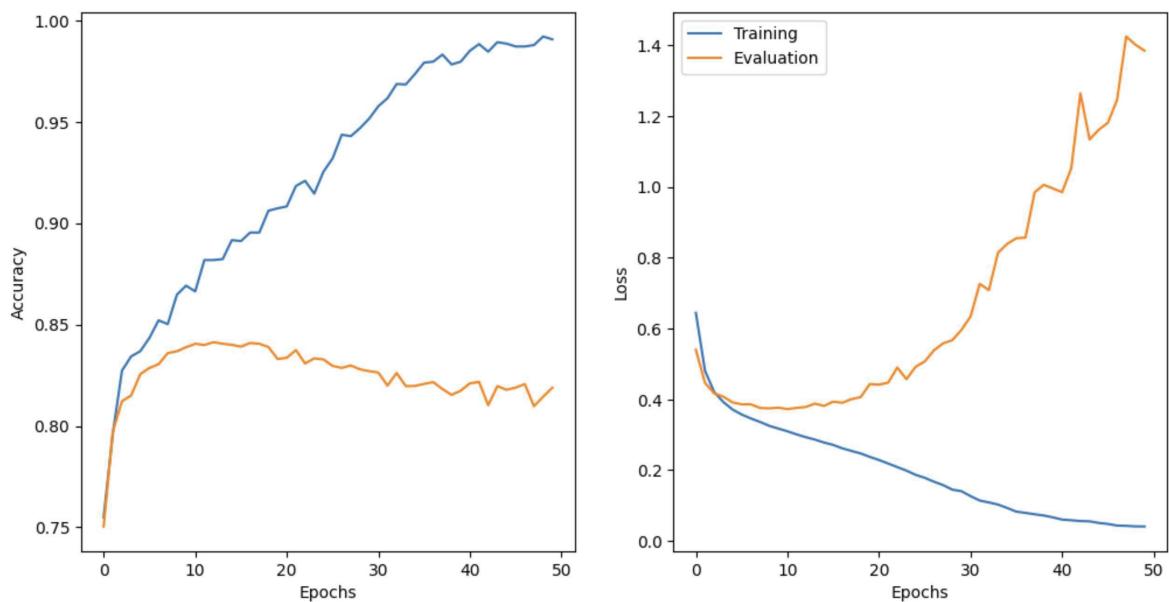
```
1 class TransformerEncoderCls(nn.Module):
2     def __init__(self,
3                  vocab_size, max_length, num_layers,
4                  embed_dim, num_heads, ff_dim,
5                  dropout=0.1, device='cpu'):
6         super().__init__()
7         self.encoder = TransformerEncoder(
8             vocab_size, embed_dim, max_length, num_layers,
9             num_heads, ff_dim, dropout, device
10            )
11         self.pooling = nn.AvgPool1d(kernel_size=max_length)
12         self.fc1 = nn.Linear(in_features=embed_dim,
13                             out_features=20)
14         self.fc2 = nn.Linear(in_features=20, out_features=2)
15         self.dropout = nn.Dropout(p=dropout)
16         self.relu = nn.ReLU()
17     def forward(self, x):
18         output = self.encoder(x)
19         output = self.pooling(output.permute(0, 2, 1)).squeeze
20         ()
21         output = self.dropout(output)
22         output = self.fc1(output)
23         output = self.dropout(output)
24         output = self.fc2(output)
25         return output
```

7. Training

```
1 import torch.optim as optim
2
3 vocab_size = 10000
4 max_length = 100
5 embed_dim = 200
6 num_layers = 2
7 num_heads = 4
8 ff_dim = 128
9 dropout=0.1
```

```
10
11 model = TransformerEncoderCls(
12     vocab_size, max_length, num_layers, embed_dim, num_heads,
13     ff_dim, dropout
14 )
15 device = torch.device('cuda' if torch.cuda.is_available()
16                         else 'cpu')
17 model = TransformerEncoderCls(
18     vocab_size, max_length, num_layers, embed_dim, num_heads,
19     ff_dim, dropout, device
20 )
21 model.to(device)
22
23 criterion = torch.nn.CrossEntropyLoss()
24 optimizer = optim.Adam(model.parameters(), lr=0.00005)
25
26 num_epochs = 50
27 save_model = './model'
28 os.makedirs(save_model, exist_ok = True)
29 model_name = 'model'
30
31 model, metrics = train(
32     model, model_name, save_model, optimizer, criterion,
33     train_dataloader, valid_dataloader, num_epochs, device
34 )
```

Kết quả training và accuracy trên tập test là 82%



Hình 37.5: Quá trình huấn luyện bộ dữ liệu NTC-SCV với mô hình Transformer-Encoder.

37.2 Text classification using BERT

Một trong những mô hình pretrained đầu tiên cho dữ liệu văn bản dựa vào kiến trúc mô hình Transformer được ứng dụng cho các downstream task khác nhau đó là BERT. Trong phần này chúng ta sẽ fine tuning BERT cho bài toán phân loại trên bộ dữ liệu NTC-SCV dựa vào thư viện transformers của huggingface.

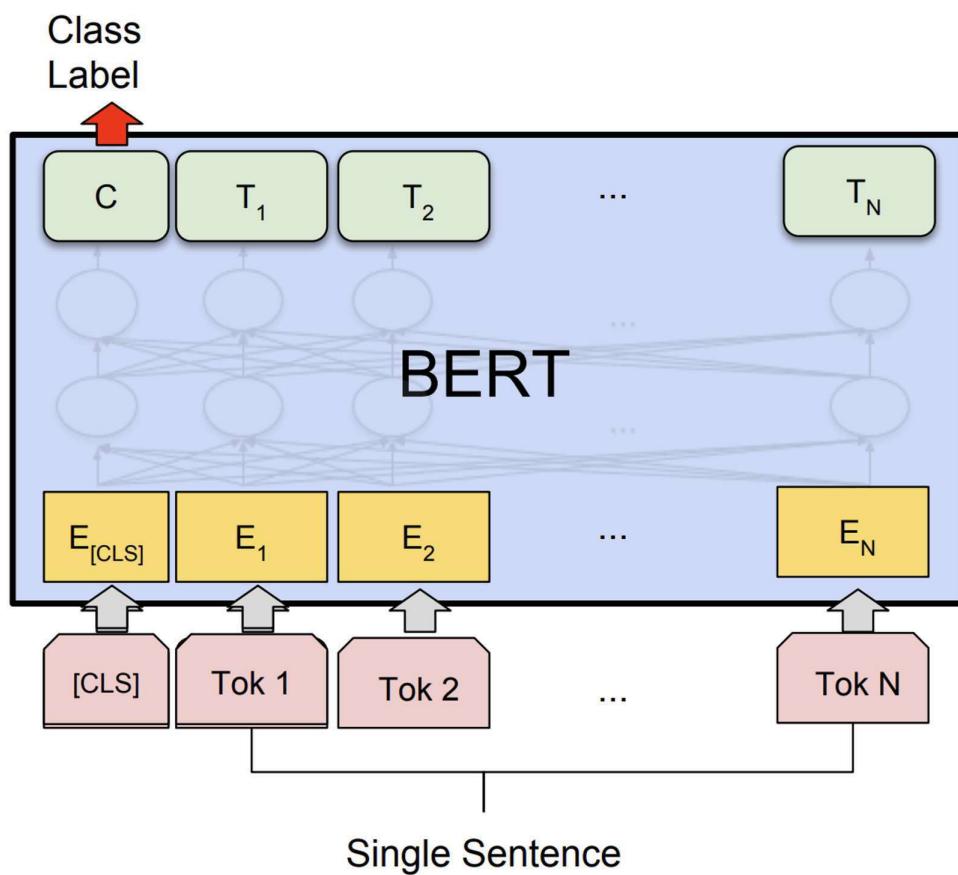
1. Load Dataset

```

1 # install libs
2 !pip install -q -U transformers datasets accelerate evaluate
3
4 from datasets import load_dataset
5
6 ds = load_dataset('thainq107/ntc-scv')

```

2. Preprocessing



Hình 37.6: Fine-Tuning BERT cho bài toán phân loại văn bản.

```

1 # tokenization
2 from transformers import AutoTokenizer
3
4 model_name = "distilbert-base-uncased" # bert-base-uncased
5
6 tokenizer = AutoTokenizer.from_pretrained(
7     model_name,
8     use_fast=True
9 )
10 max_seq_length = 100
11 max_seq_length = min(max_seq_length, tokenizer.
12                         model_max_length)
13 def preprocess_function(examples):

```

```

14     # Tokenize the texts
15
16     result = tokenizer(
17         examples["preprocessed_sentence"] ,
18         padding="max_length",
19         max_length=max_seq_length ,
20         truncation=True
21     )
22     result["label"] = examples['label']
23
24     return result
25
26 # Running the preprocessing pipeline on all the datasets
27 processed_dataset = ds.map(
28     preprocess_function ,
29     batched=True ,
30     desc="Running tokenizer on dataset",
31 )

```

3. Modeling

```

1 from transformers import AutoConfig ,
2     AutoModelForSequenceClassification
3
4 num_labels = 2
5
6 config = AutoConfig.from_pretrained(
7     model_name ,
8     num_labels=num_labels ,
9     finetuning_task="text-classification"
10)
11
12 model = AutoModelForSequenceClassification.from_pretrained(
13     model_name ,
14     config=config
15 )

```

4. Metric

```

1 import numpy as np
2 import evaluate
3
4 metric = evaluate.load("accuracy")
5 def compute_metrics(eval_pred):
6     predictions, labels = eval_pred

```

```
7     predictions = np.argmax(predictions, axis=1)
8     result = metric.compute(predictions=predictions,
9     references=labels)
10    return result
```

5. Trainer

```
1 from transformers import TrainingArguments, Trainer
2
3 training_args = TrainingArguments(
4     output_dir="save_model",
5     learning_rate=2e-5,
6     per_device_train_batch_size=128,
7     per_device_eval_batch_size=128,
8     num_train_epochs=10,
9     eval_strategy="epoch",
10    save_strategy="epoch",
11    load_best_model_at_end=True
12 )
13
14 trainer = Trainer(
15     model=model,
16     args=training_args,
17     train_dataset=processed_dataset["train"],
18     eval_dataset=processed_dataset["valid"],
19     compute_metrics=compute_metrics,
20     tokenizer=tokenizer,
21 )
22
23 trainer.train()
```

6. Training

Kết quả training và accuracy trên tập test là 85%

7. Triển khai mô hình

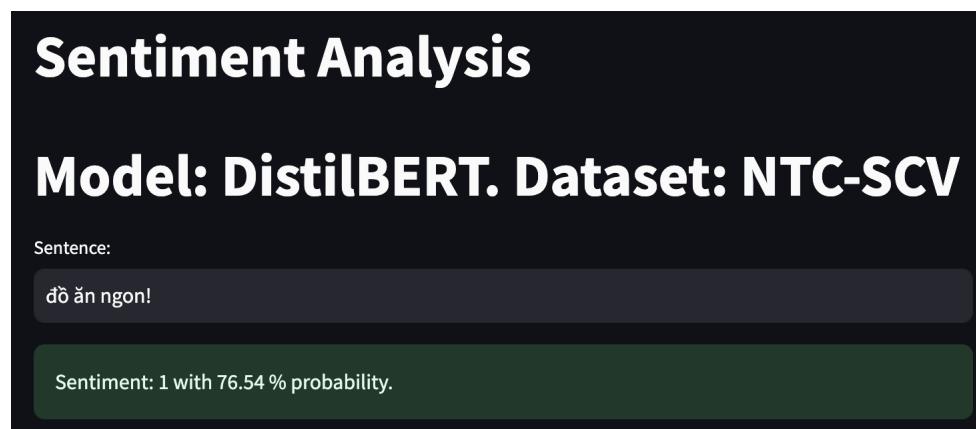
Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:

1. [Link Streamlit](#)
2. [Github](#)

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.457404	0.790200
2	No log	0.432181	0.804500
3	No log	0.419964	0.812100
4	No log	0.395226	0.827100
5	0.431800	0.398116	0.831200
6	0.431800	0.388693	0.834300
7	0.431800	0.403755	0.831600
8	0.431800	0.408512	0.833300
9	0.316800	0.409772	0.835000
10	0.316800	0.410361	0.836200

Hình 37.7: Quá trình huấn luyện bộ dữ liệu NTC-SCV dựa vào fine tuning BERT.

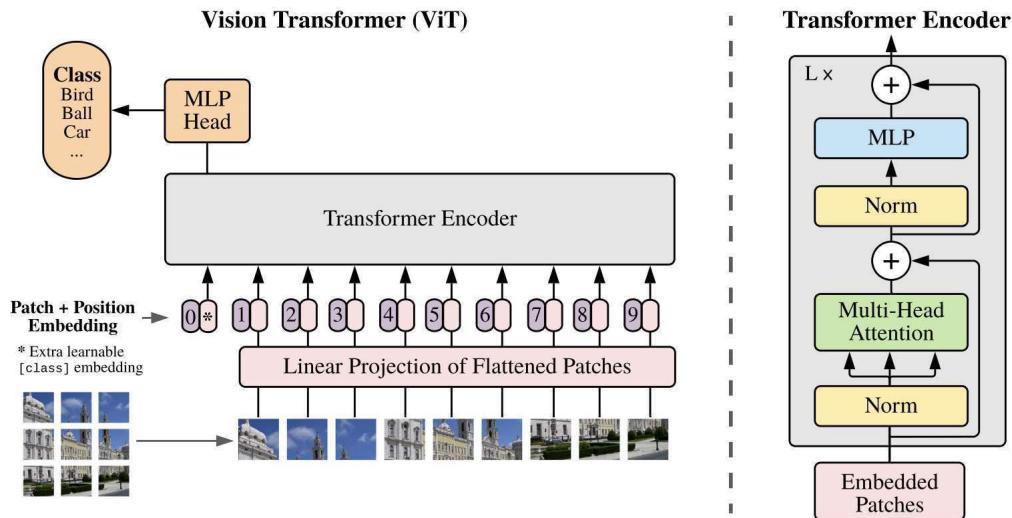
3. Giao diện:



Hình 37.8: Giao diện ứng dụng.

37.3 Vision Transformer

Trong phần này, chúng ta sẽ huấn luyện mô hình Vision Transformer cho bài toán phân loại hình ảnh.



Hình 37.9: Mô hình Vision Transformer cho bài toán phân loại hình ảnh.

1. Load Dataset

```

1 import torch
2 import torchvision.transforms as transforms
3 from torch.utils.data import DataLoader, random_split
4 import torch.optim as optim
5 from torchvision.datasets import ImageFolder
6 from torch import nn
7 import math
8 import os

1 # download
2 !gdown 1vSevps_hV5zhVf6aWuN8X7dd-qSAIgcc
3 !unzip ./flower_photos.zip
4
5 # load data
6 data_patch = "./flower_photos"
7 dataset = ImageFolder(root=data_patch)

```

```
8 num_samples = len(dataset)
9 classes = dataset.classes
10 num_classes = len(dataset.classes)
11
12 # split
13 TRAIN_RATIO, VALID_RATIO = 0.8, 0.1
14 n_train_examples = int(num_samples * TRAIN_RATIO)
15 n_valid_examples = int(num_samples * VALID_RATIO)
16 n_test_examples = num_samples - n_train_examples -
    n_valid_examples
17 train_dataset, valid_dataset, test_dataset = random_split(
18     dataset,
19     [n_train_examples, n_valid_examples, n_test_examples]
20 )
```

2. Preprocessing

```
1 # resize + convert to tensor
2 IMG_SIZE = 224
3
4 train_transforms = transforms.Compose([
5     transforms.Resize((IMG_SIZE, IMG_SIZE)),
6     transforms.RandomHorizontalFlip(),
7     transforms.RandomRotation(0.2),
8     transforms.ToTensor(),
9     transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
10 ])
11
12 test_transforms = transforms.Compose([
13     transforms.Resize((IMG_SIZE, IMG_SIZE)),
14     transforms.ToTensor(),
15     transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
16 ])
17
18 # apply
19 train_dataset.dataset.transform = train_transforms
20 valid_dataset.dataset.transform = test_transforms
21 test_dataset.dataset.transform = test_transforms
```

3. Dataloader

```
1 BATCH_SIZE = 512
2
3 train_loader = DataLoader(
```

```

4     train_dataset ,
5     shuffle=True ,
6     batch_size=BATCH_SIZE
7 )
8
9 val_loader = DataLoader(
10    valid_dataset ,
11    batch_size=BATCH_SIZE
12 )
13
14 test_loader = DataLoader(
15    test_dataset ,
16    batch_size=BATCH_SIZE
17 )

```

4. Training from Scratch

4.1. Modeling

```

1 class TransformerEncoder(nn.Module):
2     def __init__(self, embed_dim, num_heads, ff_dim, dropout
=0.1):
3         super().__init__()
4         self.attn = nn.MultiheadAttention(
5             embed_dim=embed_dim,
6             num_heads=num_heads,
7             batch_first=True
8         )
9         self.ffn = nn.Sequential(
10            nn.Linear(in_features=embed_dim, out_features=
ff_dim, bias=True),
11            nn.ReLU(),
12            nn.Linear(in_features=ff_dim, out_features=
embed_dim, bias=True)
13        )
14         self.layernorm_1 = nn.LayerNorm(normalized_shape=
embed_dim, eps=1e-6)
15         self.layernorm_2 = nn.LayerNorm(normalized_shape=
embed_dim, eps=1e-6)
16         self.dropout_1 = nn.Dropout(p=dropout)
17         self.dropout_2 = nn.Dropout(p=dropout)
18
19     def forward(self, query, key, value):
20         attn_output, _ = self.attn(query, key, value)
21         attn_output = self.dropout_1(attn_output)
22         out_1 = self.layernorm_1(query + attn_output)

```

```
23         ffn_output = self.ffn(out_1)
24         ffn_output = self.dropout_2(ffn_output)
25         out_2 = self.layernorm_2(out_1 + ffn_output)
26     return out_2

1 class PatchPositionEmbedding(nn.Module):
2     def __init__(self, image_size=224, embed_dim=512,
3                  patch_size=16, device='cpu'):
4         super().__init__()
5         self.conv1 = nn.Conv2d(in_channels=3, out_channels=
6                           embed_dim, kernel_size=patch_size, stride=patch_size, bias
7                           =False)
8         scale = embed_dim ** -0.5
9         self.positional_embedding = nn.Parameter(scale *
10            torch.randn((image_size // patch_size) ** 2, embed_dim))
11         self.device = device
12
13     def forward(self, x):
14         x = self.conv1(x) # shape = [* , width , grid , grid]
15         x = x.reshape(x.shape[0], x.shape[1], -1) # shape =
16         [* , width , grid ** 2]
17         x = x.permute(0, 2, 1) # shape = [* , grid ** 2 ,
18         width]
19
20         x = x + self.positional_embedding.to(self.device)
21     return x

1 class VisionTransformerCls(nn.Module):
2     def __init__(self,
3                  image_size, embed_dim, num_heads, ff_dim,
4                  dropout=0.1, device='cpu', num_classes = 10,
5                  patch_size=16
6                  ):
7         super().__init__()
8         self.embed_layer = PatchPositionEmbedding(
9             image_size=image_size, embed_dim=embed_dim,
10            patch_size=patch_size, device=device
11            )
12         self.transformer_layer = TransformerEncoder(
13             embed_dim, num_heads, ff_dim, dropout
14             )
15         # self.pooling = nn.AvgPool1d(kernel_size=max_length)
16         self.fc1 = nn.Linear(in_features=embed_dim,
17             out_features=20)
18         self.fc2 = nn.Linear(in_features=20, out_features=
19             num_classes)
```

```

16         self.dropout = nn.Dropout(p=dropout)
17         self.relu = nn.ReLU()
18     def forward(self, x):
19         output = self.embed_layer(x)
20         output = self.transformer_layer(output, output,
21                                         output)
22         output = output[:, 0, :]
23         output = self.dropout(output)
24         output = self.fc1(output)
25         output = self.dropout(output)
26         output = self.fc2(output)
27     return output

```

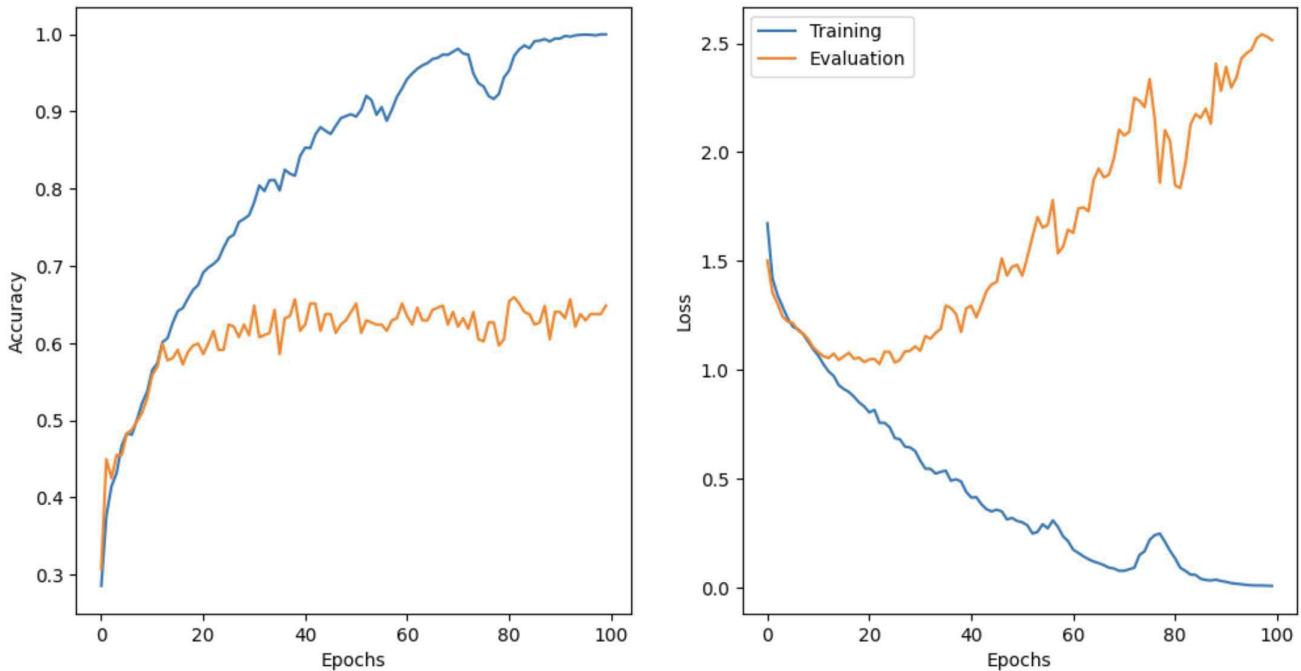
4.2. Training

```

1 image_size=224
2 embed_dim = 512
3 num_heads = 4
4 ff_dim = 128
5 dropout=0.1
6
7 device = torch.device('cuda' if torch.cuda.is_available()
8 else 'cpu')
8 model = VisionTransformerCls(
9     image_size=224, embed_dim=512, num_heads=num_heads,
10    ff_dim=ff_dim, dropout=dropout, num_classes=num_classes,
11    device=device
10 )
11 model.to(device)
12
13 criterion = torch.nn.CrossEntropyLoss()
14 optimizer = optim.Adam(model.parameters(), lr=0.0005)
15
16 num_epochs = 100
17 save_model = './vit_flowers'
18 os.makedirs(save_model, exist_ok = True)
19 model_name = 'vit_flowers'
20
21 model, metrics = train(
22     model, model_name, save_model, optimizer, criterion,
23     train_loader, val_loader, num_epochs, device
23 )

```

Kết quả training và accuracy trên tập test là 60%



Hình 37.10: Quá trình huấn luyện bộ dữ liệu Flower với mô hình Vision Transformer.

5. Fine Tuning

5.1. Modeling

```

1 from transformers import ViTForImageClassification
2
3 id2label = {id:label for id, label in enumerate(classes)}
4 label2id = {label:id for id,label in id2label.items()}
5
6 model = ViTForImageClassification.from_pretrained('google/vit'
7     '-base-patch16-224-in21k',
8     num_labels=
9     num_classes,
10    id2label,
11    label2id)
12 device = torch.device('cuda' if torch.cuda.is_available()
13 else 'cpu')

```

```
11 model.to(device)
```

5.2. Metric

```
1 import evaluate
2 import numpy as np
3
4 metric = evaluate.load("accuracy")
5
6 def compute_metrics(eval_pred):
7     predictions, labels = eval_pred
8     predictions = np.argmax(predictions, axis=1)
9     return metric.compute(predictions=predictions, references
10    =labels)
```

5.3. Trainer

```
1 import torch
2 from transformers import ViTImageProcessor
3 from transformers import TrainingArguments, Trainer
4
5 feature_extractor = ViTImageProcessor.from_pretrained("google
6 /vit-base-patch16-224-in21k")
7
8 metric_name = "accuracy"
9
10 args = TrainingArguments(
11     f"vit_flowers",
12     save_strategy="epoch",
13     evaluation_strategy="epoch",
14     learning_rate=2e-5,
15     per_device_train_batch_size=32,
16     per_device_eval_batch_size=32,
17     num_train_epochs=10,
18     weight_decay=0.01,
19     load_best_model_at_end=True,
20     metric_for_best_model=metric_name,
21     logging_dir='logs',
22     remove_unused_columns=False,
23 )
24
25 def collate_fn(examples):
26     # example => Tuple(image, label)
27     pixel_values = torch.stack([example[0] for example in
```

```
    examples])
27     labels = torch.tensor([example[1] for example in examples
28 ])
29
30 trainer = Trainer(
31     model,
32     args,
33     train_dataset=train_dataset,
34     eval_dataset=valid_dataset,
35     data_collator=collate_fn,
36     compute_metrics=compute_metrics,
37     tokenizer=feature_extractor,
38 )
```

5.4. Training

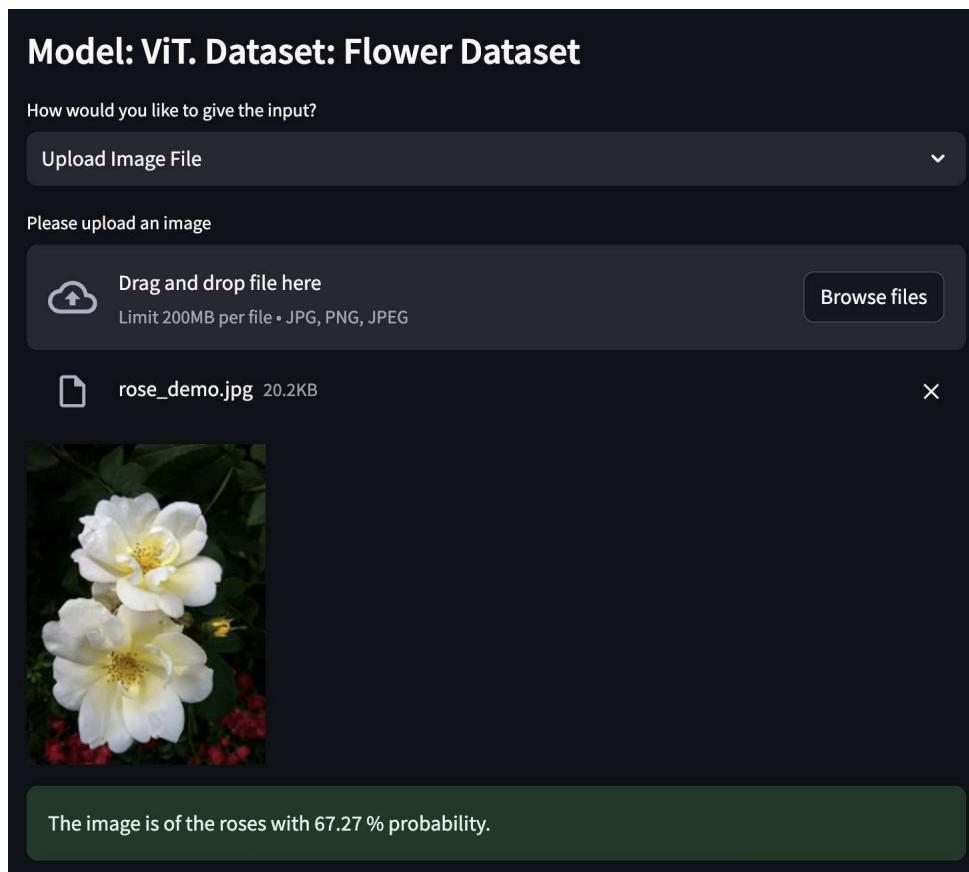
```
1 trainer.train()
2 outputs = trainer.predict(test_dataset)
3 outputs.metrics
```

Kết quả training và accuracy trên tập test là 97%

5.5. Triển khai mô hình

Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:

1. [Link Streamlit](#)
2. [Github](#)
3. Giao diện:



Hình 37.11: Giao diện ứng dụng.

37.4 Câu hỏi trắc nghiệm

Câu hỏi 33 Lớp Positional Encoding trong kiến trúc mô hình Transformer dùng để làm gì?

- a) Biểu diễn vị trí của các tokens
- b) Tính Attention
- c) Dự đoán token tiếp theo
- d) Lính Loss

Câu hỏi 34 Layer nào sau đây không có trong Transformer-Encoder?

- a) Masked Multi-Head Attention
- b) Multi-Head Attention
- c) Layer Normaliation
- d) Feed Forward

Câu hỏi 35 Layer nào sau đây không có trong Transformer-Decoder?

- a) Masked Multi-Head Attention
- b) Multi-Head Attention
- c) CNN
- d) Feed Forward

Câu hỏi 36 Masked Multi-Head Attention được sử dụng để làm gì?

- a) Mask những token lịch sử
- b) Mask vị trí các token lịch sử
- c) Mask những token trong tương lai
- d) Mask cả những token lịch sử và token trong tương lai

Câu hỏi 37 Phát biểu nào sau đây là đúng về BERT?

- a) Bao gồm các khối Transformer-Encoder
- b) Bao gồm các khối Transformer-Decoder
- c) Cả 2 đáp án a và b đều đúng
- d) Cả 2 đáp án a và b đều sai

Câu hỏi 38 Objective Function của BERT là?

- a) Masked Language Model
- b) Next Sentence Prediction
- c) Cả a và b đều đúng
- d) Cả a và b đều sai

Câu hỏi 39 Thành phần nào không có trong các giá trị Input của BERT

- a) Token Embeddings
- b) Seqment Embeddings
- c) Position Embeddings
- d) Language Model Head

Câu hỏi 40 Số lượng tham số của mô hình BERT-base và BERT-large là?

- a) 340M và 110M
- b) 110M và 340M
- c) 340M và 340M
- d) 110M và 340M

Câu hỏi 41 Các Patch trong Vision Transformer được xác định như thế nào?

- a) Flatten ảnh đầu vào
- b) Trung bình các giá trị điểm ảnh theo hàng
- c) Trung bình các giá trị điểm ảnh theo cột
- d) Chia ảnh đầu vào thành các Patch với kích thước các mảng cố định

Câu hỏi 42 Bộ dữ liệu nào sau đây được sử dụng trong phần thực nghiệm so sánh kết quả Vision Transformer với các phương pháp huấn luyện từ đầu và sử dụng pretrained?

- a) CIFAR10
- b) Flower
- c) CIFAR100
- d) MNIST

37.5 Phụ lục

1. **Hint:** Các file code gợi ý có thể được tải về tại đây:

- [Transformer-For-Text](#)
- [Transformer-For-Image](#)

2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

3. **Rubric:**

Phần	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> - Hiểu rõ kiến trúc mô hình Transformer - Hiểu rõ mô hình BERT - Áp dụng Transformer-Encoder và BERT cho bài toán phân loại văn bản 	<ul style="list-style-type: none"> - Xây dựng mô hình phân loại văn bản sử dụng Transformer-Encoder và BERT
2	<ul style="list-style-type: none"> - Hiểu rõ kiến trúc mô hình Vision Transformer 	<ul style="list-style-type: none"> - Phân loại hình ảnh sử dụng mô hình Vision Transformer và các mô hình tiền huấn luyện cho Vision Transformer

- *Hết* -

Chương 38

Project 1: OCR (Yolov9+CNN) để trích xuất thông tin ID Card

38.1 Giới thiệu

Nhận dạng Văn bản trong Ảnh (Scene Text Recognition) là một bài toán ứng dụng các kỹ thuật xử lý hình ảnh và nhận dạng chữ viết để xác định các văn bản xuất hiện trong các bức ảnh chụp từ môi trường thực tế. Bài toán này có nhiều ứng dụng thực tế, chẳng hạn như:

- Xử lý văn bản trong ảnh: Nhận diện chữ trong các loại tài liệu, báo chí, biển hiệu,...
- Tìm kiếm thông tin: Nhận diện chữ trong ảnh trên internet để thu thập dữ liệu quan trọng.
- Tự động hóa quy trình: Nhận diện chữ trong ảnh để tự động hóa các công việc, ví dụ như xử lý đơn hàng, thanh toán,...

Quy trình Nhận dạng Văn bản trong Ảnh điển hình gồm hai giai đoạn chính:

- Phát hiện chữ viết (Detector): Xác định vị trí các khối văn bản trong ảnh.
- Nhận diện chữ viết (Recognizer): Giải mã văn bản tại các vị trí đã được xác định.



Hình 38.1: Minh họa về bài toán Nhận dạng Văn bản trong Ảnh.

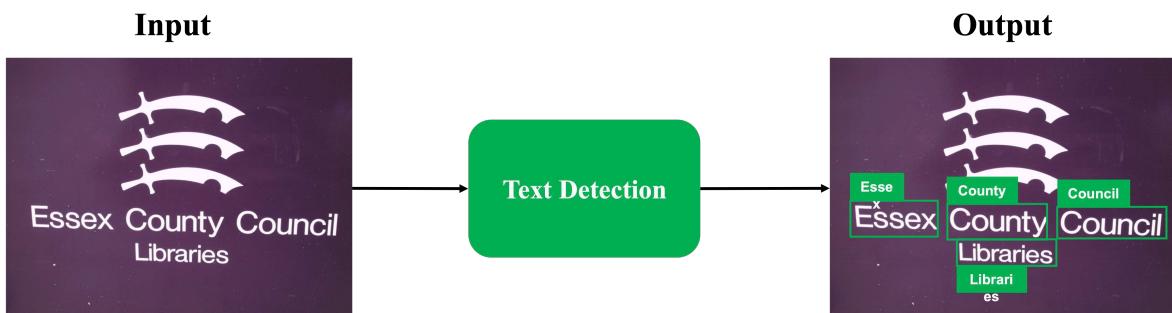
Trong dự án này, chúng ta sẽ phát triển một chương trình Nhận dạng Văn bản trong Ảnh sử dụng YOLOv11 (cho việc phát hiện văn bản) và CRNN (cho việc nhận dạng chữ). Đầu vào và đầu ra của chương trình như sau:

- **Đầu vào:** Một bức ảnh chứa văn bản.
- **Đầu ra:** Tọa độ vị trí và nội dung văn bản trong ảnh.

38.2 Thiết lập chương trình

Chúng ta sẽ triển khai dự án này theo bốn giai đoạn, mỗi giai đoạn tập trung vào việc xây dựng các module đã được đề cập ở phần giới thiệu.

- Tải dữ liệu:** Các module trong chương trình Nhận dạng Văn bản trongẢnh của dự án này, bao gồm phần Text Detection và Nhận dạng Văn bản, sẽ được huấn luyện trên tập dữ liệu ICDAR2003. Bạn có thể tải tập dữ liệu này tại [đây](#).
- Thiết lập module Text Detection:** Trong module này, chúng ta sẽ xây dựng một chương trình nhận ảnh đầu vào và trả về các tọa độ bao quanh toàn bộ văn bản có trong ảnh. Cụ thể, chúng ta sẽ sử dụng YOLOv11 để thực hiện công việc này. Bạn có thể xem chi tiết hơn về input/output của bài toán này qua hình sau:



Hình 38.2: Input/Output của bài toán Text Detection.

- Cài đặt các thư viện cần thiết:** Mã nguồn của YOLOv11 được phát triển dựa trên một số thư viện Python khác nhau. Vì vậy, để chạy YOLOv11, chúng ta cần cài đặt các gói thư viện mà YOLOv11 yêu cầu. YOLOv11 đã bao gồm một thư viện có tên ultralytics, và chúng ta có thể cài đặt thư viện này thông qua lệnh pip như sau:

```

1 import ultralytics
2
3 ultralytics.checks()

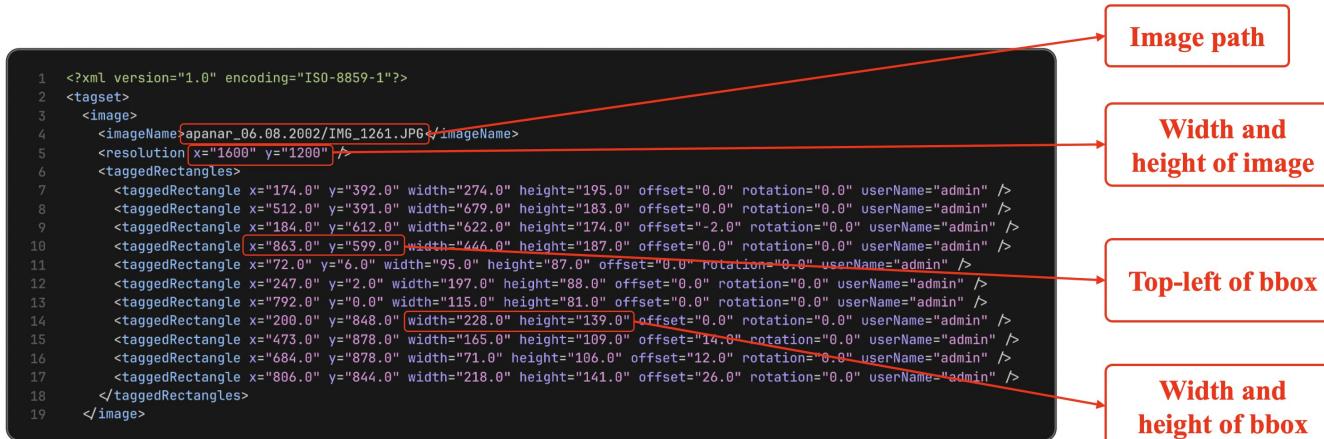
```

Ultralytics 8.3.51 🚀 Python-3.10.16 torch-2.3.0+cu118 CUDA:0 (NVIDIA GeForce RTX 3060, 11931MiB)
Setup complete ✅ (20 CPUs, 31.1 GB RAM, 604.6/915.3 GB disk)

Hình 38.3: Cài đặt thư viện ultralytics.

- (b) **Chuẩn bị bộ dữ liệu:** Cấu trúc thư mục và metadata của ICDAR2003 khác với yêu cầu của mô hình YOLOv11. Do đó, để có thể huấn luyện mô hình YOLOv11, bước đầu tiên là trích xuất thông tin và chuyển đổi chúng sang định dạng tương thích với YOLOv11.

- **Cấu trúc file XML:** Dataset ICDAR2003 được lưu trữ trong file XML (`words.xml`). Dưới đây là một ví dụ về một hình ảnh trong file `words.xml`. Mỗi hình ảnh sẽ bao gồm các thông tin cơ bản như tên file, kích thước ảnh, và tọa độ của các bounding box cho từng từ xuất hiện trong ảnh.



Hình 38.4: Các thông tin quan trọng trong file XML.

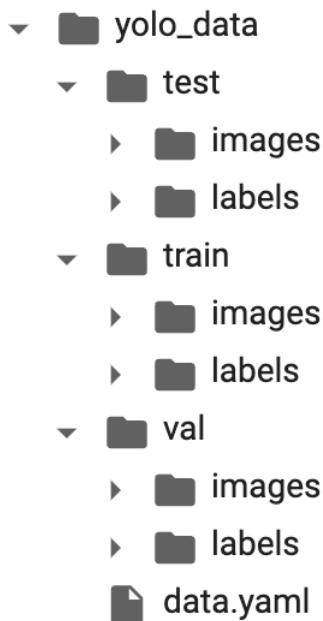
- **Trích xuất thông tin từ file XML:** Chúng ta sẽ sử dụng

mảng (array) để trích xuất các thông tin quan trọng từ file XML. Đối với các bounding box và label, chúng ta sẽ sử dụng mảng hai chiều để lưu trữ vì mỗi hình ảnh có thể có nhiều hơn một bounding box và label. Chúng ta chỉ chọn những từ có label là các ký tự từ 0-9 hoặc a-z để huấn luyện mô hình (dòng 17).

```
1 def extract_data_from_xml(root_dir):
2     xml_path = os.path.join(root_dir, 'words.
3         xml')
4     tree = ET.parse(xml_path)
5     root = tree.getroot()
6
7     img_paths = []
8     img_sizes = []
9     img_labels = []
10    bboxes = []
11
12    for img in root:
13        bbs_of_img = []
14        labels_of_img = []
15
16        for bbs in img.findall('
17            taggedRectangles'):
18            for bb in bbs:
19                # check non-alphabet and non-
20                # number
21                if not bb[0].text.isalnum():
22                    continue
23
24                if "e" in bb[0].text.lower() or "n" in bb[0].text.lower():
25                    continue
26
27                bbs_of_img.append([
28                    float(bb.attrib["x"]),
29                    float(bb.attrib["y"]),
30                    float(bb.attrib["
31                        width"])),
```

```
29                     float(bb.attrib["  
30                         height"])
31                     ]
32                     )
33                     labels_of_img.append(bb[0].
34                     text.lower())
35
36                     img_path = os.path.join(root_dir, img
37                     [0].text)
38                     img_paths.append(img_path)
39                     img_sizes.append((int(img[1].attrib["  
x"])), int(img[1].attrib["y"]))
40                     bboxes.append(bbs_of_img)
41                     img_labels.append(labels_of_img)
42
43                     return img_paths, img_sizes, img_labels,
44                     bboxes
45
46 dataset_dir = 'SceneTrialTrain'
47 img_paths, img_sizes, img_labels, bboxes =
48     extract_data_from_xml(dataset_dir)
```

- **Chuyển đổi sang định dạng YOLOv11:** Sau khi đã trích xuất được các thông tin, bước tiếp theo là chuyển chúng sang định dạng YOLOv11.



Hình 38.5: Cấu trúc thư mục của YOLOv11.

Hình 38.5 minh họa cấu trúc thư mục của YOLOv11 mà chúng ta cần chuyển đổi sang. Cấu trúc này bao gồm 3 thư mục chính: train, test, và val. Trong mỗi thư mục, sẽ có 2 thư mục con: images và labels. Thư mục images sẽ chỉ chứa các hình ảnh, và mỗi file hình ảnh trong đó sẽ có một file tương ứng cùng tên (nhưng khác đuôi, ví dụ *.png và *.txt) trong thư mục labels. Các file label này sẽ chứa thông tin về label và tọa độ của từng bounding box.

```

1 0 0.8162692847124825 0.8125854993160054 0.13183730715287517 0.09028727770177838
2 0 0.5007012622720898 0.9432284541723667 0.8583450210378681 0.10259917920656635
  
```

Hình 38.6: Ví dụ về file label theo định dạng YOLOv11.

Hình 38.6 là ví dụ về một file label theo cấu trúc YOLOv11. Mỗi file này chứa 2 dòng, tương ứng với 2 bounding box xuất hiện trong ảnh. Mỗi dòng bao gồm 5 giá trị: class_id, x, y, width, height, trong đó các tọa độ của bounding box đã được chuẩn hóa trong khoảng [0, 1]. (x, y) là tọa độ của điểm

trung tâm (center) của bounding box.

Sau khi hiểu rõ về định dạng YOLOv11, chúng ta sẽ tiến hành chuyển đổi thông tin sang định dạng này. Các dòng 13-16 trong mã nguồn sẽ chuyển đổi từ tọa độ top left (theo ICDAR2003) sang tọa độ center của YOLOv11, đồng thời chuẩn hóa các giá trị vào khoảng [0, 1]. Mục tiêu của mô hình YOLOv11 trong dự án này là nhận diện vị trí của văn bản, vì vậy chúng ta chỉ cần một class duy nhất là "text".

```
1 def convert_to_yolo_format(image_paths,
2     image_sizes, bounding_boxes):
3     yolo_data = []
4
5     for image_path, image_size, bboxes in zip(
6         image_paths, image_sizes, bounding_boxes
7     ):
8         image_width, image_height =
9             image_size
10
11        yolo_labels = []
12
13        for bbox in bboxes:
14            x, y, w, h = bbox
15
16            # Calculate normalized bounding
17            # box coordinates
18            center_x = (x + w / 2) /
19            image_width
20            center_y = (y + h / 2) /
21            image_height
22            normalized_width = w /
23            image_width
24            normalized_height = h /
25            image_height
26
27            # Because we only have one class,
28            # we set class_id to 0
29            class_id = 0
30
31            # Convert to YOLO format
32            yolo_label = f"{class_id} {
33                center_x} {center_y} {normalized_width} {
34                normalized_height}
```

```

    normalized_height}"
23             yolo_labels.append(yolo_label)
24
25         yolo_data.append((image_path,
26                         yolo_labels))
27
28     return yolo_data
29
30 # Define class labels
31 class_labels = ["text"]
32
33 # Convert data into YOLO format
34 yolo_data = convert_to_yolo_format(
35     image_paths, image_sizes, bounding_boxes)
36

```

- **Lưu data vào folder mới:** Sau khi đã chuyển đổi sang format của YOLOv11, bây giờ chúng ta sẽ lưu data này thành 1 folder mới để có thể tiết kiệm thời gian cho những training sau.

```

1 def save_data(data, src_img_dir, save_dir):
2     # Create folder if not exists
3     os.makedirs(save_dir, exist_ok=True)
4
5     # Make images and labels folder
6     os.makedirs(os.path.join(save_dir, "images"), exist_ok=True)
7     os.makedirs(os.path.join(save_dir, "labels"), exist_ok=True)
8
9     for image_path, yolo_labels in data:
10        # Copy image to images folder
11        shutil.copy(
12            os.path.join(src_img_dir,
13                         image_path), os.path.join(save_dir, "images"))
14
15        # Save labels to labels folder
16        image_name = os.path.basename(

```

```
    image_path)
17         image_name = os.path.splitext(
18             image_name)[0]
19
20         with open(os.path.join(save_dir, "labels",
21             f"{image_name}.txt"), "w") as f:
22             for label in yolo_labels:
23                 f.write(f"{label}\n")
24
25     seed = 0
26     val_size = 0.2
27     test_size = 0.125
28     is_shuffle = True
29     train_data, test_data = train_test_split(
30         yolo_data,
31         test_size=val_size,
32         random_state=seed,
33         shuffle=is_shuffle,
34     )
35     test_data, val_data = train_test_split(
36         test_data,
37         test_size=test_size,
38         random_state=seed,
39         shuffle=is_shuffle,
40     )
41     save_yolo_data_dir = "datasets/yolo_data"
42     os.makedirs(save_yolo_data_dir, exist_ok=True)
43     save_train_dir = os.path.join(
44         save_yolo_data_dir, "train")
45     save_val_dir = os.path.join(
46         save_yolo_data_dir, "val")
47     save_test_dir = os.path.join(
48         save_yolo_data_dir, "test")
49
50     save_data(train_data, dataset_dir,
51             save_train_dir)
52     save_data(test_data, dataset_dir,
53             save_val_dir)
54     save_data(val_data, dataset_dir,
55             save_test_dir)
```

- **Tạo file data.yaml để quản lý chung:** Để thuận tiện trong việc truy cập các đường dẫn của các thư mục train, test và val, cũng như số lượng và tên các class, chúng ta cần tạo một file `data.yaml` để lưu trữ những thông tin này.

`data.yaml` ×

```

1 names:
2 - text
3 nc: 1
4 path: yolo_data
5 test: test/images
6 train: train/images
7 val: val/images

```

Hình 38.7: Ví dụ về file `data.yaml`.

Hình 38.7 là ví dụ về file `data.yaml` trong bài toán của chúng ta. Trong file này, ta có các thành phần sau:

- **names:** tên của các class.
- **nc:** số lượng các class.
- **path:** đường dẫn đến thư mục gốc.
- **train, test, val:** đường dẫn đến các thư mục tương ứng.

Chúng ta cũng có thể tạo file `data.yaml` thông qua đoạn mã dưới đây:

```

1 # Create data.yaml file
2 data_yaml = {
3     "path": "./datasets/yolo_data",
4     "train": "train/images",
5     "test": "test/images",
6     "val": "val/images",
7     "nc": 1,
8     "names": class_labels,
9 }
10
11 yolo_yaml_path = os.path.join(
12     save_yolo_data_dir, "data.yaml")

```

```

12 with open(yolo_yaml_path, "w") as f:
13     yaml.dump(data_yaml, f,
14                 default_flow_style=False)

```

- (c) **Training:** Sau khi đã chuẩn bị toàn bộ data, chúng ta đã có thể bắt đầu training model. Trong project này ta sẽ sử dụng YOLOs làm model chính, ngoài ra các bạn có thể sử dụng các phiên bản khác thông qua github của ultralytics tại [đây](#).

Ta sẽ train model thông qua đoạn code sau:

```

1 from ultralytics import YOLO
2
3 # Load a model
4 model = YOLO("yolo11m.pt")
5
6 # Train model
7 results = model.train(
8     data=yolo_yaml_path,
9     epochs=100,
10    imgsz=640,
11    cache=True,
12    patience=20,
13    plots=True,
14 )
15

```

- (d) **Evaluation:** Bây giờ chúng ta sẽ eval model đã train.

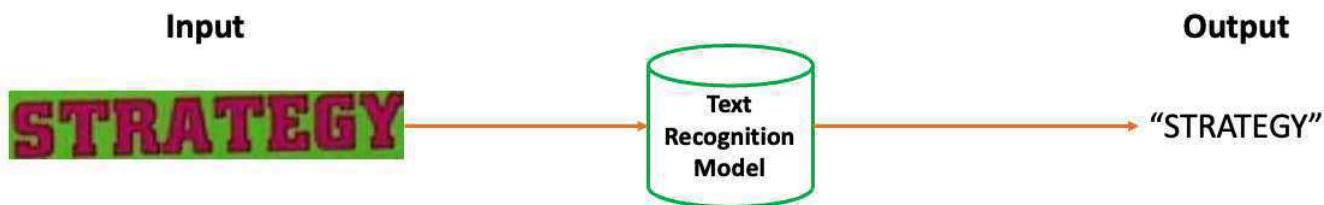
```

1 from ultralytics import YOLO
2
3 model_path = "runs/detect/train/weights/best.pt"
4 model = YOLO(model_path)
5
6 metrics = model.val()
7

```

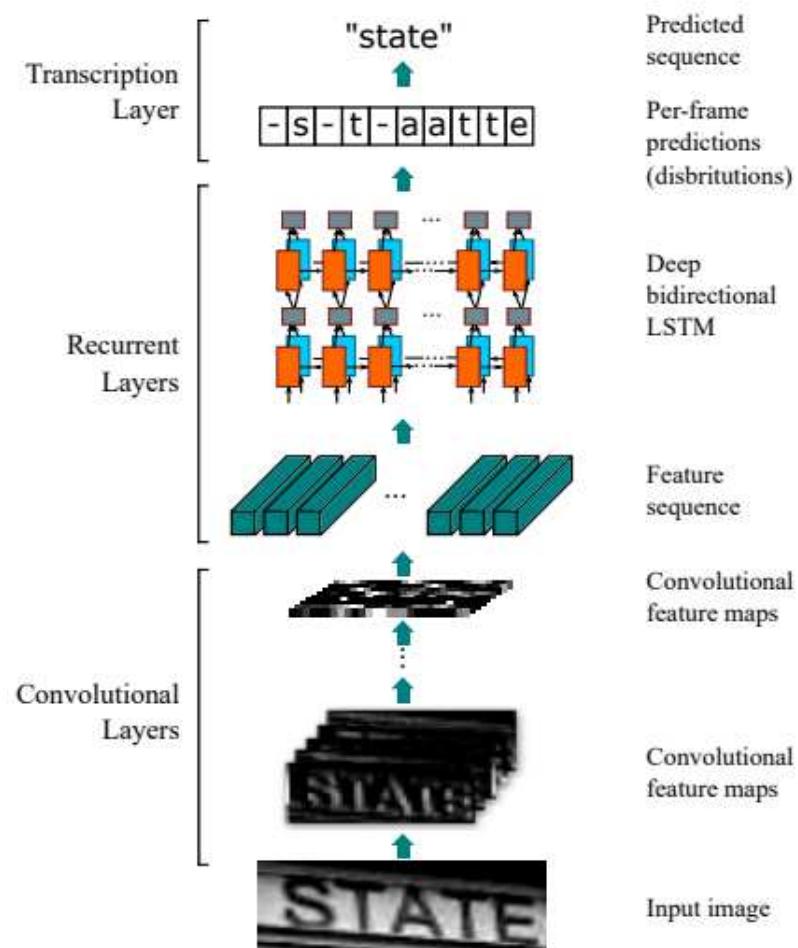
3. **Cài đặt module Text Recognition:** Trong module này, chúng ta sẽ

xây dựng một chương trình nhận vào một ảnh chỉ chứa văn bản và trả về nội dung văn bản dưới dạng chuỗi (string). Các bạn có thể tham khảo rõ hơn về đầu vào và đầu ra (I/O) của bài toán này qua hình dưới đây:



Hình 38.8: Input/Output của bài toán Text Recognition

Để giải quyết bài toán này, chúng ta sẽ chọn mô hình Convolutional Recurrent Neural Networks (CRNN), một mô hình cơ bản nhưng hiệu quả. Đây là một trong những mô hình đầu tiên được phát triển nhằm giải quyết các bài toán dữ liệu vừa ở dạng hình ảnh vừa có tính chuỗi. Cấu trúc của mô hình CRNN như sau:



Hình 38.9: Mô phỏng kiến trúc của mạng CRNN

Mô hình này kết hợp giữa CNN (Convolutional Neural Networks) và RNN (Recurrent Neural Networks) để có thể trích xuất đặc trưng từ hình ảnh và chuỗi văn bản trong ảnh. Đồng thời, CRNN còn sử dụng một hàm mất mát đặc biệt, CTC Loss, nhằm cải thiện kết quả của mô hình. Từ các thông tin trên, chúng ta sẽ xây dựng module này như sau:

- Import các thư viện cần thiết:

```
1 import os
2 import random
3 import time
4 import xml.etree.ElementTree as ET
5
6 import cv2
7 import matplotlib.pyplot as plt
8 import numpy as np
9 import timm
10 import torch
11 import torch.nn as nn
12 import torchvision
13 from PIL import Image
14 from sklearn.model_selection import
15     train_test_split
16 from torch.nn import functional as F
17 from torch.utils.data import DataLoader, Dataset
18 from torchvision import transforms
```

- (b) **Chuẩn bị bộ dữ liệu:** Để huấn luyện mô hình Text Recognition, chúng ta cần có các cặp ảnh chứa chữ văn bản và label văn bản dạng string tương ứng. Song, dataset ICDAR2003 hiện tại chưa thỏa mãn điều này. Chính vì vậy, dựa vào file label của bộ ICDAR2003, ta sẽ cài đặt chương trình trích xuất thành một bộ dữ liệu dành cho bài toán Text Recognition. Tương tự như module trước, ta dùng hàm `extract_data_from_xml()` để lấy thông tin ảnh bounding box như sau:

```
1 def extract_data_from_xml(root_dir):
2     xml_path = os.path.join(root_dir, "words.xml")
3     tree = ET.parse(xml_path)
4     root = tree.getroot()
5
6     img_paths = []
7     img_sizes = []
8     img_labels = []
9     bboxes = []
10
```

```
11     for img in root:
12         bbs_of_img = []
13         labels_of_img = []
14
15         for bbs in img.findall("taggedRectangles"):
16             for bb in bbs:
17                 # check non-alphabet and non-
18                 # number
19                 if not bb[0].text.isalnum():
20                     continue
21
22                 if "e" in bb[0].text.lower() or
23                 "n" in bb[0].text.lower():
24                     continue
25
26                 bbs_of_img.append(
27                     [
28                         float(bb.attrib["x"]),
29                         float(bb.attrib["y"]),
30                         float(bb.attrib["width"]),
31                         float(bb.attrib["height"])
32                     ],
33                     ]
34                 )
35                 labels_of_img.append(bb[0].text.
36                     lower())
37
38                 img_path = os.path.join(root_dir, img
39                     [0].text)
40                 img_paths.append(img_path)
41                 img_sizes.append(int(img[1].attrib["x"]
42                     ), int(img[1].attrib["y"])))
43                 bboxes.append(bbs_of_img)
44                 img_labels.append(labels_of_img)
45
46             return img_paths, img_sizes, img_labels,
47             bboxes
48
49 dataset_dir = "datasets/SceneTrialTrain"
50 img_paths, img_sizes, img_labels, bboxes =
51     extract_data_from_xml(dataset_dir)
```

Với danh sách các ảnh và bounding box ứng với từng ảnh, ta sẽ xây dựng một hàm để cắt các bounding box thành các ảnh riêng biệt chỉ chứa văn bản. Nội dung văn bản từ các bounding box này sẽ được sử dụng làm label và lưu trữ vào một file .txt riêng biệt. Hàm thực hiện công việc này có nội dung như sau:

```
1 def split_bounding_boxes(img_paths, img_labels,
2     bboxes, save_dir):
3     os.makedirs(save_dir, exist_ok=True)
4
5     count = 0
6     labels = [] # List to store labels
7
8     for img_path, img_label, bbs in zip(
9         img_paths, img_labels, bboxes):
10        img = Image.open(img_path)
11
12        for label, bb in zip(img_label, bbs):
13            # Crop image
14            cropped_img = img.crop((bb[0], bb
15 [1], bb[0] + bb[2], bb[1] + bb[3]))
16
17            # filter out if 90% of the cropped
18            # image is black or white
19            if np.mean(cropped_img) < 35 or np.
20            mean(cropped_img) > 220:
21                continue
22
23                if cropped_img.size[0] < 10 or
24                cropped_img.size[1] < 10:
25                    continue
26
27                # Save image
28                filename = f"{count:06d}.jpg"
29                cropped_img.save(os.path.join(
30                    save_dir, filename))
31
32                new_img_path = os.path.join(save_dir
33 , filename)
34
35                label = new_img_path + "\t" + label
36
37                labels.append(label) # Append label
```

```

        to the list
30
31         count += 1
32
33     print(f"Created {count} images")
34
35     # Write labels to a text file
36     with open(os.path.join(save_dir, "labels.txt",
37     "w")) as f:
38         for label in labels:
39             f.write(f"{label}\n")
40
41 save_dir = "datasets/ocr_dataset"
42 split_bounding_boxes(img_paths, img_labels,
43     bboxes, save_dir)
44

```

Khi chạy đoạn code trên, ta sẽ có một thư mục dataset mới dành cho bài Text Recognition, các ảnh này sẽ có dạng như sau:



Hình 38.10: Một số mẫu dữ liệu từ bộ dữ liệu OCR

- (c) **Đọc dữ liệu:** Với thư mục dữ liệu mới, ta viết chương trình đọc và lưu trữ vào hai list rỗng, một list dùng để chứa đường dẫn ảnh và một list để chứa string label tương ứng như sau:

```

1 root_dir = save_dir
2
3 img_paths = []

```

```

4 labels = []
5
6 # Read labels from text file
7 with open(os.path.join(root_dir, "labels.txt"),
8     "r") as f:
9     for label in f:
10         labels.append(label.strip().split("\t")
11             [1])
12         img_paths.append(label.strip().split("\t"
13             )[0])
14
15 print(f"Total images: {len(img_paths)}")

```

- (d) **Xây dựng bộ từ vựng (vocabulary):** Đối với loại bài toán này, để mô hình CRNN có thể xuất ra văn bản từ hình ảnh, chúng ta cần xác định một bộ từ vựng trước. Mô hình CRNN sẽ thực hiện dự đoán ở cấp độ ký tự, vì vậy bộ từ vựng sẽ bao gồm các ký tự có mặt trong dữ liệu. Do đó, chương trình của chúng ta sẽ như sau:

```

1 letters = [char.split(".")[0].lower() for char
2     in labels]
3 letters = "".join(letters)
4 letters = sorted(list(set(list(letters))))
5
6 # create a string of all characters in the
7 # dataset
8 chars = "".join(letters)
9
10 # for "blank" character
11 blank_char = "-"
12 chars += blank_char
13 vocab_size = len(chars)
14
15 print(f"Vocab: {chars}")
16 print(f"Vocab size: {vocab_size}")

```

Trong đoạn mã trên, chúng ta duyệt qua tất cả các nhãn trong bộ dữ liệu và chọn ra các ký tự xuất hiện để tạo thành bộ từ vựng.

Cần lưu ý rằng, khi sử dụng CTC Loss, ta phải định nghĩa một ký tự "blank". Dựa trên bộ từ vựng này, chúng ta sẽ xây dựng một từ điển để lưu trữ các ký tự và mã số thứ tự tương ứng như sau:

```
1 char_to_idx = {char: idx + 1 for idx, char in
2     enumerate(sorted(chars))}3 idx_to_char = {index: char for char, index in
3     char_to_idx.items()}
```

Cuối cùng, ta xây dựng hàm dùng để chuyển đổi một string thành vector các kí tự được đại diện bằng mã số của chúng (token):

```
1 max_label_len = max([len(label) for label in
2     labels])3
3 def encode(label, char_to_idx, max_label_len):
4     encoded_labels = torch.tensor(
5         [char_to_idx[char] for char in label],
6         dtype=torch.int32
7     )
8     label_len = len(encoded_labels)
9     lengths = torch.tensor(
10        label_len,
11        dtype=torch.int32
12    )
13     padded_labels = F.pad(
14         encoded_labels,
15         (0, max_label_len - label_len),
16         value=0
17    )
18
19     return padded_labels, lengths
20
```

Khi chuyển đổi các nhãn thành tensor, cần đảm bảo rằng tất cả các tensor đều có kích thước đồng nhất để tránh lỗi khi sử dụng DataLoader. Tuy nhiên, các nhãn có độ dài khác nhau, vì vậy chúng ta cần xác định một chiều dài chung (max_label_len) và thực hiện padding nếu cần thiết để đảm bảo điều này (lưu ý rằng

mã số của ký tự padding là 0).

```

1 def decode(encoded_sequences, idx_to_char,
2           blank_char="-"):
3     decoded_sequences = []
4
5     for seq in encoded_sequences:
6         decoded_label = []
7         prev_char = None # To track the
8         previous character
9
10        for token in seq:
11            if token != 0: # Ignore padding (
12                token = 0)
13                char = idx_to_char[token.item()]
14                # Append the character if it's
15                not a blank or the same as the previous
16                character
17                if char != blank_char:
18                    if char != prev_char or
19                    prev_char == blank_char:
20                        decoded_label.append(
21                            char)
22                        prev_char = char # Update
23                        previous character
24
25                decoded_sequences.append("".join(
26                    decoded_label))
27
28        print(f"From {encoded_sequences} to {(
29            decoded_sequences})")
30
31    return decoded_sequences
32

```

Ngược lại với hàm `encode()`, ta cũng sẽ xây dựng một hàm để đổi ngược lại tensor các token thành dạng string của chúng như trên.

- (e) **Xây dựng hàm tiền xử lý dữ liệu:** Để kết quả huấn luyện được tốt hơn, ta sẽ xây dựng hàm tiền xử lý dữ liệu đầu vào như sau:

```
1 data_transforms = {
2     "train": transforms.Compose(
3         [
4             transforms.Resize((100, 420)),
5             transforms.ColorJitter(
6                 brightness=0.5,
7                 contrast=0.5,
8                 saturation=0.5,
9             ),
10            transforms.Grayscale(
11                num_output_channels=1,
12            ),
13            transforms.GaussianBlur(3),
14            transforms.RandomAffine(
15                degrees=1,
16                shear=1,
17            ),
18            transforms.RandomPerspective(
19                distortion_scale=0.3,
20                p=0.5,
21                interpolation=3,
22            ),
23            transforms.RandomRotation(degrees=2)
24        ,
25            transforms.ToTensor(),
26            transforms.Normalize((0.5,), (0.5,))
27        ,
28    ],
29    "val": transforms.Compose(
30        [
31            transforms.Resize((100, 420)),
32            transforms.Grayscale(
33                num_output_channels=1),
34            transforms.ToTensor(),
35            transforms.Normalize((0.5,), (0.5,))
36        ],
37    )
38}
```

Lưu ý rằng, trong quá trình huấn luyện, chúng ta cần áp dụng

một số kỹ thuật tăng cường dữ liệu để cải thiện hiệu quả của mô hình. Do đó, sẽ có hai hàm chuyển đổi (transforms) riêng biệt cho hai giai đoạn huấn luyện và đánh giá.

- (f) **Chia bộ dữ liệu train, val, test:** Với dữ liệu đã đọc, ta chia ba bộ dữ liệu train, val, test với tỉ lệ 7:2:1 như sau:

```
1 seed = 0
2 val_size = 0.1
3 test_size = 0.1
4 is_shuffle = True
5
6 X_train, X_val, y_train, y_val =
    train_test_split(
        img_paths,
        labels,
        test_size=val_size,
        random_state=seed,
        shuffle=is_shuffle,
    )
13
14 X_train, X_test, y_train, y_test =
    train_test_split(
        X_train,
        y_train,
        test_size=test_size,
        random_state=seed,
        shuffle=is_shuffle,
    )
21
```

- (g) **Xây dựng class pytorch datasets:** Ta xây dựng pytorch dataset cho bộ dữ liệu Text Recognition như sau:

```
1 class STRDataset(Dataset):
2     def __init__(
3         self,
4         X,
5         y,
6         char_to_idx,
7         max_label_len,
```

```

8         label_encoder=None ,
9         transform=None ,
10    ):
11        self.transform = transform
12        self.img_paths = X
13        self.labels = y
14        self.char_to_idx = char_to_idx
15        self.max_label_len = max_label_len
16        self.label_encoder = label_encoder
17
18    def __len__(self):
19        return len(self.img_paths)
20
21    def __getitem__(self, idx):
22        label = self.labels[idx]
23        img_path = self.img_paths[idx]
24        img = Image.open(img_path).convert("RGB")
25
26        if self.transform:
27            img = self.transform(img)
28
29        if self.label_encoder:
30            encoded_label, label_len = self.
31            label_encoder(
32                label, self.char_to_idx, self.
33                max_label_len
34            )
35        return img, encoded_label, label_len

```

- (h) **Xây dựng dataloader:** Với pytorch dataset đã định nghĩa, ta tạo dataloader cho ba bộ train, val, test như sau:

```

1 train_dataset = STRDataset(
2     X_train,
3     y_train,
4     char_to_idx=char_to_idx,
5     max_label_len=max_label_len,
6     label_encoder=encode,
7     transform=data_transforms["train"],
8 )

```

```
9 val_dataset = STRDataset(
10     X_val,
11     y_val,
12     char_to_idx=char_to_idx,
13     max_label_len=max_label_len,
14     label_encoder=encode,
15     transform=data_transforms["val"],
16 )
17 test_dataset = STRDataset(
18     X_test,
19     y_test,
20     char_to_idx=char_to_idx,
21     max_label_len=max_label_len,
22     label_encoder=encode,
23     transform=data_transforms["val"],
24 )
25
26 train_batch_size = 64
27 test_batch_size = 64 * 2
28
29 train_loader = DataLoader(
30     train_dataset,
31     batch_size=train_batch_size,
32     shuffle=True,
33 )
34 val_loader = DataLoader(
35     val_dataset,
36     batch_size=test_batch_size,
37     shuffle=False,
38 )
39 test_loader = DataLoader(
40     test_dataset,
41     batch_size=test_batch_size,
42     shuffle=False,
43 )
44
```

- (i) **Xây dựng model:** Tại đây, chúng ta sẽ bắt đầu triển khai mô hình CRNN. Cụ thể, chúng ta sẽ kết hợp mô hình ResNet101 đã được huấn luyện trước với các lớp Bi-LSTM. Đặc trưng được trích xuất từ ResNet sẽ được biến đổi để đưa vào Bi-LSTM. Sau đó, chúng ta sử dụng trạng thái ẩn cuối cùng của Bi-LSTM để dự đoán chuỗi ký tự trong ảnh. Cách cài đặt sẽ như sau:

```
1  class CRNN(nn.Module):
2      def __init__(
3          self, vocab_size, hidden_size, n_layers,
4          dropout=0.2, unfreeze_layers=3
5      ):
6          super(CRNN, self).__init__()
7
8          backbone = timm.create_model("resnet152",
9              in_chans=1, pretrained=True)
10         modules = list(backbone.children())[:-2]
11         modules.append(nn.AdaptiveAvgPool2d((1,
12             None)))
13         self.backbone = nn.Sequential(*modules)
14
15         # Unfreeze the last few layers
16         for parameter in self.backbone[-unfreeze_layers:].
17             parameters():
18             parameter.requires_grad = True
19
20         self.mapSeq = nn.Sequential(
21             nn.Linear(2048, 512), nn.ReLU(), nn.
22             Dropout(dropout)
23         )
24
25         self.gru = nn.GRU(
26             512,
27             hidden_size,
28             n_layers,
29             bidirectional=True,
30             batch_first=True,
31             dropout=dropout if n_layers > 1 else
32             0,
33         )
34         self.layer_norm = nn.LayerNorm(
35             hidden_size * 2)
36
37         self.out = nn.Sequential(
38             nn.Linear(hidden_size * 2,
39             vocab_size), nn.LogSoftmax(dim=2)
40         )
41
42         @torch.autocast(device_type="cuda")
43     def forward(self, x):
```

```

36         x = self.backbone(x)
37         x = x.permute(0, 3, 1, 2)
38         x = x.view(x.size(0), x.size(1), -1)  # 
39         Flatten the feature map
40         x = self.mapSeq(x)
41         x, _ = self.gru(x)
42         x = self.layer_norm(x)
43         x = self.out(x)
44         x = x.permute(1, 0, 2)  # Based on CTC
45
46         return x

```

Sau khi đã định nghĩa class cho CRNN, ta khai báo mô hình như sau:

```

1 hidden_size = 256
2 n_layers = 3
3 dropout_prob = 0.2
4 unfreeze_layers = 3
5 device = "cuda" if torch.cuda.is_available()
       else "cpu"
6
7 model = CRNN(
8     vocab_size=vocab_size,
9     hidden_size=hidden_size,
10    n_layers=n_layers,
11    dropout=dropout_prob,
12    unfreeze_layers=unfreeze_layers,
13 ).to(device)
14

```

- (j) **Xây dựng hàm train và evaluate:** Ta định nghĩa hàm dùng để huấn luyện mô hình và đánh giá mô hình như sau:

```

1 def evaluate(model, dataloader, criterion,
2              device):
3     model.eval()
4     losses = []
5     with torch.no_grad():

```

```
5         for inputs, labels, labels_len in
6             dataloader:
7                 inputs = inputs.to(device)
8                 labels = labels.to(device)
9                 labels_len = labels_len.to(device)
10
11                 outputs = model(inputs)
12                 logits_lens = torch.full(
13                     size=(outputs.size(1),),
14                     fill_value=outputs.size(0), dtype=torch.long
15                     ).to(device)
16
17                 loss = criterion(outputs, labels,
18                     logits_lens, labels_len)
19                 losses.append(loss.item())
20
21                 loss = sum(losses) / len(losses)
22
23             return loss
24
25     def fit(
26         model, train_loader, val_loader, criterion,
27         optimizer, scheduler, device, epochs
28     ):
29         train_losses = []
30         val_losses = []
31
32         for epoch in range(epochs):
33             start = time.time()
34
35             batch_train_losses = []
36
37             model.train()
38             for idx, (inputs, labels, labels_len) in
39                 enumerate(train_loader):
40                 inputs = inputs.to(device)
41                 labels = labels.to(device)
42                 labels_len = labels_len.to(device)
43
44                 optimizer.zero_grad()
45
46                 outputs = model(inputs)
47
48                 logits_lens = torch.full(
```

```
44             size=(outputs.size(1),),
45             fill_value=outputs.size(0),
46             dtype=torch.long,
47             ).to(device)
48
49             loss = criterion(outputs, labels,
50                               logits_lens, labels_len)
51
52             loss.backward()
53             torch.nn.utils.clip_grad_norm_(model
54             .parameters(), 5)
55             optimizer.step()
56
57             batch_train_losses.append(loss.item()
58             ())
59
60             train_loss = sum(batch_train_losses) /
61             len(batch_train_losses)
62             train_losses.append(train_loss)
63
64             val_loss = evaluate(model, val_loader,
65             criterion, device)
66             val_losses.append(val_loss)
67
68             print(
69                 f"EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal loss: {val_loss:.4f}\t\t
70                 Time: {time.time() - start:.2f} seconds"
71             )
72
73             scheduler.step()
74
75         return train_losses, val_losses
```

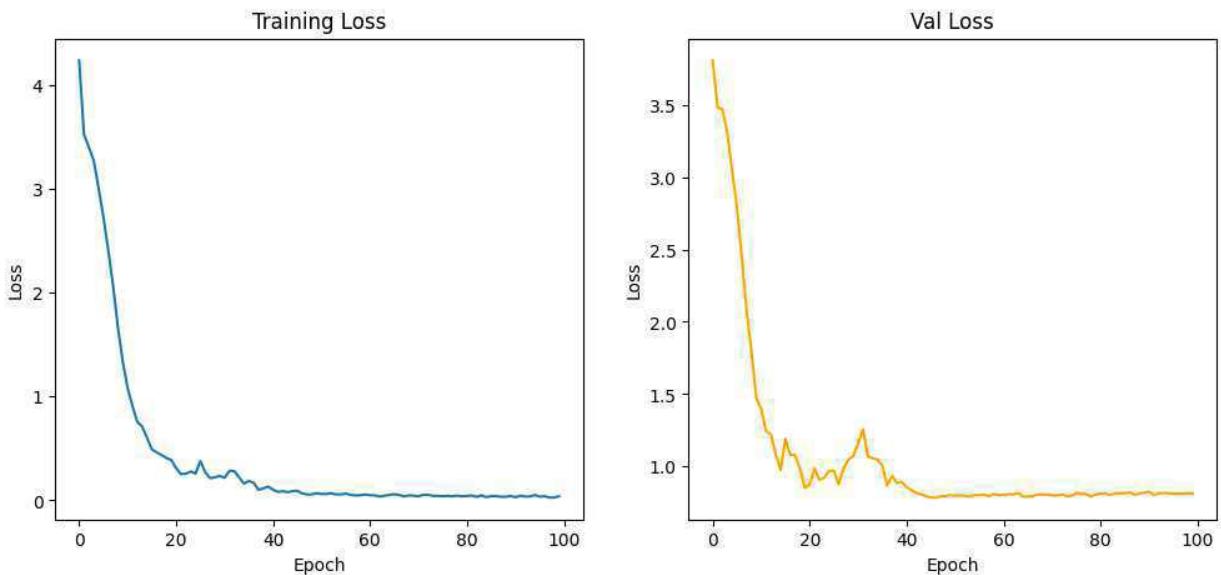
- (k) **Khai báo hàm loss, optimizer và scheduler:** Sau khi xây dựng mô hình, chúng ta cần khai báo hàm loss, optimizer và learning rate scheduler. Như đã đề cập ở phần đầu, chúng ta sẽ sử dụng hàm CTC Loss cho bài toán này. Learning rate scheduler sẽ giúp giảm learning rate trong các epoch sau để cải thiện hiệu quả huấn luyện.

```
1 epochs = 100
2 lr = 5e-4
3 weight_decay = 1e-5
4 scheduler_step_size = epochs * 0.5
5
6 criterion = nn.CTCLoss(
7     blank=char_to_idx[blank_char],
8     zero_infinity=True,
9     reduction="mean",
10 )
11 optimizer = torch.optim.Adam(
12     model.parameters(),
13     lr=lr,
14     weight_decay=weight_decay,
15 )
16 scheduler = torch.optim.lr_scheduler.StepLR(
17     optimizer, step_size=scheduler_step_size,
18     gamma=0.1
19 )
```

- (l) **Training:** Ta gọi hàm `fit()` kèm các tham số phù hợp để bắt đầu quá trình huấn luyện:

```
1 train_losses, val_losses = fit(
2     model, train_loader, val_loader, criterion,
3     optimizer, scheduler, device, epochs
4 )
```

Khi quá trình huấn luyện kết thúc, ta có thể dùng kết quả loss ghi nhận được qua từng epoch trên hai tập train và val lên đồ thị như sau:



Hình 38.11: Kết quả huấn luyện của mô hình CRNN trên hai tập train và val

- (m) **Evaluation:** Sau khi quá trình huấn luyện hoàn tất, ta đánh giá mô hình trên hai tập val và test:

```

1 val_loss = evaluate(model, val_loader, criterion
                      , device)
2 test_loss = evaluate(model, test_loader,
                      criterion, device)
3
4 print("Evaluation on val/test dataset")
5 print("Val loss: ", val_loss)
6 print("Test loss: ", test_loss)
7

```

- (n) **Lưu model:** Cuối cùng, chúng ta lưu lại model đã train được vào máy.

```

1 save_model_path = "ocr_crnn.pt"
2 torch.save(model.state_dict(), save_model_path)

```

3

4. **Cài đặt toàn bộ pipeline:** Sau khi đã hoàn tất xây dựng hai mô hình Text Detection và Text Recognition, chúng ta sẽ xây dựng chương trình Scene Text Recognition hoàn chỉnh thông qua việc kết hợp hai mô hình trên lại. Các bước thực hiện như sau:

(a) Import các thư viện cần thiết:

```
1 %pip install ultralytics
2 import ultralytics
3 ultralytics.checks()
4
5 import os
6 import numpy as np
7 import timm
8 import matplotlib.pyplot as plt
9
10 import torch
11 import torch.nn as nn
12 from torchvision import models
13 from torchvision import transforms
14 from PIL import Image
15
16 from ultralytics import YOLO
17
```

(b) Load các mô hình đã huấn luyện: Khởi tạo hai model về Text Detection và Text Recognition đã huấn luyện:

i. Text Detection:

```
1 text_det_model_path = 'runs/detect/train/
2   weights/best.pt'
3 yolo = YOLO(text_det_model_path)
```

ii. Text Recognition:

```
1  class CRNN(nn.Module):
2      def __init__(self, vocab_size,
3          hidden_size, n_layers, dropout=0.2,
4          unfreeze_layers=3):
5          super(CRNN, self).__init__()
6
7          backbone = timm.create_model(
8              'resnet101',
9              in_chans=1,
10             pretrained=True
11         )
12         modules = list(backbone.children())
13         [:-2]
14         modules.append(nn.AdaptiveAvgPool2d
15             ((1, None)))
16         self.backbone = nn.Sequential(*
17             modules)
18
19         # Unfreeze the last few layers
20         for parameter in self.backbone[-unfreeze_layers:].parameters():
21             parameter.requires_grad = True
22
23         self.mapSeq = nn.Sequential(
24             nn.Linear(2048, 512),
25             nn.ReLU(),
26             nn.Dropout(dropout)
27         )
28
29         self.lstm = nn.LSTM(
30             512, hidden_size,
31             n_layers, bidirectional=True,
32             batch_first=True,
33             dropout=dropout if n_layers > 1
34             else 0
35         )
36         self.layer_norm = nn.LayerNorm(
37             hidden_size * 2)
38
39         self.out = nn.Sequential(
40             nn.Linear(hidden_size * 2,
41             vocab_size),
42             nn.LogSoftmax(dim=2)
```

```
34         )
35
36     def forward(self, x):
37         x = self.backbone(x)
38         x = x.permute(0, 3, 1, 2)
39         x = x.view(x.size(0), x.size(1), -1)
40         # Flatten the feature map
41         x = self.mapSeq(x)
42         x, _ = self.lstm(x)
43         x = self.layer_norm(x)
44         x = self.out(x)
45         x = x.permute(1, 0, 2) # Based on CTC
46
47         return x
48
49 chars = '0123456789abcdefghijklmnopqrstuvwxyz'
50
51 vocab_size = len(chars)
52 char_to_idx = {char: idx + 1 for idx, char in
53                 enumerate(sorted(chars))}
54 idx_to_char = {idx: char for char, idx in
55                 char_to_idx.items()}
56
57 hidden_size = 256
58 n_layers = 3
59 dropout_prob = 0.2
60 unfreeze_layers=3
61 device = 'cuda' if torch.cuda.is_available()
62 else 'cpu'
63 model_path = 'ocr_crnn_base_best.pt'
64
65 crnn_model = CRNN(
66     vocab_size=vocab_size,
67     hidden_size=hidden_size,
68     n_layers=n_layers,
69     dropout=dropout_prob,
70     unfreeze_layers=unfreeze_layers
71 ).to(device)
72 crnn_model.load_state_dict(torch.load(
73     model_path))
74
```

- (c) **Xây dựng hàm decode:** Khởi tạo hàm decode() để chuyển kết quả dự đoán của mô hình thành string:

```
1 def decode(encoded_sequences, idx_to_char,
2     blank_char='-' ):
3     decoded_sequences = []
4
5     for seq in encoded_sequences:
6         decoded_label = []
7         for idx, token in enumerate(seq):
8             if token != 0:
9                 char = idx_to_char[token.item()]
10                if char != blank_char:
11                    decoded_label.append(char)
12
13                decoded_sequences.append(''.join(
14                    decoded_label))
15
16    return decoded_sequences
```

- (d) **Xây dựng hàm Text Detection:** Ta dùng hàm này để phát hiện vị trí của tất cả văn bản xuất hiện trong một ảnh sử dụng mô hình YOLOv11 đã huấn luyện:

```
1 def text_detection(img_path, text_det_model):
2     text_det_results = text_det_model(img_path,
3                                     verbose=False)[0]
4
5     bboxes = text_det_results.boxes.xyxy.tolist()
6     classes = text_det_results.boxes.cls.tolist()
7     names = text_det_results.names
8     confs = text_det_results.boxes.conf.tolist()
9
10    return bboxes, classes, names, confs
```

- (e) **Xây dựng hàm nhận diện văn bản:** Ta dùng hàm này để nhận diện văn bản bên trong một ảnh bất kì với model CRNN đã huấn luyện:

```

1 def text_recognition(img, data_transforms,
2     text_reg_model, idx_to_char, device):
3     transformed_image = data_transforms(img)
4     transformed_image = transformed_image.
5         unsqueeze(0).to(device)
6     text_reg_model.eval()
7     with torch.no_grad():
8         logits = text_reg_model(
9             transformed_image).detach().cpu()
10        text = decode(logits.permute(1, 0, 2).argmax
11                      (2), idx_to_char)
12
13    return text
14
15

```

(f) Xây dựng hàm trực quan hóa kết quả dự đoán:

```

1 def visualize_detections(img, detections):
2     plt.figure(figsize=(12, 8))
3     plt.imshow(img)
4     plt.axis('off')
5
6     for bbox, detected_class, confidence,
7         transcribed_text in detections:
8         x1, y1, x2, y2 = bbox
9         plt.gca().add_patch(plt.Rectangle((x1,
10             y1), x2 - x1, y2 - y1, fill=False, edgecolor=
11             'red', linewidth=2))
12         plt.text(
13             x1, y1 - 10, f'{detected_class} ({confidence:.2f}): {transcribed_text}',
14             fontsize=9, bbox=dict(facecolor='red',
15             , alpha=0.5)
16         )
17
18     plt.show()
19
20

```

(g) Xây dựng hàm tiền xử lý liệu: Ta sử dụng lại hàm transforms đã triển khai cho mô hình CRNN:

```

1 data_transforms = {
2     'train': transforms.Compose([
3         transforms.Resize((100, 420)),
4         transforms.ColorJitter(brightness=0.5,
5             contrast=0.5, saturation=0.5),
6         transforms.Grayscale(num_output_channels
7             =1),
8         transforms.GaussianBlur(3),
9         transforms.RandomAffine(degrees=1, shear
10            =1),
11         transforms.RandomPerspective(
12             distortion_scale=0.2, p=0.3, interpolation=3)
13         ,
14         transforms.RandomRotation(degrees=2),
15         transforms.ToTensor(),
16         transforms.Normalize((0.5,), (0.5,)),
17     ]),
18     'val': transforms.Compose([
19         transforms.Resize((100, 420)),
20         transforms.Grayscale(num_output_channels
21             =1),
22         transforms.ToTensor(),
23         transforms.Normalize((0.5,), (0.5,)),
24     ]),
25 }

```

- (h) **Xây dựng hàm dự đoán:** Cuối cùng, ta xây dựng hàm predict() để sử dụng chương trình STR:

```

1 def predict(img_path, data_transforms,
2             text_det_model, text_reg_model, idx_to_char,
3             device):
4     # Detection
5     bboxes, classes, names, confs =
6     text_detection(img_path, text_det_model)
7
8     # Load the image
9     img = Image.open(img_path)
10
11     predictions = []

```

```
9      # Iterate through the results
10     for bbox, cls, conf in zip(bboxes, classes,
11                               confs):
12         x1, y1, x2, y2 = bbox
13         confidence = conf
14         detected_class = cls
15         name = names[int(cls)]
16
17         # Extract the detected object and crop
18         it
19         cropped_image = img.crop((x1, y1, x2, y2))
20
21         transcribed_text = text_recognition(
22             cropped_image,
23             data_transforms,
24             text_reg_model,
25             idx_to_char,
26             device
27         )
28
29         predictions.append((bbox, name,
30                             confidence, transcribed_text))
31
32     visualize_detections(img, predictions)
33
34     return predictions
```



Hình 38.12: Một số kết quả của mô hình mà chúng ta đã xây dựng

38.3 Câu hỏi trắc nghiệm

1. Bài toán Scene Text Recognition nhằm mục đích gì??
 - (a) Nhận diện văn bản trong hình ảnh.
 - (b) Phân loại hình ảnh.
 - (c) Xác định đối tượng trong video.
 - (d) Tính toán biểu đồ dữ liệu.
2. Ứng dụng phổ biến của Scene Text Recognition bao gồm:
 - (a) Tự động chụp ảnh selfie.
 - (b) Dịch ngôn ngữ tự động từ văn bản trong hình ảnh.
 - (c) Nhận diện khuôn mặt.
 - (d) Phân loại thẻ loại sách.
3. Mục tiêu của Detector trong bài Scene Text Recognition là gì?
 - (a) Nhận diện ngôn ngữ tự nhiên.
 - (b) Xác định vị trí của văn bản trong hình ảnh.
 - (c) Phân loại thẻ loại sách.
 - (d) Dự đoán từ ngữ chính trong câu.
4. Mục tiêu của Regconizer trong bài Scene Text Recognition là gì?
 - (a) Phân loại hình ảnh.
 - (b) Chuyển đổi văn bản hình ảnh thành dạng văn bản có thể đọc được.
 - (c) Nhận diện ngôn ngữ tự nhiên.
 - (d) Đoán từ ngữ chính trong câu.
5. Đặc điểm chung của hình ảnh gây thách thức trong bài toán Scene Text Recognition là gì?
 - (a) Ánh sáng đồng đều.
 - (b) Kích thước văn bản đồng nhất.

- (c) Nền phức tạp và nhiễu.
(d) Màu sắc đa dạng.
6. Deep learning models thường được ưa chuộng trong Scene Text Recognition vì:
(a) Chúng có khả năng nhận diện đối tượng.
(b) Chúng tự động học các đặc trưng.
(c) Chúng chỉ chấp nhận dữ liệu ảnh đen trắng.
(d) Chúng không ổn định và khó điều chỉnh.
7. YOLOv11 là một model detection thuộc loại:
(a) Single-stage
(b) Two-stage
(c) Three-stage
(d) Multi-stage
8. YOLOv11 hoạt động như thế nào?
(a) Chia hình ảnh thành các ô nhỏ và dự đoán các đối tượng trong mỗi ô.
(b) Dự đoán các đối tượng trong hình ảnh bằng cách sử dụng các bounding box.
(c) Dự đoán các đối tượng trong hình ảnh bằng cách sử dụng các đặc trưng.
(d) Tất cả các đáp án trên.
9. CRNN hoạt động như thế nào?
(a) Sử dụng một mạng CNN để trích xuất các đặc trưng từ hình ảnh văn bản.
(b) Sử dụng một mạng RNN để dự đoán các ký tự trong văn bản.
(c) Sử dụng một mạng CTC để chuyển đổi các dự đoán của mạng RNN thành văn bản.
(d) Tất cả các đáp án trên.

10. Trong CRNN, mô hình CNN (Convolutional Neural Network) thường được sử dụng để làm gì?
 - (a) Nhận diện vị trí văn bản.
 - (b) Phân loại hình ảnh.
 - (c) Trích xuất đặc trưng từ hình ảnh.
 - (d) Xác định kích thước hình ảnh.
11. Quy trình tích hợp giữa YOLOv11 và CRNN trong bài toán Scene Text Recognition thường như thế nào?
 - (a) CRNN chỉ hoạt động độc lập, không cần YOLOv11.
 - (b) YOLOv11 chỉ được sử dụng để định vị vị trí của văn bản, sau đó CRNN đọc văn bản.
 - (c) YOLOv11 và CRNN hoạt động độc lập, không tương tác.
 - (d) CRNN chỉ được sử dụng để cải thiện kết quả của YOLOv11.

38.4 Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 6 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Triển khai streamlit:** Các bạn có thể truy cập vào web streamlit và link GitHub tại [đây](#).
4. **Rubric:**

Scene Text Recognition Project - Rubric		
Câu	Kiến Thức	Đánh Giá
II.2.	<ul style="list-style-type: none"> - Kiến thức về YOLOv11. - Kiến thức về cách huấn luyện YOLOv11 cho bài toán Text Detection. 	<ul style="list-style-type: none"> - Biết cách ứng dụng YOLOv11 để xây dựng mô hình Text Detection.
II.3.	<ul style="list-style-type: none"> - Kiến thức về Convolution Recurrent Neural Network (CRNN). - Cách về việc xây dựng mô hình CRNN sử dụng pytorch. - Kiến thức về CTCLoss. - Khả năng ứng dụng mô hình CRNN trong bài toán Text Recognition có trong ảnh. 	<ul style="list-style-type: none"> - Hiểu được các khái niệm liên quan đến mô hình CRNN. - Cách lập trình và ứng dụng mô hình CRNN vào giải quyết bài toán Text Recognition.
II.3.	<ul style="list-style-type: none"> - Kiến thức về cách xây dựng chương trình Scene Text Recognition (STR). 	<ul style="list-style-type: none"> - Hiểu được cách vận dụng hai mô hình Text Detection và Text Recognition để xây dựng một chương trình về Scene Text Detection.

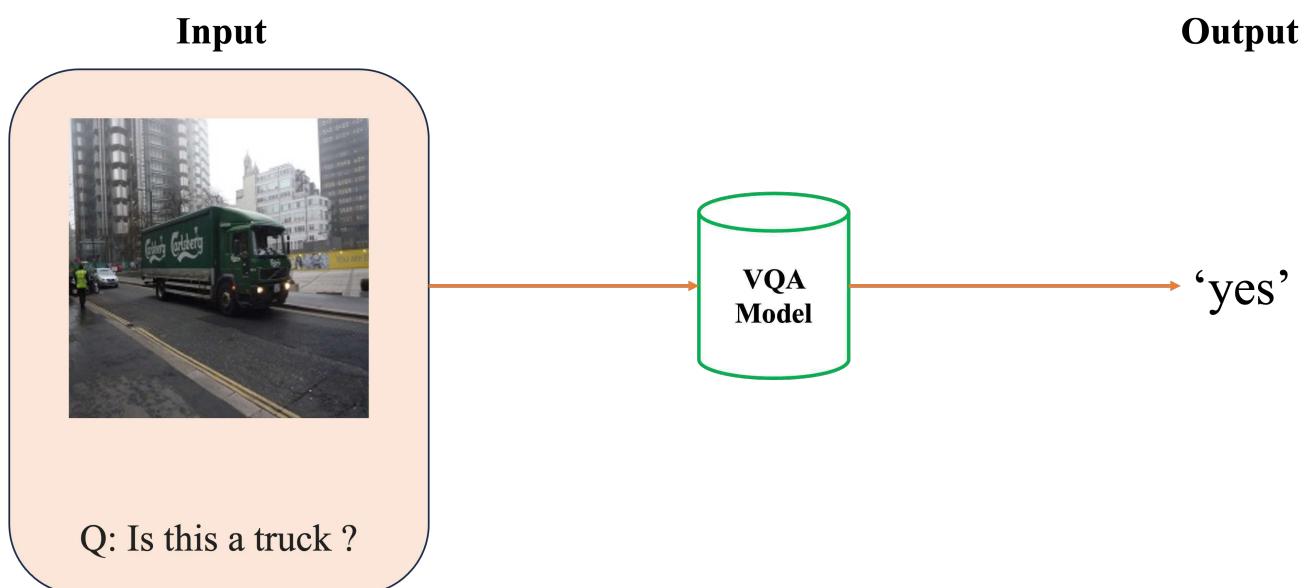
- *Hết* -

Chương 39

Project 2: Visual Question Answering (luyện tập dùng 2 kiểu dữ liệu text và ảnh)

39.1 Giới thiệu

Visual Question Answering (VQA) là một bài toán phổ biến trong Machine Learning, ứng dụng các kỹ thuật liên quan đến hai lĩnh vực Computer Vision và Natural Language Processing để giải quyết bài toán này. Mục tiêu cốt lõi của VQA là dựa trên ảnh đầu vào để trả lời các câu hỏi có liên quan đến hình ảnh. Theo đó, bước đầu sẽ sử dụng các kỹ thuật xử lý hình ảnh và ngôn ngữ tự nhiên để trích các xuất đặc trưng phù hợp. Tiếp đến, mô hình VQA sẽ tổng hợp thông tin thu được từ đặc trưng ảnh và ngữ nghĩa trong câu hỏi để đưa ra câu trả lời đúng nhất. Vì vậy, một chương trình VQA có độ chính xác cao cần xây dựng tốt cả hai thành phần này, đặt ra thách thức rất lớn trong việc giải quyết tốt bài toán hỏi đáp trên ảnh.

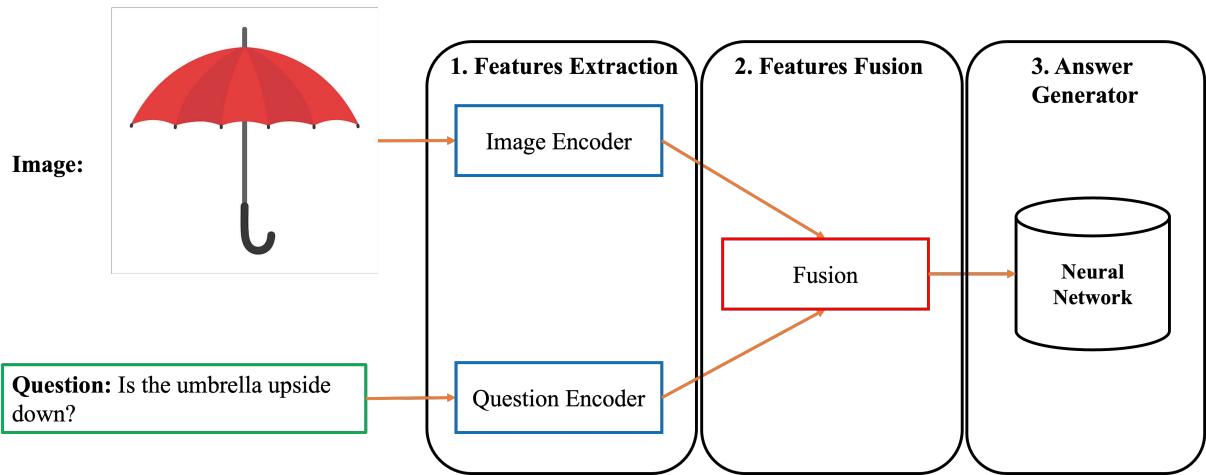


Hình 39.1: Minh họa về bài toán hỏi đáp trên ảnh - Visual Question Answering.

Theo đó, Input và Output của một mô hình VQA sẽ có dạng như sau:

- **Input:** Một cặp hình ảnh và câu hỏi bằng ngôn ngữ tự nhiên.
- **Output:** Câu trả lời cho câu hỏi về hình ảnh.

Ở project này, chúng ta sẽ cùng tìm hiểu và xây dựng một mô hình VQA với hướng tiếp cận cơ bản cũng như truyền thống của bài toán này. Theo đó, kiến trúc tổng quan của mô hình mà ta xây dựng như sau:



Hình 39.2: Minh họa kiến trúc của một mô hình VQA cơ bản.

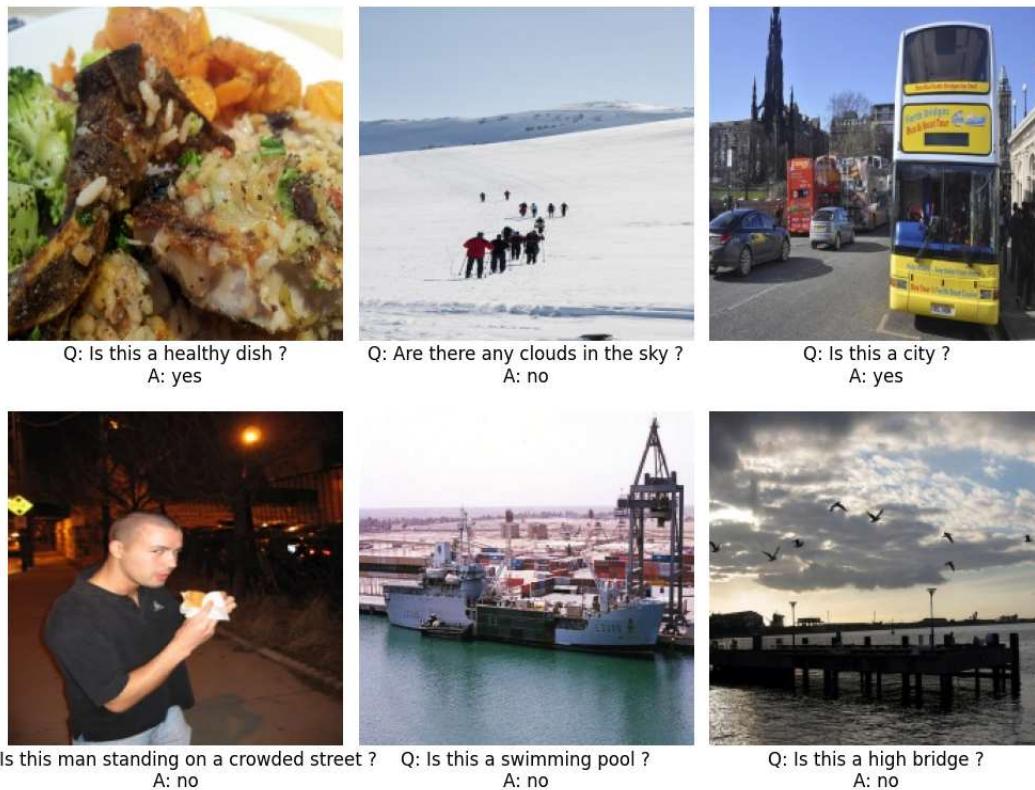
Trong đó, ba thành phần chính gồm:

- 1. Feature Extraction:** Trích xuất các vector đặc trưng phù hợp để đại diện cho dữ liệu ảnh và dữ liệu văn bản.
- 2. Features Fusion:** Với hai đặc trưng từ hai nhánh dữ liệu, áp dụng kỹ thuật kết hợp cả hai loại thông tin này về một vector biểu diễn duy nhất.
- 3. Answer Generator:** Đưa vector đặc trưng kết hợp vào mạng neural để dự đoán câu trả lời.

Dựa theo kiến trúc trên, ta sẽ cài đặt hai phiên bản gồm phiên bản dựa trên các mạng CNN+LSTM và phiên bản dựa trên các mạng thuộc họ transformer, cụ thể là ViT+RoBERTa. Ngoài ra, chúng ta cũng sẽ tìm hiểu hướng tiếp cận đang rất được quan tâm hiện nay, đó là sử dụng mô hình lớn. Cụ thể hơn, ta sẽ cài đặt mô hình lớn để có thể giải bài toán VQA trong project này.

39.2 Cài đặt chương trình

- **CNN + LSTM:** Trong phần này, chúng ta sẽ tiếp cận bài toán bằng cách phối hợp hai mô hình Deep Learning cơ bản dùng trong xử lý ảnh và văn bản là mạng CNN và LSTM.
1. **Tải bộ dữ liệu:** Cho bộ dữ liệu về Visual Question Answering có nội dung quan đến câu hỏi đáp dạng Yes/No, các bạn tải bộ dữ liệu **vqa_coco_dataset.zip** trong đường dẫn thuộc mục Datasets tại phần 60.4 Một số mẫu từ bộ dữ liệu khi được trực quan hóa sẽ có dạng như hình bên dưới:



Hình 39.3: Một vài mẫu dữ liệu trong bộ dữ liệu VQA dạng câu hỏi Yes/No.

2. **Import các thư viện cần thiết:** Chúng ta sẽ sử dụng thư viện

PyTorch để xây dựng và huấn luyện mô hình deep learning. Thêm vào đó, vì làm việc liên quan đến dữ liệu ảnh và text, chúng ta sẽ sử dụng thư viện PIL cho ảnh và spacy, nltk cho text:

```

1 import torch
2 import torch.nn as nn
3 import torchtext
4 import os
5 import random
6 import numpy as np
7 import pandas as pd
8 import spacy
9 import timm
10 import matplotlib.pyplot as plt
11
12 from PIL import Image
13 from torch.utils.data import Dataset, DataLoader
14 from sklearn.model_selection import train_test_split
15 from torchtext.data.utils import get_tokenizer
16 from torchtext.vocab import build_vocab_from_iterator
17 from torchvision import transforms
18

```

3. Cài đặt giá trị ngẫu nhiên cố định: Để có thể tái tạo lại kết quả huấn luyện mô hình, ta sẽ cài đặt một giá trị ngẫu nhiên cố định cho các thư viện có liên quan đến việc tạo các giá trị ngẫu nhiên:

```

1 def set_seed(seed):
2     random.seed(seed)
3     np.random.seed(seed)
4     torch.manual_seed(seed)
5     torch.cuda.manual_seed(seed)
6     torch.cuda.manual_seed_all(seed)
7     torch.backends.cudnn.deterministic = True
8     torch.backends.cudnn.benchmark = False
9
10 seed = 59
11 set_seed(seed)

```

4. Chia bộ dữ liệu train, val, test: Vì bộ dữ liệu này đã được chia sẵn thành train, val, test, ta chỉ cần đọc dữ liệu lên từ file .txt cho trước từ bộ dữ liệu như sau:

```

1 # Load train data
2 train_data = []

```

```
3 train_set_path = './vaq2.0.TrainImages.txt'
4
5 with open(train_set_path, "r") as f:
6     lines = f.readlines()
7     for line in lines:
8         temp = line.split('\t')
9         qa = temp[1].split('?')
10
11     if len(qa) == 3:
12         answer = qa[2].strip()
13     else:
14         answer = qa[1].strip()
15
16     data_sample = {
17         'image_path': temp[0][-2],
18         'question': qa[0] + '?',
19         'answer': answer
20     }
21     train_data.append(data_sample)
22
23 # Load val data
24 val_data = []
25 val_set_path = './vaq2.0.DevImages.txt'
26
27 with open(val_set_path, "r") as f:
28     lines = f.readlines()
29     for line in lines:
30         temp = line.split('\t')
31         qa = temp[1].split('?')
32
33     if len(qa) == 3:
34         answer = qa[2].strip()
35     else:
36         answer = qa[1].strip()
37
38     data_sample = {
39         'image_path': temp[0][-2],
40         'question': qa[0] + '?',
41         'answer': answer
42     }
43     val_data.append(data_sample)
44
45 # Load test data
46 test_data = []
47 test_set_path = './vaq2.0.TestImages.txt'
```

```

48
49 with open(test_set_path, "r") as f:
50     lines = f.readlines()
51     for line in lines:
52         temp = line.split('\t')
53         qa = temp[1].split('?')
54
55         if len(qa) == 3:
56             answer = qa[2].strip()
57         else:
58             answer = qa[1].strip()
59
60         data_sample = {
61             'image_path': temp[0][:-2],
62             'question': qa[0] + '?',
63             'answer': answer
64         }
65         test_data.append(data_sample)
66

```

5. **Xây dựng bộ từ vựng:** Chúng ta cần tiên xử lý text bằng cách biến đổi câu hỏi đầu vào thành các token bằng thư viện spacy và xây dựng bộ từ vựng cho model:

```

1 eng = spacy.load("en_core_web_sm")
2
3 def get_tokens(data_iter):
4     for sample in data_iter:
5         question = sample['question']
6         yield [token.text for token in eng.tokenizer(
7             question)]
8
9 vocab = build_vocab_from_iterator(
10     get_tokens(train_data),
11     min_freq=2,
12     specials=['<pad>', '<sos>', '<eos>', '<unk>'],
13     special_first=True
14 )
15 vocab.set_default_index(vocab['<unk>'])

```

6. **Xây dựng dictionary mapping classes:** Để thuận tiện trong việc chuyển đổi tên class (trong trường hợp này gồm 2 class là "yes" và "no"), ta tạo dictionary dùng để chuyển tên class thành mã số tương ứng và ngược lại. Cách làm như sau:

```

1 classes = set([sample['answer'] for sample in
2     train_data])
3 label2idx = {
4     cls_name: idx for idx, cls_name in enumerate(
5         classes)
6 }
7 idx2label = {
8     idx: cls_name for idx, cls_name in enumerate(
9         classes)
10 }
```

7. **Xây dựng hàm tokenize:** Sau khi đã có bộ từ vựng cho model, ta xây dựng một hàm tokenize để biến câu hỏi đầu vào thành danh sách các token tương ứng trong bộ từ vựng:

```

1 def tokenize(question, max_seq_len):
2     tokens = [token.text for token in eng.tokenizer(
3         question)]
4     sequence = [vocab[token] for token in tokens]
5     if len(sequence) < max_seq_len:
6         sequence += [vocab['<pad>']] * (max_seq_len -
7             len(sequence))
8     else:
9         sequence = sequence[:max_seq_len]
10
11 return sequence
```

8. **Xây dựng class PyTorch dataset:** Chúng ta xây dựng class datasets cho bộ dữ liệu VQA như sau:

```

1 class VQADataset(Dataset):
2     def __init__(
3         self,
4         data,
5         label2idx,
6         max_seq_len=20,
7         transform=None,
8         img_dir='val2014-resized/'
9     ):
10         self.transform = transform
11         self.data = data
12         self.max_seq_len = max_seq_len
13         self.img_dir = img_dir
14         self.label2idx = label2idx
```

```

15
16     def __len__(self):
17         return len(self.data)
18
19     def __getitem__(self, index):
20         img_path = os.path.join(self.img_dir, self.
21         data[index]['image_path'])
22         img = Image.open(img_path).convert('RGB')
23         if self.transform:
24             img = self.transform(img)
25
26         question = self.data[index]['question']
27         question = tokenize(question, self.
28         max_seq_len)
29         question = torch.tensor(question, dtype=torch
30         .long)
31
32         label = self.data[index]['answer']
33         label = label2idx[label]
34         label = torch.tensor(label, dtype=torch.long)
35
36         return img, question, label

```

9. Xây dựng PyTorch transform: Để dữ liệu ảnh đầu vào được đồng bộ về kích thước cũng như thực hiện một số các phép tăng cường dữ liệu ảnh lên bộ dữ liệu, chúng ta sẽ khai báo PyTorch transforms như sau:

```

1 data_transform = {
2     'train': transforms.Compose([
3         transforms.Resize(size=(224, 224)),
4         transforms.CenterCrop(size=180),
5         transforms.ColorJitter(brightness=0.1,
6             contrast=0.1, saturation=0.1),
7         transforms.RandomHorizontalFlip(),
8         transforms.GaussianBlur(3),
9         transforms.ToTensor(),
10        transforms.Normalize((0.485, 0.456, 0.406),
11        (0.229, 0.224, 0.225)),
12    ]),
13    'val': transforms.Compose([
14        transforms.Resize(size=(224, 224)),
15        transforms.ToTensor(),
16        transforms.Normalize((0.485, 0.456, 0.406),
17        (0.229, 0.224, 0.225)),
18    ])
19}

```

```

15     ])
16 }
17

```

10. Khai báo datasets object cho ba bộ train, val, test:

```

1 train_dataset = VQADataset(
2     train_data,
3     label2idx=label2idx,
4     transform=data_transform['train']
5 )
6 val_dataset = VQADataset(
7     val_data,
8     label2idx=label2idx,
9     transform=data_transform['val']
10 )
11 test_dataset = VQADataset(
12     test_data,
13     label2idx=label2idx,
14     transform=data_transform['val']
15 )

```

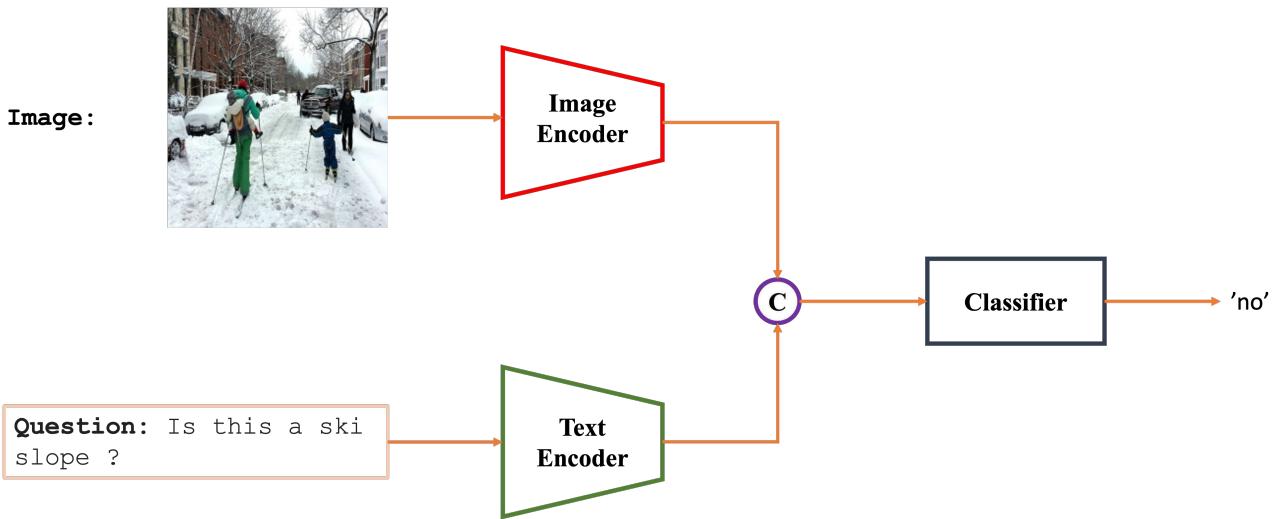
11. Khai báo DataLoader: Với ba object datasets trên, ta khai báo giá trị batch size và tạo dataloader như sau:

```

1 train_batch_size = 256
2 test_batch_size = 32
3
4 train_loader = DataLoader(
5     train_dataset,
6     batch_size=train_batch_size,
7     shuffle=True
8 )
9 val_loader = DataLoader(
10    val_dataset,
11    batch_size=test_batch_size,
12    shuffle=False
13 )
14 test_loader = DataLoader(
15    test_dataset,
16    batch_size=test_batch_size,
17    shuffle=False
18 )

```

12. Xây dựng model: Trong phần này, ta sẽ dùng kiến trúc ResNet cho phần xử lý ảnh và kiến trúc BiLSTM cho phần xử lý text. Hướng tiếp cận này có thể được mô tả qua ảnh sau:



Hình 39.4: Ảnh mô tả pipeline cho hướng tiếp cận sử dụng mô hình CNN để trích xuất đặc trưng ảnh và mô hình Bi-LSTM để trích xuất đặc trưng văn bản.

Cụ thể, đối với ResNet, ta dùng thư viện `timm` để load kiến trúc ResNet18. Code của class model có nội dung như sau:

```

1  class VQAModel(nn.Module):
2      def __init__(self,
3          n_classes,
4          img_model_name,
5          embedding_dim,
6          n_layers=2,
7          hidden_size=256,
8          drop_p=0.2
9      ):
10         super(VQAModel, self).__init__()
11         self.image_encoder = timm.create_model(
12             img_model_name,
13             pretrained=True,
14             num_classes=hidden_size
15         )
16
17         for param in self.image_encoder.parameters():
18             param.requires_grad = True
19
20

```

```

21         self.embedding = nn.Embedding(len(vocab),
22                                         embeddding_dim)
23         self.lstm1 = nn.LSTM(
24             input_size=embeddding_dim,
25             hidden_size=hidden_size,
26             num_layers=n_layers,
27             batch_first=True,
28             bidirectional=True,
29             dropout=drop_p
30         )
31
32         self.fc1 = nn.Linear(hidden_size * 3,
33                             hidden_size)
34         self.dropout = nn.Dropout(drop_p)
35         self.gelu = nn.GELU()
36         self.fc2 = nn.Linear(hidden_size, n_classes)
37
38     def forward(self, img, text):
39         img_features = self.image_encoder(img)
40
41         text_emb = self.embedding(text)
42         lstm_out, _ = self.lstm1(text_emb)
43
44         lstm_out = lstm_out[:, -1, :]
45
46         combined = torch.cat((img_features, lstm_out),
47                               dim=1)
48         x = self.fc1(combined)
49         x = self.gelu(x)
50         x = self.dropout(x)
51         x = self.fc2(x)

52
53     return x

```

Sau khi định nghĩa class, ta tiến hành khai báo model như sau:

```

1 n_classes = len(classes)
2 img_model_name = 'resnet18'
3 hidden_size = 256
4 n_layers = 2
5 embeddding_dim = 128
6 drop_p = 0.2
7 device = 'cuda' if torch.cuda.is_available() else 'cpu'
8

```

```

9 model = VQAModel(
10     n_classes=n_classes,
11     img_model_name=img_model_name,
12     embeddding_dim=embeddding_dim,
13     n_layers=n_layers,
14     hidden_size=hidden_size,
15     drop_p=drop_p
16 ).to(device)
17

```

Lưu ý rằng, trong trường hợp của bài toán VQA, việc thay đổi các giá trị tham số mô hình sẽ ảnh hưởng rất lớn đến kết quả của mô hình. Các giá trị gán mặc định trong code đã được tinh chỉnh cho phù hợp. Vì vậy, nếu muốn thay đổi, các bạn cần để ý để điều chỉnh cho phù hợp. Điều này cũng áp dụng tương tự cho các tham số thuộc phần optimizer.

13. **Xây dựng hàm train và evaluate:** Để huấn luyện mô hình, ta cần xây dựng hàm huấn luyện và đánh giá mô hình như sau:

```

1 def evaluate(model, dataloader, criterion, device):
2     model.eval()
3     correct = 0
4     total = 0
5     losses = []
6     with torch.no_grad():
7         for image, question, labels in dataloader:
8             image, question, labels = image.to(device),
9             question.to(device), labels.to(device)
10            outputs = model(image, question)
11            loss = criterion(outputs, labels)
12            losses.append(loss.item())
13            _, predicted = torch.max(outputs.data, 1)
14            total += labels.size(0)
15            correct += (predicted == labels).sum().
16            item()
17
18    loss = sum(losses) / len(losses)
19    acc = correct / total
20
21 def fit(
22     model,
23     train_loader,

```

```
24     val_loader,
25     criterion,
26     optimizer,
27     scheduler,
28     device,
29     epochs
30 ):
31     train_losses = []
32     val_losses = []
33
34     for epoch in range(epochs):
35         batch_train_losses = []
36
37         model.train()
38         for idx, (images, questions, labels) in
39             enumerate(train_loader):
40             images = images.to(device)
41             questions = questions.to(device)
42             labels = labels.to(device)
43
44             optimizer.zero_grad()
45             outputs = model(images, questions)
46             loss = criterion(outputs, labels)
47             loss.backward()
48             optimizer.step()
49
50             batch_train_losses.append(loss.item())
51
52             train_loss = sum(batch_train_losses) / len(
53                 batch_train_losses)
54             train_losses.append(train_loss)
55
56             val_loss, val_acc = evaluate(
57                 model, val_loader,
58                 criterion, device
59             )
60             val_losses.append(val_loss)
61
62             print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal loss: {val_loss:.4f}\tVal
63             Acc: {val_acc}')
64
65             scheduler.step()
66
67     return train_losses, val_losses
```

65

14. Khai báo hàm loss, optimizer và scheduler:

```
1 lr = 1e-3
2 epochs = 50
3
4 scheduler_step_size = epochs * 0.8
5 criterion = nn.CrossEntropyLoss()
6
7 optimizer = torch.optim.Adam(
8     model.parameters(),
9     lr=lr
10)
11 scheduler = torch.optim.lr_scheduler.StepLR(
12     optimizer,
13     step_size=scheduler_step_size,
14     gamma=0.1
15)
16
```

15. Training: Tổng hợp tất cả các thành phần trên, ta gọi hàm `fit()` để bắt đầu quá trình huấn luyện mô hình VQA:

```
1 train_losses, val_losses = fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     scheduler,
8     device,
9     epochs
10)
11
```

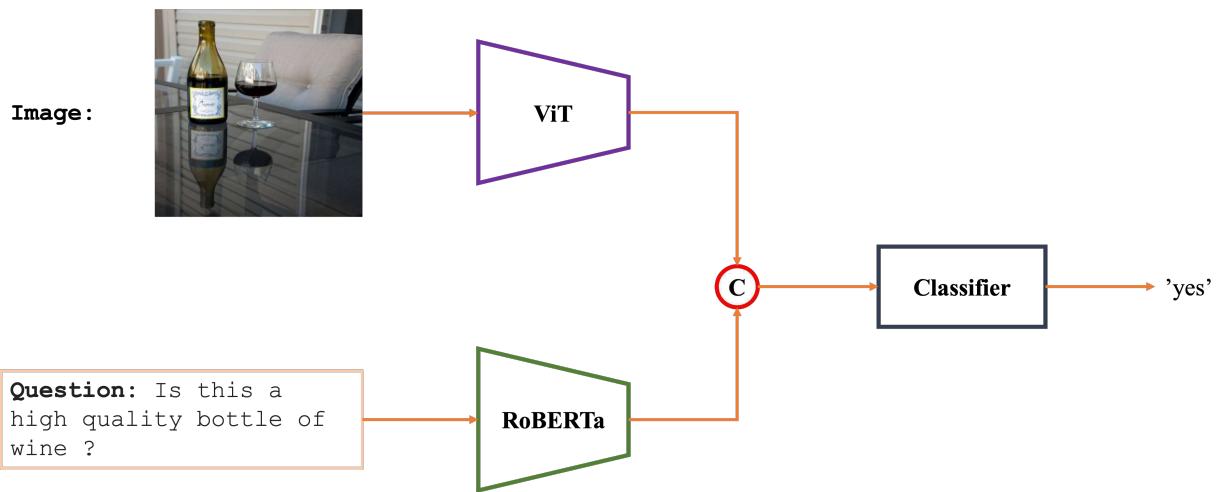
16. Evaluation: Với mô hình đã huấn luyện, ta đưa vào đánh giá trên hai tập val và test như sau:

```
1 val_loss, val_acc = evaluate(
2     model,
3     val_loader,
4     criterion,
5     device
6)
7 test_loss, test_acc = evaluate(
```

```
8     model ,
9     test_loader ,
10    criterion ,
11    device
12 )
13
14 print('Evaluation on val/test dataset')
15 print('Val accuracy: ', val_acc)
16 print('Test accuracy: ', test_acc)
17
18 ## Evaluation results from author
19 # Val accuracy: 0.4989754098360656
20 # Test accuracy: 0.5084075173095944
21
```

Có thể thấy rằng, với hướng tiếp cận cơ bản trên, mô hình VQA chúng ta xây dựng không đạt được kết quả khả quan. Vì vậy ở phần sau, ta sẽ thử sử dụng các mô hình tốt hơn để cải thiện kết quả bài toán này.

- **VisionTransformers + RoBERTa:** Trong phần này, chúng ta sẽ tiếp cận bài toán bằng cách phối hợp hai mô hình Deep Learning thuộc họ mô hình Transformer, một kiến trúc mạng cực kì mạnh mẽ đang được sử dụng rộng rãi tại thời điểm này. Cụ thể, với ảnh đầu vào, ta dùng VisionTransformer (ViT). Đối với câu hỏi, ta dùng RoBERTa. Từ đó, trong mô hình VQA, dựa trên kết quả từ hai mô hình, ta sẽ kết hợp chúng để dự đoán câu trả lời. Hướng tiếp cận này có thể được mô tả như hình sau:



Hình 39.5: Ảnh mô tả pipeline cho hướng tiếp cận sử dụng các mô hình thuộc họ Transformer, cụ thể trong trường hợp này là ViT và RoBERTa.

Chúng ta sẽ triển khai hướng tiếp cận này thông qua các bước thực hiện dưới đây:

1. **Import các thư viện cần thiết:** Đối với transformer, ta sẽ sử dụng thư viện HuggingFace để gọi các mô hình và một số hàm xử lý cần thiết:

```

1 import torch
2 import torch.nn as nn
3 import os
4 import random
5 import numpy as np
6 import pandas as pd

```

```
7 import matplotlib.pyplot as plt
8
9 from PIL import Image
10 from torch.utils.data import Dataset, DataLoader
11 from torchvision import transforms
12 from transformers import ViTModel, ViTImageProcessor
13 from transformers import AutoTokenizer, RobertaModel
14
```

2. Cài đặt giá trị ngẫu nhiên cố định:

```
1 def set_seed(seed):
2     random.seed(seed)
3     np.random.seed(seed)
4     torch.manual_seed(seed)
5     torch.cuda.manual_seed(seed)
6     torch.cuda.manual_seed_all(seed)
7     torch.backends.cudnn.deterministic = True
8     torch.backends.cudnn.benchmark = False
9
10 seed = 59
11 set_seed(seed)
```

3. Chia bộ dữ liệu train, val, test:

```
1 # Load train data
2 train_data = []
3 train_set_path = './vaq2.0.TrainImages.txt'
4
5 with open(train_set_path, "r") as f:
6     lines = f.readlines()
7     for line in lines:
8         temp = line.split('\t')
9         qa = temp[1].split('?')
10
11     if len(qa) == 3:
12         answer = qa[2].strip()
13     else:
14         answer = qa[1].strip()
15
16     data_sample = {
17         'image_path': temp[0][-2],
18         'question': qa[0] + '?',
19         'answer': answer
20     }
21     train_data.append(data_sample)
```

```
22
23 # Load val data
24 val_data = []
25 val_set_path = './vaq2.0.DevImages.txt'
26
27 with open(val_set_path, "r") as f:
28     lines = f.readlines()
29     for line in lines:
30         temp = line.split('\t')
31         qa = temp[1].split('?')
32
33         if len(qa) == 3:
34             answer = qa[2].strip()
35         else:
36             answer = qa[1].strip()
37
38         data_sample = {
39             'image_path': temp[0][-2] ,
40             'question': qa[0] + '?',
41             'answer': answer
42         }
43         val_data.append(data_sample)
44
45 # Load test data
46 test_data = []
47 test_set_path = './vaq2.0.TestImages.txt'
48
49 with open(test_set_path, "r") as f:
50     lines = f.readlines()
51     for line in lines:
52         temp = line.split('\t')
53         qa = temp[1].split('?')
54
55         if len(qa) == 3:
56             answer = qa[2].strip()
57         else:
58             answer = qa[1].strip()
59
60         data_sample = {
61             'image_path': temp[0][-2] ,
62             'question': qa[0] + '?',
63             'answer': answer
64         }
65         test_data.append(data_sample)
66
```

4. Xây dựng dictionary mapping classes:

```
1 classes = set([sample['answer'] for sample in
2     train_data])
3 label2idx = {
4     cls_name: idx for idx, cls_name in enumerate(
5         classes)
6 }
7 idx2label = {
8     idx: cls_name for idx, cls_name in enumerate(
9         classes)
10 }
```

5. **Xây dựng class PyTorch dataset:** Với việc sử dụng hai mô hình liên quan đến transformers, chúng ta sẽ có một chút sự thay đổi trong việc triển khai class dataset như sau:

```
1 class VQADataset(Dataset):
2     def __init__(self,
3                  data,
4                  label2idx,
5                  img_feature_extractor,
6                  text_tokenizer,
7                  device,
8                  transforms=None,
9                  img_dir='val2014-resized'
10                 ):
11         self.data = data
12         self.img_dir = img_dir
13         self.label2idx = label2idx
14         self.img_feature_extractor =
15             img_feature_extractor
16         self.text_tokenizer = text_tokenizer
17         self.device = device
18         self.transforms = transforms
19
20     def __len__(self):
21         return len(self.data)
22
23     def __getitem__(self, index):
24         img_path = os.path.join(self.img_dir, self.
data[index]['image_path'])
25         img = Image.open(img_path).convert('RGB')
26
```

```

27         if self.transforms:
28             img = self.transforms(img)
29
30         if self.img_feature_extractor:
31             img = self.img_feature_extractor(images=
32                 img, return_tensors="pt")
33                 img = {k: v.to(self.device).squeeze(0)}
34
35         question = self.data[index]['question']
36         if self.text_tokenizer:
37             question = self.text_tokenizer(
38                 question,
39                 padding="max_length",
40                 max_length=20,
41                 truncation=True,
42                 return_tensors="pt"
43             )
44             question = {k: v.to(self.device).squeeze
45 (0) for k, v in question.items()}
46
47         label = self.data[index]['answer']
48         label = torch.tensor(
49             self.label2idx[label],
50             dtype=torch.long
51         ).to(self.device)
52
53         sample = {
54             'image': img,
55             'question': question,
56             'label': label
57         }
58
59         return sample

```

Ở đây, ta bổ sung vào hai thành phần mới gồm `img_feature_extractor` và `text_tokenizer`, đây là hai hàm dùng để tiền xử lý dữ liệu đầu vào dành riêng cho VisionTransformers và RoBERTa. Vì kết quả của hai hàm này là tensor, chúng ta cũng cần truyền vào tham số `device` để chuyển đổi tensor về format tính toán của máy.

6. Xây dựng PyTorch transform:

```

1 data_transform = transforms.Compose([

```

```
2     transforms.Resize(size=(224, 224)),
3     transforms.CenterCrop(size=180),
4     transforms.ColorJitter(brightness=0.1, contrast
5         =0.1, saturation=0.1),
6     transforms.RandomHorizontalFlip(),
7     transforms.GaussianBlur(3),
8 ])
```

7. Khai báo datasets object cho ba bộ train, val, test:

```
1 img_feature_extractor = ViTImageProcessor(
2     from_pretrained("google/vit-base-patch16-224")
3 text_tokenizer = AutoTokenizer.from_pretrained(
4     "roberta-base")
5 device = 'cuda' if torch.cuda.is_available() else 'cpu'
6
7 train_dataset = VQADataset(
8     train_data,
9     label2idx=label2idx,
10    img_feature_extractor=img_feature_extractor,
11    text_tokenizer=text_tokenizer,
12    device=device,
13    transforms=data_transform
14 )
15 val_dataset = VQADataset(
16     val_data,
17     label2idx=label2idx,
18     img_feature_extractor=img_feature_extractor,
19     text_tokenizer=text_tokenizer,
20     device=device
21 )
22 test_dataset = VQADataset(
23     test_data,
24     label2idx=label2idx,
25     img_feature_extractor=img_feature_extractor,
26     text_tokenizer=text_tokenizer,
27     device=device
28 )
```

8. Khai báo DataLoader:

```
1 train_batch_size = 256
2 test_batch_size = 32
3
```

```

4 train_loader = DataLoader(
5     train_dataset,
6     batch_size=train_batch_size,
7     shuffle=True
8 )
9 val_loader = DataLoader(
10    val_dataset,
11    batch_size=test_batch_size,
12    shuffle=False
13 )
14 test_loader = DataLoader(
15    test_dataset,
16    batch_size=test_batch_size,
17    shuffle=False
18 )

```

9. Xây dựng model: Ta sẽ chia mô hình thành 3 phần gồm TextEncoder, VisualEncoder và Classifier. Sau đó, ta kết hợp ba thành phần này lại để trở thành một mô hình hoàn chỉnh. Code triển khai như sau:

– **TextEncoder:**

```

1 class TextEncoder(nn.Module):
2     def __init__(self):
3         super(TextEncoder, self).__init__()
4         self.model = RobertaModel.from_pretrained(
5             "roberta-base")
6
7     def forward(self, inputs):
8         outputs = self.model(**inputs)
9
10    return outputs.pooler_output

```

– **VisualEncoder:**

```

1 class VisualEncoder(nn.Module):
2     def __init__(self):
3         super(VisualEncoder, self).__init__()
4         self.model = ViTModel.from_pretrained(
5             "google/vit-base-patch16-224")
6
7     def forward(self, inputs):
8         outputs = self.model(**inputs)
9
10    return outputs.pooler_output

```

– Classifier:

```

1  class Classifier(nn.Module):
2      def __init__(self,
3                  hidden_size=512,
4                  dropout_prob=0.2,
5                  n_classes=2
6      ):
7          super(Classifier, self).__init__()
8          self.fc1 = nn.Linear(768 * 2, hidden_size)
9
10         self.dropout = nn.Dropout(dropout_prob)
11         self.gelu = nn.GELU()
12         self.fc2 = nn.Linear(hidden_size,
13                             n_classes)
14
15     def forward(self, x):
16         x = self.fc1(x)
17         x = self.gelu(x)
18         x = self.dropout(x)
19         x = self.fc2(x)
20
21     return x

```

– VQAModel: Tổng hợp lại 3 thành phần trên, ta được mô hình VQA hoàn chỉnh như sau:

```

1  class VQAModel(nn.Module):
2      def __init__(self,
3                  visual_encoder,
4                  text_encoder,
5                  classifier
6      ):
7          super(VQAModel, self).__init__()
8          self.visual_encoder = visual_encoder
9          self.text_encoder = text_encoder
10         self.classifier = classifier
11
12
13     def forward(self, image, answer):
14         text_out = self.text_encoder(answer)
15         image_out = self.visual_encoder(image)
16
17

```

```

18         x = torch.cat((image_out, text_out), dim
19 =1)
20         x = self.classifier(x)
21
22     return x
23
24     def freeze(self, visual=True, textual=True,
25 clas=False):
26         if visual:
27             for n,p in self.visual_encoder.
28             named_parameters():
29                 p.requires_grad = False
30         if textual:
31             for n,p in self.text_encoder.
32             named_parameters():
33                 p.requires_grad = False

```

Các bạn có thể thấy trong phần code này, chúng ta có triển khai thêm hàm `freeze()`. Hàm này có chức năng dùng để đóng băng các tham số của những pre-trained model, cụ thể ở đây là một trong ba thành phần chính của mô hình. Trong bài này, chúng ta sẽ chỉ cập nhật trọng số cho phần Classifier, các phần khác chúng ta sẽ đóng băng lại.

Sau khi định nghĩa xong, ta khai báo mô hình VQA:

```

1 n_classes = len(classes)
2 hidden_size = 256
3 dropout_prob = 0.2
4
5 text_encoder = TextEncoder().to(device)
6 visual_encoder = VisualEncoder().to(device)
7 classifier = Classifier(
8     hidden_size=hidden_size,
9     dropout_prob=dropout_prob,
10    n_classes=n_classes
11 ).to(device)
12
13 model = VQAModel(
14     visual_encoder=visual_encoder,
15     text_encoder=text_encoder,

```

```
16     classifier=classifier
17 ).to(device)
18 model.freeze()
19
```

10. **Xây dựng hàm train và evaluate:** Ta xây dựng hàm train và evaluate. Ở phần này, nội dung code của hai hàm sẽ có đôi chút sự khác biệt trong việc lấy dữ liệu từ dataloader vì chúng ta đã thay đổi cấu trúc data lúc định nghĩa PyTorch dataset:

```
1 def evaluate(model, dataloader, criterion):
2     model.eval()
3     correct = 0
4     total = 0
5     losses = []
6     with torch.no_grad():
7         for idx, inputs in enumerate(dataloader):
8             images = inputs['image']
9             questions = inputs['question']
10            labels = inputs['label']
11            outputs = model(images, questions)
12            loss = criterion(outputs, labels)
13            losses.append(loss.item())
14            _, predicted = torch.max(outputs.data, 1)
15            total += labels.size(0)
16            correct += (predicted == labels).sum().
17                         item()
18
19            loss = sum(losses) / len(losses)
20            acc = correct / total
21
22    return loss, acc
23
24 def fit(
25     model,
26     train_loader,
27     val_loader,
28     criterion,
29     optimizer,
30     scheduler,
31     epochs
32 ) :
33     train_losses = []
34     val_losses = []
```

```

35     for epoch in range(epochs):
36         batch_train_losses = []
37
38         model.train()
39         for idx, inputs in enumerate(train_loader):
40             images = inputs['image']
41             questions = inputs['question']
42             labels = inputs['label']
43
44             optimizer.zero_grad()
45             outputs = model(images, questions)
46             loss = criterion(outputs, labels)
47             loss.backward()
48             optimizer.step()
49
50             batch_train_losses.append(loss.item())
51
52             train_loss = sum(batch_train_losses) / len(
batch_train_losses)
53             train_losses.append(train_loss)
54
55             val_loss, val_acc = evaluate(
56                 model, val_loader,
57                 criterion
58             )
59             val_losses.append(val_loss)
60
61             print(f'EPOCH {epoch + 1}:\tTrain loss: {train_loss:.4f}\tVal loss: {val_loss:.4f}\tVal Acc: {val_acc}')
62
63             scheduler.step()
64
65     return train_losses, val_losses
66

```

11. Khai báo hàm loss, optimizer và scheduler:

```

1 lr = 1e-3
2 epochs = 50
3 scheduler_step_size = epochs * 0.8
4 criterion = nn.CrossEntropyLoss()
5
6 optimizer = torch.optim.Adam(
7     model.parameters(),
8     lr=lr

```

```

9 )
10 scheduler = torch.optim.lr_scheduler.StepLR(
11     optimizer,
12     step_size=scheduler_step_size,
13     gamma=0.1
14 )
15

```

12. **Training:** Với tất cả các thành phần có được, ta tiến hành huấn luyện mô hình:

```

1 train_losses, val_losses = fit(
2     model,
3     train_loader,
4     val_loader,
5     criterion,
6     optimizer,
7     scheduler,
8     epochs
9 )
10

```

13. **Evaluation:** Cuối cùng, ta đánh giá mô hình đã huấn luyện được trên hai tập val và test:

```

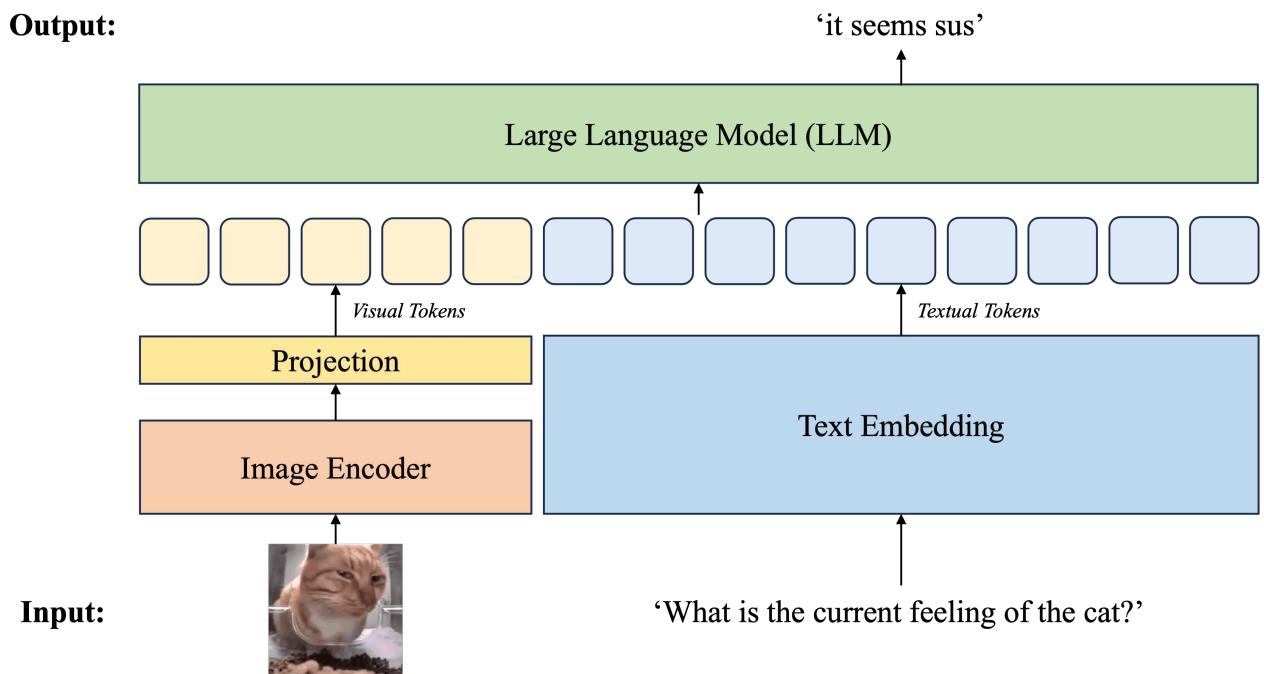
1 val_loss, val_acc = evaluate(
2     model,
3     val_loader,
4     criterion
5 )
6 test_loss, test_acc = evaluate(
7     model,
8     test_loader,
9     criterion
10 )
11
12 print('Evaluation on val/test dataset')
13 print('Val accuracy: ', val_acc)
14 print('Test accuracy: ', test_acc)
15
16 ## Evaluation results from author
17 # Val accuracy: 0.6664959016393442
18 # Test accuracy: 0.6533135509396637
19

```

Như vậy, ta có thể thấy một sự cải thiện rõ rệt của mô hình khi sử dụng các mô hình thuộc họ transformer để làm encoder cho hai

kênh dữ liệu. Dù rằng chưa đạt được kết quả tốt nhất, song điều này cũng thể hiện được phần nào tiềm năng và hiệu quả mà các mô hình transformer mang lại trên bài toán VQA.

- **Vision-Language Models (VLMs):** Mô hình thị giác ngôn ngữ - Vision Language Models (VLMs) là thuật ngữ chỉ loại mô hình deep learning có khả năng xử lý đồng thời cả dữ liệu ảnh và văn bản. Hiện nay, các mô hình này đang rất được quan tâm bởi sự phát triển của các mô hình ngôn ngữ lớn - Large Language Models (LLMs). Theo đó, dựa trên ý tưởng tận dụng tri thức khổng lồ trong LLMs, các nhóm nghiên cứu hiện nay đã và đang phát triển, xây dựng một mô hình Large Vision Language Models (LVLMs) thông qua việc kết nối thông tin thị giác từ một Vision Encoder vào một LLM đủ tốt. Với cách thức này, các mô hình dạng LVLMs được kỳ vọng có thể giải quyết hầu hết tất cả mọi bài toán không chỉ liên quan đến dữ liệu văn bản mà còn về dữ liệu ảnh như Object Detection, Image Captioning...



Hình 39.6: Một kiến trúc cơ bản của Vision Language Models (VLMs) với hướng tiếp cận kết hợp với mô hình ngôn ngữ lớn (LLMs).

Để hiểu hơn về khả năng của VLM, ở phần này chúng ta sẽ tìm hiểu cách sử dụng một mô hình VLM để giải bài toán VQA. Code cài đặt

nhusau:

1. **Import các thư viện cần thiết:** Tương tự phần transformer, chúng ta cũng sẽ gọi các mô hình thị giác ngôn ngữ lớn trong thư viện HuggingFace:

```
1 import torch
2 import os
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6
7 from transformers import
8     LlavaForConditionalGeneration
9 from transformers import AutoProcessor
10 from transformers import BitsAndBytesConfig
11 from transformers import GenerationConfig
12 from PIL import Image
```

2. **Đọc bộ dữ liệu test:** Ở phần này, chúng ta chỉ thực hiện dự đoán sử dụng mô hình pre-trained VLM. Vì vậy, chúng ta sẽ tận dụng các mẫu dữ liệu trong bộ test của bộ dữ liệu được sử dụng bài project này để đưa vào VLM. Theo đó, ta sẽ đọc dữ liệu lên như sau:

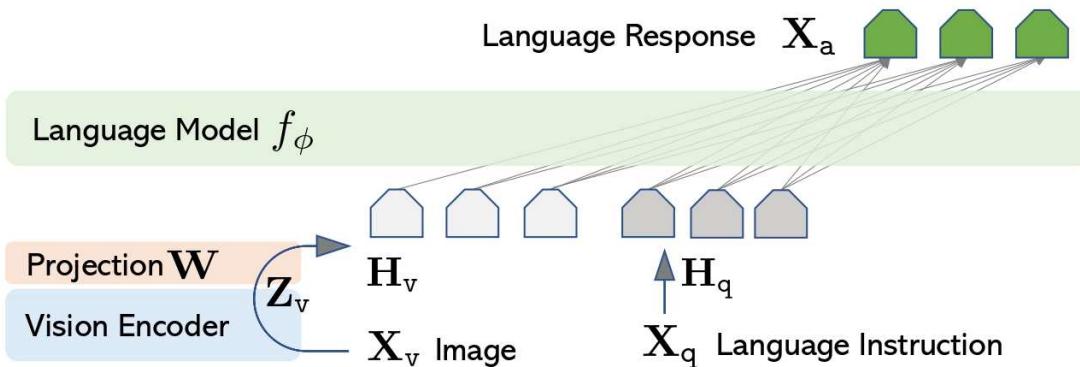
```
1 test_data = []
2 test_set_path = './vaq2.0.TestImages.txt'
3
4 with open(test_set_path, "r") as f:
5     lines = f.readlines()
6     for line in lines:
7         temp = line.split('\t')
8         qa = temp[1].split('?')
9
10        if len(qa) == 3:
11            answer = qa[2].strip()
12        else:
13            answer = qa[1].strip()
14
15        data_sample = {
16            'image_path': temp[0][:-2],
17            'question': qa[0] + '?',
18            'answer': answer
19        }
20        test_data.append(data_sample)
```

3. **Khai báo mô hình VLM:** Ta tải và khai báo mô hình VLM sử dụng đoạn code sau. Trong project này, ta sẽ sử dụng mô hình **LLaVA** với 7 tỷ tham số.

```

1 quantization_config = BitsAndBytesConfig(
2     load_in_4bit=True,
3     bnb_4bit_compute_dtype=torch.float16
4 )
5
6 model_id = "llava-hf/llava-1.5-7b-hf"
7 device = 'cuda' if torch.cuda.is_available() else 'cpu'
8 processor = AutoProcessor.from_pretrained(model_id)
9 model = LlavaForConditionalGeneration.from_pretrained(
10    model_id,
11    quantization_config=quantization_config,
12    device_map=device)

```



Hình 39.7: Kiến trúc của mô hình LLaVA. Nguồn: [link](#).

4. **Xây dựng hàm tạo prompt:** Để sử dụng các mô hình lớn nói chung, chúng ta cần thiết kế và đưa vào mô hình một mô tả văn bản gọi là “prompt”. Có rất nhiều cách để thiết kế một mẫu prompt. Ở project VQA này, ta sẽ sử dụng một mẫu prompt đơn giản như sau:

```

1 def create_prompt(question):
2     prompt = f"""### INSTRUCTION:

```

```

3 Your task is to answer the question based on the
   given image. You can only answer 'yes' or 'no'.
4 ### USER: <image>
5 {question}
6 ### ASSISTANT:""
7     return prompt

```

5. **Khai báo các tham số tạo sinh:** Việc tạo sinh của một mô hình lớn bị ảnh hưởng bởi rất nhiều các tham số khác nhau. Trong bài này, ta sẽ cài đặt một số tham số như sau:

```

1 generation_config = GenerationConfig(
2     max_new_tokens=10,
3     do_sample=True,
4     temperature=0.1,
5     top_p=0.95,
6     top_k=50,
7     eos_token_id=model.config.eos_token_id,
8     pad_token=model.config.pad_token_id,
9 )

```

6. **Thực hiện dự đoán:** Cuối cùng, ta sẽ gọi mô hình để thực hiện dự đoán cho một mẫu dữ liệu trong bộ test của chúng ta bằng đoạn code sau:

```

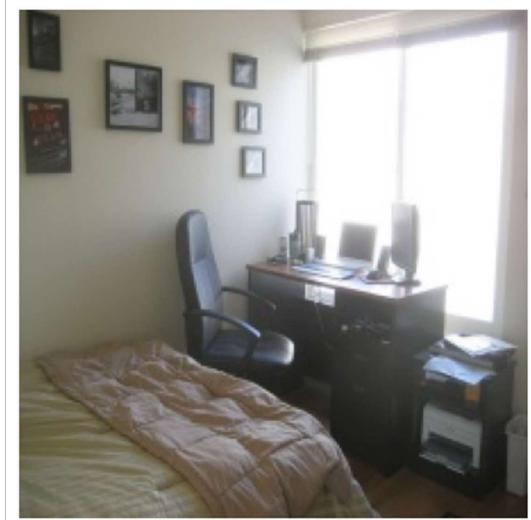
1 idx = 0
2 question = test_data[idx]['question']
3 image_name = test_data[idx]['image_path']
4 image_path = os.path.join('val2014-resized',
   image_name)
5 label = test_data[idx]['answer']
6 image = Image.open(image_path)
7
8 prompt = create_prompt(question)
9 inputs = processor(prompt,
10                     image,
11                     padding=True,
12                     return_tensors="pt").to(device)
13
14 output = model.generate(**inputs,
15                         generation_config=
16                         generation_config)
16 generated_text = processor.decode(output[0],
17                                   skip_special_tokens
18                                   =True)

```

```

19 plt.imshow(image)
20 plt.axis("off")
21 plt.show()
22 print(f"Question: {question}")
23 print(f"Label: {label}")
24 print(f"Prediction: {generated_text.split('###'
ASSISTANT:')[ -1]}")

```



Question: Are there any trees visible ?

Groundtruth: no

Predicted: no

Trained ViT+RoBERTa Model.

Question: Are there any boxes in the room ?

Groundtruth: no

Prediction: No

Pre-trained LLaVA-7B Model.

Hình 39.8: Kết quả dự đoán của mô hình ViT+RoBERTa sau khi huấn luyện vs mô hình pre-trained LLaVA.

Mặc dù không cần phải trải qua thêm bất cứ bước training trên bộ dữ liệu mới hoặc cài đặt gì khác, song mô hình pre-trained LLaVA vẫn hoàn toàn có thể dự đoán chính xác một số mẫu dữ liệu trên bộ test. Điều này cho ta thấy tiềm năng và sức mạnh của các mô hình ngôn

ngữ thị giác không chỉ trong bài toán VQA mà còn liên quan đến các ứng dụng khác trong trí tuệ nhân tạo.

39.3 Câu hỏi trắc nghiệm

1. Mục tiêu của bài toán Visual Question Answering là gì?
 - (a) Để tăng kích cỡ của tấm ảnh.
 - (b) Để trả lời câu hỏi dựa trên hình ảnh.
 - (c) Để tạo sinh ra hình ảnh dựa trên câu mô tả.
 - (d) Để tạo ra mô hình 3D từ hình ảnh 2D.
2. Trong VQA, mô hình thường nhận đầu vào là gì?
 - (a) Chỉ là hình ảnh.
 - (b) Chỉ là câu hỏi văn bản.
 - (c) Hình ảnh và câu hỏi văn bản.
 - (d) Âm thanh.
3. Visual Question Answering là sự kết hợp giữa 2 lĩnh vực nào sau đây?
 - (a) Robotics và Xử lý ngôn ngữ tự nhiên.
 - (b) Thị giác máy tính và học máy.
 - (c) Thị giác máy tính và xử lý ngôn ngữ tự nhiên.
 - (d) Học máy và robotics.
4. Trong VQA, câu trả lời có thể là gì?
 - (a) Chỉ là “có” hoặc “không”.
 - (b) Chỉ là một số nguyên.
 - (c) Có thể là một câu trả lời văn bản hoặc câu trả lời “có” hoặc “không”.
 - (d) Chỉ là một màu sắc.
5. Thành phần nào sau đây là một trong những thành phần chính của một mô hình VQA?
 - (a) Nhận diện giọng nói.
 - (b) Trích xuất đặc trưng từ hình ảnh.

- (c) Dịch câu hỏi ngôn ngữ tự nhiên.
(d) Xử lý âm thanh.
6. Một số thách thức trong VQA bao gồm:
- (a) Hiểu biết ngôn ngữ tự nhiên.
(b) Nhận dạng đối tượng trong hình ảnh.
(c) Không có thách thức nào.
(d) Cả a và b đều đúng.
7. Một hệ thống VQA thường xử lý câu hỏi ngôn ngữ tự nhiên như thế nào?
- (a) Bằng cách chuyển đổi câu hỏi đó thành hình ảnh.
(b) Bằng cách áp dụng sentiment analysis cho tấm ảnh đó.
(c) Dịch câu hỏi đó qua ngôn ngữ khác.
(d) Bằng cách trích xuất đặc trưng câu hỏi đó bằng các kỹ thuật NLP.
8. Đoạn code sau đây dùng để làm gì?
- ```
1 eng = spacy.load("en_core_web_sm")
2
3 def get_tokens(data_iter):
4 for sample in data_iter:
5 question = sample['question']
6 yield [token.text for token in eng.tokenizer(
7 question)]
```
- (a) Để load mô hình tiếng anh Spacy.  
(b) Tạo ra một danh sách các từ dựa trên danh sách token.  
(c) Thêm ký tự đặc biệt vào các câu hỏi.  
(d) Chia văn bản tiếng Anh thành những token.
9. Mục đích của đoạn code sau là gì?

```
1 vocab = build_vocab_from_iterator(
2 get_tokens(train_data),
3 min_freq=2,
4 specials= ['<pad>', '<sos>', '<eos>', '<unk>'],
5 special_first=True
6)
7 vocab.set_default_index(vocab['<unk>'])
8
```

- (a) Xây dựng từ vựng từ tập train, bao gồm các token đặc biệt và token mặc định <unk> cho các từ chưa biết.
- (b) Dịch tập data huấn luyện sang một ngôn ngữ khác bằng cách sử dụng các token đặc biệt.
- (c) Sắp xếp các từ trong tập huấn luyện dựa trên tần suất xuất hiện của các từ.
- (d) Đào tạo một mô hình mới để hiểu các mẫu ngôn ngữ trong tập data huấn luyện.
10. Mô hình học máy nào thường được sử dụng cho bài toán VQA?
- (a) Mô hình dựa trên luật có sẵn.
- (b) Các mô hình CNN.
- (c) Mô hình linear regression.
- (d) Mô hình Decision Tree.
11. Thách thức trong bài toán VQA là gì?
- (a) Tăng độ phân giải của hình ảnh.
- (b) Sự mơ hồ trong câu hỏi ngôn ngữ tự nhiên.
- (c) Xử lý âm thanh.
- (d) Tối ưu hóa phần cứng.
12. Output của mô hình VQA là:
- (a) Bản báo cáo chi tiết.
- (b) Con số cụ thể.
- (c) Câu trả lời cho câu hỏi về hình ảnh.

(d) Một đồ thị.

13. Hệ thống VQA được ứng dụng trong việc:

- (a) Chơi các video game.
- (b) Hỗ trợ người dùng khiếm thị trong việc hiểu môi trường xung quanh.
- (c) Lọc email rác.
- (d) Xử lý âm thanh.

14. Dòng code sau đây có tác dụng gì?

```
1 image_encoder = timm.create_model(image_model, pretrained
 =True, num_classes=hidden_dim))
2
```

- (a) Tải pretrained hình ảnh từ thư viện timm.
- (b) Chuyển đổi dữ liệu hình ảnh màu sang hình ảnh xám.
- (c) Tăng độ phân giải của hình ảnh.
- (d) Nén hình ảnh để xử lý nhanh hơn.

15. Output của image\_encoder trong đoạn code sau là gì?

```
1 image_encoder = timm.create_model(image_model, pretrained
 =True, num_classes=hidden_dim))
2
```

- (a) Ảnh được biến đổi từ tập dữ liệu.
- (b) Một vector embedding với số chiều là hidden\_dim.
- (c) Xác suất tấm ảnh đầu vào thuộc về từng class.
- (d) Chiều của tấm ảnh đầu vào.

16. Điều KHÔNG làm ảnh hưởng đến độ chính xác của một mô hình VQA?

- (a) Tốc độ Internet.
- (b) Chất lượng của tập dữ liệu đầu vào.
- (c) Độ nhiễu trong các tấm ảnh đầu vào.
- (d) Độ phức tạp trong câu hỏi từ ngôn ngữ tự nhiên.

## 39.4 Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
2. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
3. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).

4. **Rubric:**

| Mục   | Kiến Thức                                                                                                                                                                                                                                                    | Đánh Giá                                                                                                                                                                                                         |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| II.1. | <ul style="list-style-type: none"> <li>- Kiến thức về việc giải quyết bài toán VQA.</li> <li>- Cách kết hợp mô hình CNN và LSTM để giải quyết bài toán VQA.</li> </ul>                                                                                       | <ul style="list-style-type: none"> <li>- Nắm được các bước cơ bản giải quyết bài toán VQA.</li> <li>- Có thể cài đặt và huấn luyện mô hình giải quyết bài toán VQA sử dụng CNN và LSTM trong PyTorch.</li> </ul> |
| II.2. | <ul style="list-style-type: none"> <li>- Kiến thức về một số mô hình transformers.</li> <li>- Cách sử dụng mô hình transformers.</li> <li>- Kiến thức về việc sử dụng các mô hình thuộc họ transformers để giải quyết bài toán VQA.</li> </ul>               | <ul style="list-style-type: none"> <li>- Có thể cài đặt và huấn luyện mô hình giải quyết bài toán VQA sử dụng VisionTransformers và RoBERTa trong PyTorch.</li> </ul>                                            |
| II.3. | <ul style="list-style-type: none"> <li>- Kiến thức về mô hình ngôn ngữ thị giác lớn (LVLMs).</li> <li>- Cách sử dụng mô hình LVLMs bằng thư viện HuggingFace.</li> <li>- Kiến thức về việc sử dụng mô hình pretrained LVLMs để giải bài toán VQA.</li> </ul> | <ul style="list-style-type: none"> <li>- Có thể sử dụng mô hình thị giác ngôn ngữ lớn LLaVA để thực hiện bài toán VQA.</li> </ul>                                                                                |

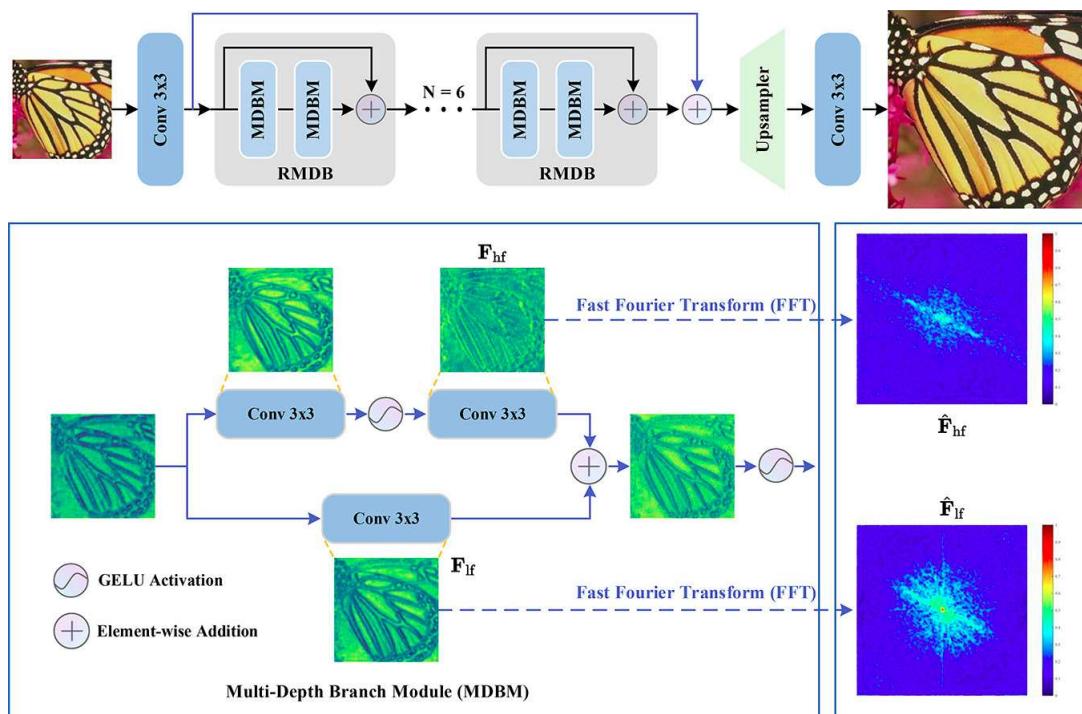
- *Hết* -

## **Phần IX**

**Module 9: Deep learning cho  
dữ liệu ảnh**

# Chương 40

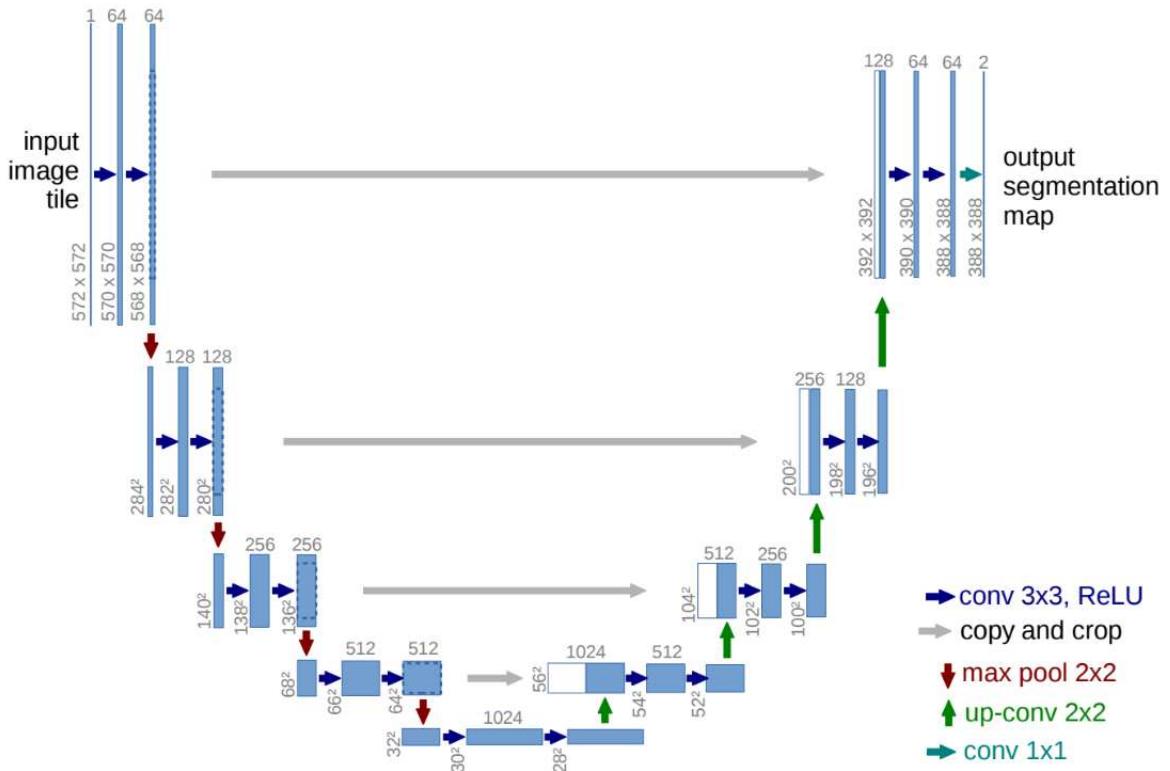
## Transfer learning và các bài toán ở dạng domain conversion (segmentation và super-resolution)



### 40.1 Lý Thuyết

#### 1. UNet:

UNET là một mô hình mạng nơ-ron sâu được phát triển bởi Olaf Ronneberger, Philipp Fischer và Thomas Brox vào năm 2015, chủ yếu được sử dụng trong lĩnh vực xử lý ảnh và trí tuệ nhân tạo (AI). Mô



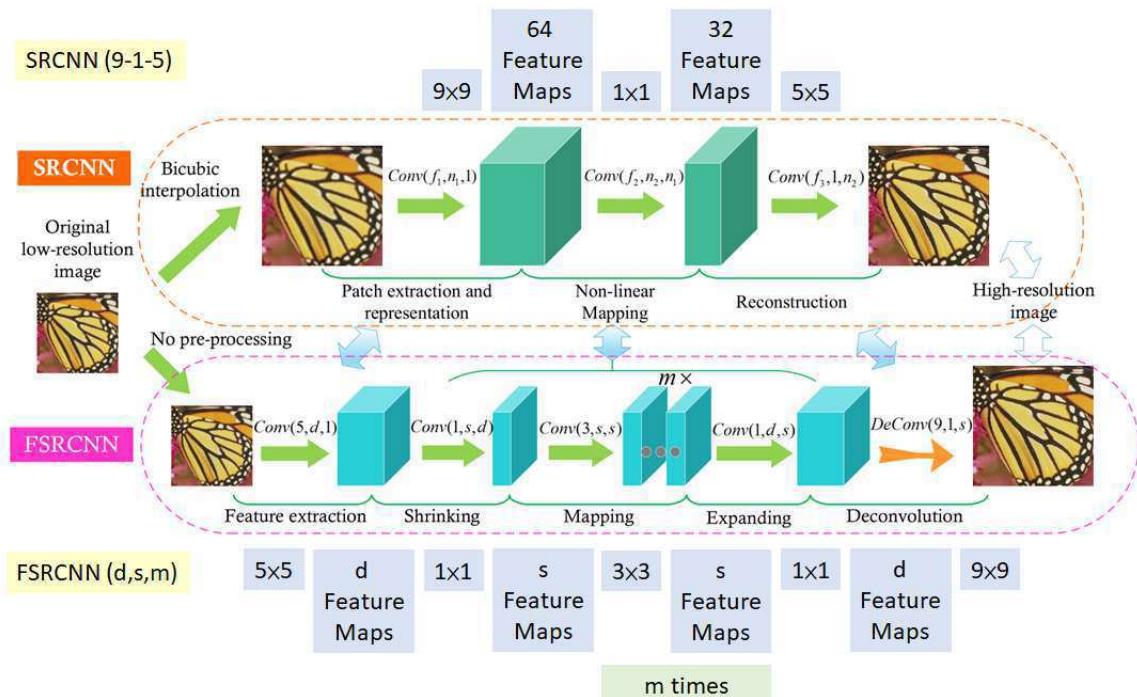
Hình 40.1: Image Super Resolution

hình này đã đạt được sự phổ biến rộng rãi trong các ứng dụng liên quan đến phân đoạn ảnh, như phân đoạn tế bào trong hình ảnh y học hoặc phân đoạn đối tượng trong hình ảnh chất lượng cao.

UNET sử dụng kiến trúc mạng Encoder-Decoder, với mục đích chính của lớp skip connection là tạo đường dẫn ngắn từ đầu vào đến đầu ra. Hàm kích hoạt thường được sử dụng trong UNET là Rectified Linear Unit (ReLU).

## 2. Super Resolution:

Super-resolution (SISR) là một kỹ thuật quan trọng trong xử lý hình ảnh, nhằm tăng độ phân giải của hình ảnh. Phương pháp này có thể cải thiện chất lượng và chi tiết của hình ảnh, thường được áp dụng trong việc nâng cấp hình ảnh cũ, tăng cường hiệu suất giám sát video,



Hình 40.2: Image Super Resolution

và trong nhiều lĩnh vực ứng dụng khác.

Trong SISR, mô hình máy học được huấn luyện với bộ dữ liệu chứa các cặp hình ảnh, với mỗi cặp bao gồm một hình ảnh độ phân giải cao (SR/HR image) và một phiên bản thu nhỏ có độ phân giải thấp (LR image). Mô hình sử dụng thông tin từ các cặp này để tái tạo chi tiết bị mất mát và biến đổi hình ảnh thấp độ phân giải thành hình ảnh độ phân giải cao hơn.

Các phương pháp nội suy (truyền thống) như Nearest-neighbor Interpolation, Bilinear Interpolation và Bicubic Interpolation thường được sử dụng trong SISR để tái tạo các giá trị pixel bị thiếu trong quá trình tăng độ phân giải.

Một điểm đáng lưu ý là single-image super-resolution (SISR) thường khó khăn hơn so với multi-image super-resolution (MISR)/video super-resolution (VSR). SISR chỉ có một hình ảnh đầu vào để làm việc, trong khi MISR/VSR có thêm thông tin tham khảo từ nhiều hình ảnh hoặc

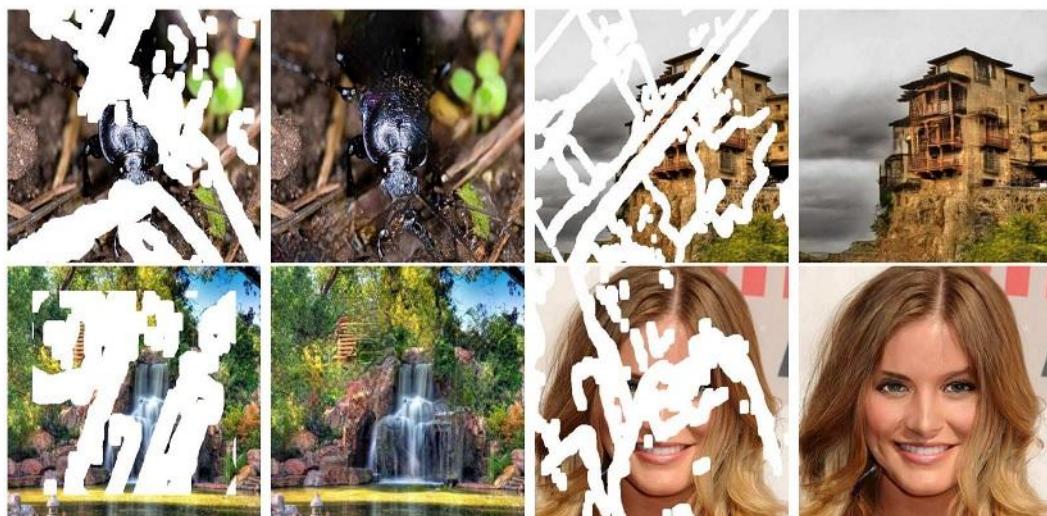
video, giúp cải thiện khả năng tái tạo hình ảnh độ phân giải cao. Trong SISR task, chúng ta cần khôi phục lại ảnh  $I_{SR}$  từ ảnh LR  $I_x$ .

$$I_{SR} = \mathcal{F}(I_x; \theta_{\mathcal{F}}) \quad (40.1)$$

$\mathcal{F}$ : thuật toán hoặc super resolution model

$\theta_{\mathcal{F}}$ : là parameter của  $\mathcal{F}$

### 3. Image Inpainting:



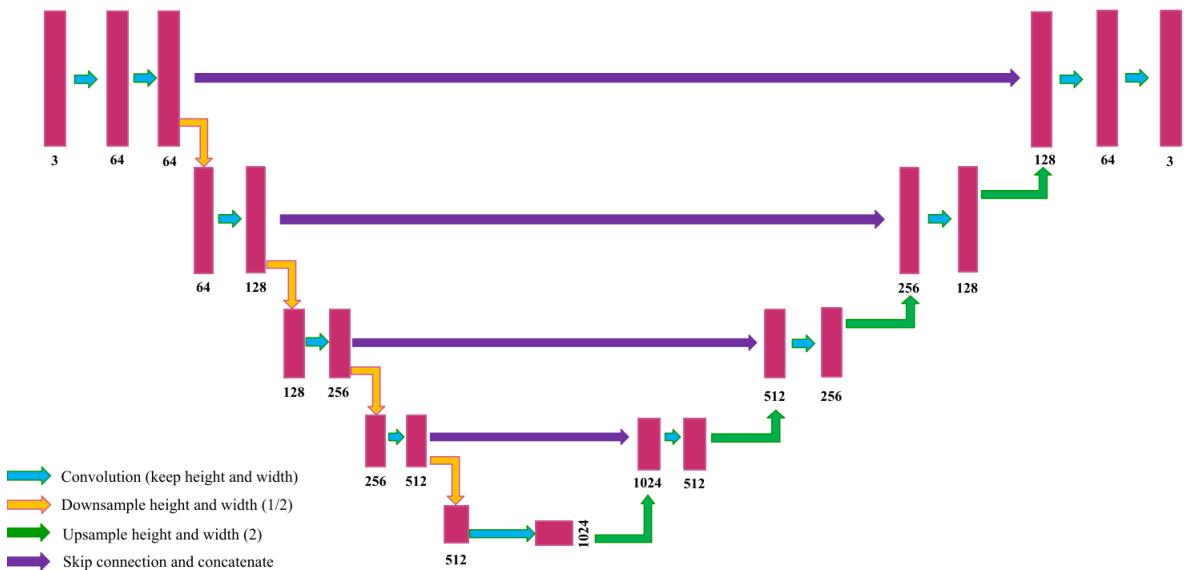
Hình 40.3: Image Inpainting

Inpainting là một phương pháp trong xử lý hình ảnh, được sử dụng để tái tạo hình ảnh bị thiếu một phần hoặc che khuất bằng cách dự đoán các pixel bị thiếu dựa trên thông tin có sẵn trong hình ảnh. Quá trình này có thể được thực hiện trong một hoặc hai giai đoạn. Trong inpainting, thông tin ngữ cảnh xung quanh các pixel bị thiếu là quan trọng để tái tạo hình ảnh một cách chính xác. Một số phương pháp inpainting sử dụng ví dụ hoặc các kỹ thuật khác để điền vào các pixel bị thiếu. Để biểu diễn một hình ảnh màu bị che khuất và mặt nạ (mask), ta thường tạo một hình ảnh kết hợp có bốn kênh, bao gồm hình ảnh gốc và mặt nạ (mask).

## 40.2 Bài Tập và Gợi Ý

Trong bài tập này các bạn sẽ xây dựng model cho riêng mình có kiến trúc giống Unet với mục đích là nhận Input là 1 ảnh màu (3 kênh) và generate ra 1 ảnh màu mới theo yêu cầu của từng task cụ thể (Bài tập 1). Sau đó các bạn chỉnh sửa model ở bài tập 1 để xây dựng model thực hiện task super resolution nhận ảnh input có kích thước  $64 \times 64$  và output ra ảnh có cùng nội dung nhưng kích thước tăng gấp 4 lần  $256 \times 256$  (Bài tập 2). Cuối cùng các bạn dùng model ở bài tập 1 để thực hiện task image inpainting cho ảnh có kích thước  $256 \times 256$ .

- Skip Connection Unet Architecture:** Xây dựng model có kiến trúc Unet có thông tin như hình 40.4 và code gợi ý bên dưới (các bạn có thể tham khảo thêm link file code ở trên) hoặc các bạn có thể xây dựng mode cho riêng mình theo yêu cầu là downsample sẽ thực hiện được 4 lần (giảm  $1/2, 1/4, 1/8, 1/16$  so với input image), sau đó sẽ upsample 4 lần để khôi phục lại size ảnh ban đầu ( $x2, x4, x8, x16$ ). Để kết hợp với skip connection các bạn nên dùng phép concatenate.



Hình 40.4: Kiến trúc Unet model

Bên dưới là đoạn code để các bạn tham khảo xây dựng network theo

kiến trúc Unet:

```
1 class FirstFeature(nn.Module):
2 def __init__(self, in_channels, out_channels):
3 super(FirstFeature, self).__init__()
4 self.conv = nn.Sequential(
5 nn.Conv2d(in_channels, out_channels, 1, 1, 0,
6 bias=False),
7 nn.LeakyReLU()
8)
9
10 def forward(self, x):
11 return self.conv(x)
12
13
14 class ConvBlock(nn.Module):
15 def __init__(self, in_channels, out_channels):
16 super(ConvBlock, self).__init__()
17 self.conv = nn.Sequential(
18 nn.Conv2d(in_channels, out_channels, 3, 1, 1,
19 bias=False),
20 nn.BatchNorm2d(out_channels),
21 nn.LeakyReLU(inplace=True),
22 nn.Conv2d(out_channels, out_channels, 3, 1,
23 1, bias=False),
24 nn.BatchNorm2d(out_channels),
25 nn.LeakyReLU(inplace=True),
26)
27
28
29 class Encoder(nn.Module):
30 def __init__(self, in_channels, out_channels) -> None
31 :
32 super().__init__()
33 self.encoder = nn.Sequential(
34 nn.MaxPool2d(2),
35 ConvBlock(in_channels, out_channels)
36)
37
38 def forward(self, x):
39 x = self.encoder(x)
40 return x
```

```
41
42 class Decoder(nn.Module):
43 def __init__(self, in_channels, out_channels):
44 super(Decoder, self).__init__()
45 self.conv = nn.Sequential(
46 nn.UpsamplingBilinear2d(scale_factor=2),
47 nn.Conv2d(in_channels, out_channels, 1, 1, 0,
48 bias=False),
49 nn.BatchNorm2d(out_channels),
50 nn.LeakyReLU(),
51)
52 self.conv_block = ConvBlock(in_channels,
53 out_channels)
54
55 def forward(self, x, skip):
56 x = self.conv(x)
57 x = torch.concat([x, skip], dim=1)
58 x = self.conv_block(x)
59 return x
60
61
62 class FinalOutput(nn.Module):
63 def __init__(self, in_channels, out_channels):
64 super(FinalOutput, self).__init__()
65 self.conv = nn.Sequential(
66 nn.Conv2d(in_channels, out_channels, 1, 1, 0,
67 bias=False),
68 nn.Tanh()
69)
70
71
72 class Unet(nn.Module):
73 def __init__(
74 self, n_channels=3, n_classes=3, features
75 =[64, 128, 256, 512],
76):
77 super(Unet, self).__init__()
78
79 self.n_channels = n_channels
80 self.n_classes = n_classes
81
82 self.in_conv1 = FirstFeature(n_channels, 64)
```

```

82 self.in_conv2 = ConvBlock(64, 64)
83
84 self.enc_1 = Encoder(64, 128)
85 self.enc_2 = Encoder(128, 256)
86 self.enc_3 = Encoder(256, 512)
87 self.enc_4 = Encoder(512, 1024)
88
89 self.dec_1 = Decoder(1024, 512)
90 self.dec_2 = Decoder(512, 256)
91 self.dec_3 = Decoder(256, 128)
92 self.dec_4 = Decoder(128, 64)
93
94 self.out_conv = FinalOutput(64, n_classes)
95
96
97 def forward(self, x):
98 x = self.in_conv1(x)
99 x1 = self.in_conv2(x)
100
101 x2 = self.enc_1(x1)
102 x3 = self.enc_2(x2)
103 x4 = self.enc_3(x3)
104 x5 = self.enc_4(x4)
105
106 x = self.dec_1(x5, x4)
107 x = self.dec_2(x, x3)
108 x = self.dec_3(x, x2)
109 x = self.dec_4(x, x1)
110
111 x = self.out_conv(x)
112 return x

```

### FirstFeature Class:

- **Mục đích:** Tạo feature map ban đầu từ input
- **Thành phần:** Một lớp convolution duy nhất theo sau là hàm LeakyReLU. convolution này sử dụng kích thước kernel là 1, bước nhảy là 1 và không padding. Đây là một lớp đơn giản được thiết kế để mở rộng số lượng channel cho feature map

### ConvBlock Class:

- **Mục đích:** Khối convolution cơ bản để trích xuất đặc trưng

- **Thành phần:** Hai nhóm Conv-BatchNorm-LeakyReLU liên tục. Khối này là một khối cơ bản trong U-Net, được sử dụng cho cả down-sampling và up-sampling

#### Encoder Class:

- **Mục đích:** Để giảm size của feature map và trích xuất các high-level feature.
- **Thành phần:** Một lớp Max Pooling theo sau là ConvBlock. Max Pooling giảm kích thước xuống một nửa, trong khi ConvBlock xử lý các đặc trưng.

#### Decoder Class:

- **Mục đích:** Để tăng kích thước feature map và kết hợp với feature map tương ứng từ Encoder (skip connection).
- **Thành phần:** Upsampling (sử dụng nội suy bilinear) để tăng kích thước không gian. Một lớp convolution để giảm số lượng channel. Một ConvBlock để xử lý các feature được ghép (từ lớp upsampling và skip connection).

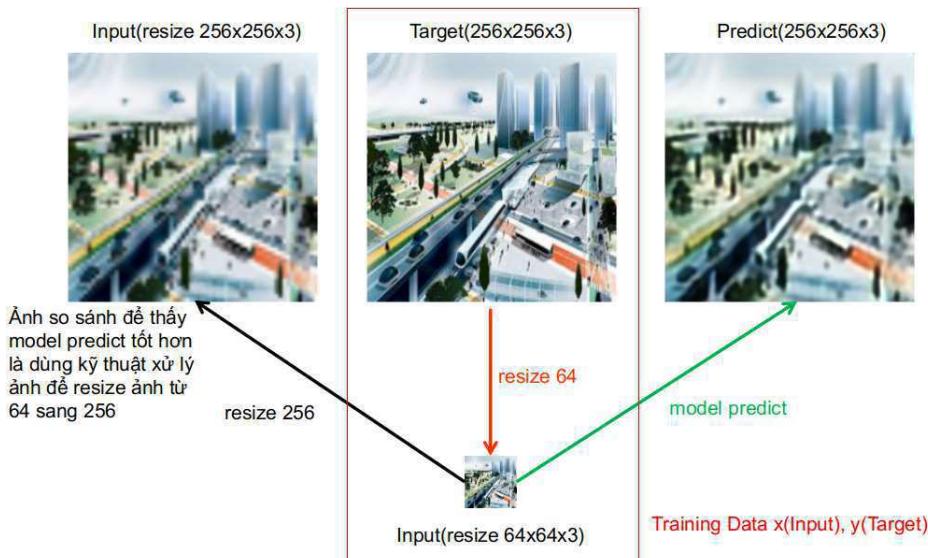
#### FinalOutput Class:

- **Mục đích:** Tạo ra đầu ra cuối cùng từ feature map cuối cùng.
- **Thành phần:** Một lớp convolution với hàm Tanh. Điều này giảm số lượng channel đầu ra xuống bằng số lượng channel của ảnh màu

#### Unet Class:

- **Mục đích:** Kết hợp tất cả các thành phần trên thành một kiến trúc U-Net đầy đủ.
- **Thành phần:** Xử lý ảnh đầu vào bằng FirstFeature và ConvBlock. Bốn lớp Encoder với số channel tăng dần, mỗi lớp tiếp tục downsample và xử lý feature map. Bốn lớp Decoder với số channel giảm dần, mỗi lớp tăng kích thước, kết hợp đặc trưng từ encoder (skip connection). Một lớp FinalOutput để tạo ra ảnh đã được xử lý.
- **Forward:** Đầu vào được xử lý qua các convolution ban đầu. Sau đó được downsample 4 lần, rồi được up sample lên 4 lần mỗi lần kết hợp với feature từ encoder. Cuối cùng đi qua lớp convolution cuối cùng để tạp ảnh đã xử lý

**2. Super Resolution with UNET:** Bài toán này giải quyết vấn đề khi một ảnh nhỏ resize lên một ảnh có kích thước lớn hơn sẽ khiến ảnh bị mờ, do đó các bạn sẽ xây dựng CNN model resize ảnh lớn hơn nhưng vẫn giữ được ảnh chất lượng tốt. Cho 1 tập data **Khoa\_LHR\_image.zip** đã được chia làm 2 phần train (685 ảnh), val (170 ảnh) và các ảnh có kích thước là 256x256x3. Các bạn hãy tự tạo cho mình tập dataset với ảnh input chính là các ảnh trong tập data nhưng kích thước ảnh đã giảm đi 4 lần (64x64x3). Sau đó dùng UNET model theo yêu cầu bên dưới để thực hiện bài toán, input là ảnh 64x64x3 và output là ảnh 256x256x3, target chính là ảnh gốc 256x256x3 của tập data



Yêu cầu model:

- Unet ở bài tập 1 (No skip connection: các bạn loại bỏ thành phần skip connection)
- Unet ở bài tập 1
- So sánh kết quả 2 models

Các bước thực hiện:

- Xây dựng dataset từ ảnh gốc (256x256x3). Khi load ảnh, mỗi sample các bạn tạo ra 2 ảnh (input và target). Input là ảnh gốc resize giảm đi 4 lần (64x64x3), target là ảnh gốc 256x256x3
- Chia data thành các tập train, validation (test tùy thuộc các bạn)
- Normalize data (dùng 1 trong các kỹ thuật chuẩn hóa đã được học). Kỹ thuật này phải phù hợp với activation của layer cuối cùng trong model để đảm bảo ảnh output từ model có giá trị trong range các giá trị của ảnh thông thường (có thể đã scale hoặc không).
- Điều chỉnh UNET model của bài tập 1 để nhận input là ảnh 64x64x3 output ra ảnh có kích thước 256x256x3
- Lựa chọn loss phù hợp cho việc output ra 1 ảnh từ model
- Config các hyperparameter
- Train và test kết quả

Đối với Super Resolution task chúng ta cần resize ảnh đầu vào từ 64 thành 256 (gấp 4 lần) trước khi đưa vào model. Do đó các bạn cần khai báo resize\_fnc (dòng 1) và sử dụng trong forward method của model (dòng 3) theo đoạn code tham khảo bên dưới

```

1 self.resize_fnc = transforms.Resize((LOW_IMG_HEIGHT*4,
2 LOW_IMG_HEIGHT*4), antialias=True)
3 x = self.resize_fnc(x)

```

Đối với việc không sử dụng skip connection trong model chúng ta cần chỉnh sửa lại phần Decoder bằng cách không dùng concat và gấp đôi out\_channel ở dòng 47, 48, 51

```

1 class FirstFeatureNoSkip(nn.Module):
2 def __init__(self, in_channels, out_channels):
3 super(FirstFeatureNoSkip, self).__init__()
4 self.conv = nn.Sequential(
5 nn.Conv2d(in_channels, out_channels, 1, 1, 0,
6 bias=False),
7 nn.LeakyReLU()
8)
9
10 def forward(self, x):
11 return self.conv(x)

```

```
11
12
13 class ConvBlockNoSkip(nn.Module):
14 def __init__(self, in_channels, out_channels):
15 super(ConvBlockNoSkip, self).__init__()
16 self.conv = nn.Sequential(
17 nn.Conv2d(in_channels, out_channels, 3, 1, 1,
18 bias=False),
19 nn.BatchNorm2d(out_channels),
20 nn.LeakyReLU(inplace=True),
21 nn.Conv2d(out_channels, out_channels, 3, 1,
22 1, bias=False),
23 nn.BatchNorm2d(out_channels),
24 nn.LeakyReLU(inplace=True),
25)
26
27
28
29 class EncoderNoSkip(nn.Module):
30 def __init__(self, in_channels, out_channels) -> None
31 :
32 super(EncoderNoSkip, self).__init__()
33 self.encoder = nn.Sequential(
34 nn.MaxPool2d(2),
35 ConvBlockNoSkip(in_channels, out_channels)
36)
37
38 def forward(self, x):
39 x = self.encoder(x)
40
41
42 class DecoderNoSkip(nn.Module):
43 def __init__(self, in_channels, out_channels):
44 super(DecoderNoSkip, self).__init__()
45 self.conv = nn.Sequential(
46 nn.UpsamplingBilinear2d(scale_factor=2),
47 nn.Conv2d(in_channels, out_channels*2, 1, 1,
48 0, bias=False),
49 nn.BatchNorm2d(out_channels*2),
50 nn.LeakyReLU(),
51)
52 self.conv_block = ConvBlockNoSkip(out_channels*2,
```

```
 out_channels)

52
53 def forward(self, x):
54 x = self.conv(x)
55 x = self.conv_block(x)
56 return x
57
58
59 class FinalOutputNoSkip(nn.Module):
60 def __init__(self, in_channels, out_channels):
61 super(FinalOutputNoSkip, self).__init__()
62 self.conv = nn.Sequential(
63 nn.Conv2d(in_channels, out_channels, 1, 1, 0,
64 bias=False),
65 nn.Tanh()
66)
67
68 def forward(self, x):
69 return self.conv(x)
70
71
72 class SR_Unet_NoSkip(nn.Module):
73 def __init__(
74 self, n_channels=3, n_classes=3
75):
76 super(SR_Unet_NoSkip, self).__init__()
77
78 self.n_channels = n_channels
79 self.n_classes = n_classes
80 self.resize_fnc = transforms.Resize(
81 LOW_IMG_HEIGHT*4, LOW_IMG_HEIGHT*4),
82 antialias=
83 True)
84 self.in_conv1 = FirstFeatureNoSkip(n_channels,
85 64)
86 self.in_conv2 = ConvBlockNoSkip(64, 64)
87
88 self.enc_1 = EncoderNoSkip(64, 128)
89 self.enc_2 = EncoderNoSkip(128, 256)
90 self.enc_3 = EncoderNoSkip(256, 512)
91 self.enc_4 = EncoderNoSkip(512, 1024)
92
93 self.dec_1 = DecoderNoSkip(1024, 512)
94 self.dec_2 = DecoderNoSkip(512, 256)
95 self.dec_3 = DecoderNoSkip(256, 128)
```

```

92 self.dec_4 = DecoderNoSkip(128, 64)
93
94 self.out_conv = FinalOutputNoSkip(64, n_classes)
95
96
97 def forward(self, x):
98 x = self.resize_fnc(x)
99 x = self.in_conv1(x)
100 x = self.in_conv2(x)
101
102 x = self.enc_1(x)
103 x = self.enc_2(x)
104 x = self.enc_3(x)
105 x = self.enc_4(x)
106
107 x = self.dec_1(x)
108 x = self.dec_2(x)
109 x = self.dec_3(x)
110 x = self.dec_4(x)
111
112 x = self.out_conv(x)
113 return x

```

Để có thể tạo được super resolution data chúng ta có thể tham khảo đoạn code bên dưới

```

1 class ImageDataset(Dataset):
2 def __init__(self, img_dir, is_train=True):
3 self.resize = transforms.Resize((LOW_IMG_WIDTH,
4 LOW_IMG_HEIGHT), antialias=True)
5 self.is_train = is_train
6 self.img_dir = img_dir
7 self.images = os.listdir(img_dir)
8
8 def __len__(self):
9 return len(self.images)
10
11 def normalize(self, input_image, target_image):
12 input_image = input_image*2 - 1
13 target_image = target_image*2 - 1
14
15 return input_image, target_image
16
17 def random_jitter(self, input_image, target_image):
18 if torch.rand([]) < 0.5:
19 input_image = transforms.functional.hflip(

```

```

20 input_image)
21 target_image = transforms.functional.hflip(
22 target_image)
23
24 def __getitem__(self, idx):
25 img_path = os.path.join(self.img_dir, self.images[
26 idx])
27 image = np.array(Image.open(img_path).convert("RGB"))
28 image = transforms.functional.to_tensor(image)
29
30 input_image = self.resize(image).type(torch.
31 float32)
32 target_image = image.type(torch.float32)
33
34 input_image, target_image = self.normalize(
35 input_image, target_image)
36
37 return input_image, target_image

```

### Khởi Tạo Class (`__init__` method):

- Hàm khởi tạo nhận `img_dir` và tham số tùy chọn `is_train`. `img_dir` là thư mục chứa ảnh, và `is_train` chỉ ra liệu tập dữ liệu có được sử dụng cho việc huấn luyện hay không (mặc định là True).
- `self.resize`: Thay đổi kích thước ảnh theo chiều rộng và chiều cao.
- `self.is_train`: Cho biết liệu tập dữ liệu có dùng cho huấn luyện không.
- `self.images`: Danh sách đường dẫn tệp cho tất cả ảnh trong `img_dir`.

### `__len__` method:

- Trả về số lượng ảnh trong tập dữ liệu, cần thiết để PyTorch hiểu kích thước của tập dữ liệu.

### Chuẩn Hóa (normalize method):

- Normalize ảnh đầu vào và target. Các ảnh được chuyển từ [0, 255] sang [-1, 1]. Đây là normalize phổ biến để ổn định quá trình huấn luyện.

#### **random\_jitter method:**

- Thực hiện tăng cường dữ liệu. Ngẫu nhiên áp dụng lật ngang cho cả ảnh đầu vào và target với xác suất 50%.

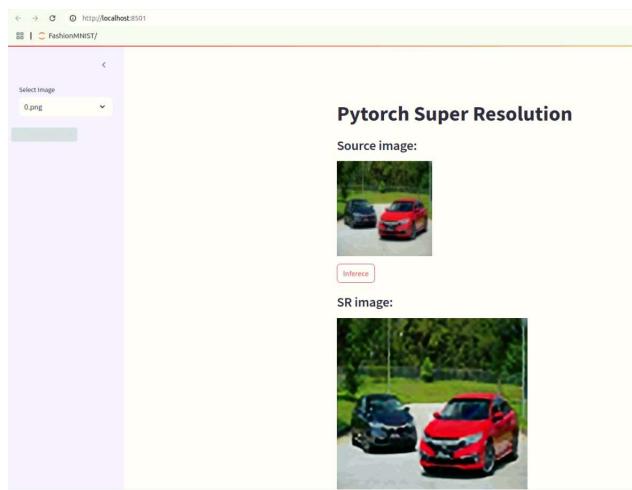
#### **\_\_getitem\_\_ method**

- `__getitem__` được sử dụng bởi PyTorch để lấy từng ảnh từ tập dữ liệu.
- `img_path`: Lấy đường dẫn của ảnh tại idx nhất định.
- `image`: Đọc ảnh theo chế độ RGB.
- `input_image` và `target_image`: Ảnh gốc được thay đổi kích thước để tạo `input_image`, và ảnh gốc cũng được giữ lại làm `target_image`. Cả hai được chuyển đổi sang kiểu dữ liệu `torch.float32`.
- Các ảnh sau đó được chuẩn hóa bằng phương thức `normalize`.
- Nếu tập dữ liệu được sử dụng cho huấn luyện (`self.is_train` là `True`), 'random\_jitter' được áp dụng cho cả `input_image` và `target_image`.

**Triển khai mô hình:** Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:

- [Github](#)
- Giao Diện

3. **Image Inpainting with UNET:** Bài toán này giải quyết vấn đề khôi phục ảnh input bị hư tổn bằng cách loại bỏ lỗi và khôi phục lại ảnh. Các bạn chọn 1 tập data bất kỳ, ví dụ chọn tập data **Khoa\_LHR\_image.zip** đã được chia làm 2 phần train (685 ảnh), val (170 ảnh) và các ảnh có kích thước là 256x256x3. Các bạn hãy tự tạo cho mình tập dataset với ảnh input chính là các ảnh trong tập data nhưng đã được vẽ random các line vào trong ảnh. Sau đó tạo một UNET model input là ảnh bị hư tổn 256x256x3 và output là ảnh 256x256x3 đã khôi phục được ảnh bị hư, target chính



Hình 40.5: Giao diện ứng dụng

là ảnh gốc 256x256x3 của tập data khi chưa được vẽ random các line

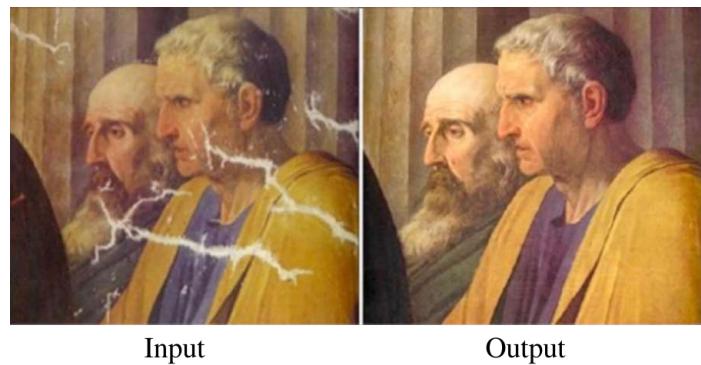
Yêu cầu model:

- Unet ở bài tập 1 (No skip connection: các bạn loại bỏ thành phần skip connection)
- Unet ở bài tập 1
- (Optional) Unet với pretrained weights
- So sánh kết quả 3 models

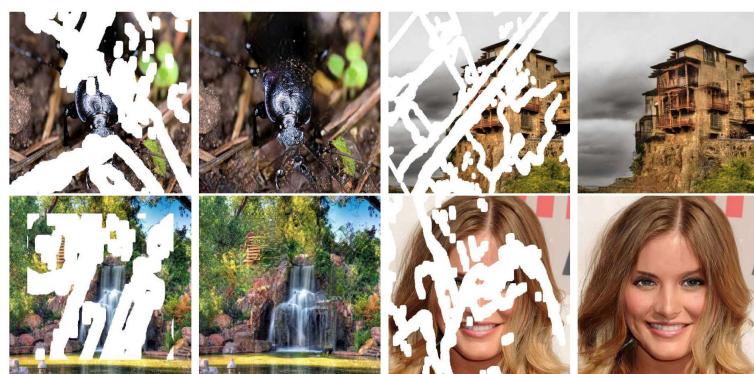
Các bước thực hiện:

- Xây dựng dataset từ ảnh gốc. Khi load ảnh mỗi sample các bạn tạo ra 2 ảnh (input và target). Input là ảnh gốc và được vẽ thêm các line random, target là ảnh gốc. (Cách vẽ line random các bạn có thể tùy chọn các cách mà các bạn tự học và tìm hiểu được)
- Chia data thành các tập train, validation (test tùy thuộc các bạn)
- Normalize data (dùng 1 trong các kỹ thuật chuẩn hóa đã được học). Kỹ thuật này phải phù hợp với activation của layer cuối cùng trong model để đảm bảo ảnh output từ model có giá trị trong range các giá trị của ảnh thông thường (có thể đã scale hoặc không).

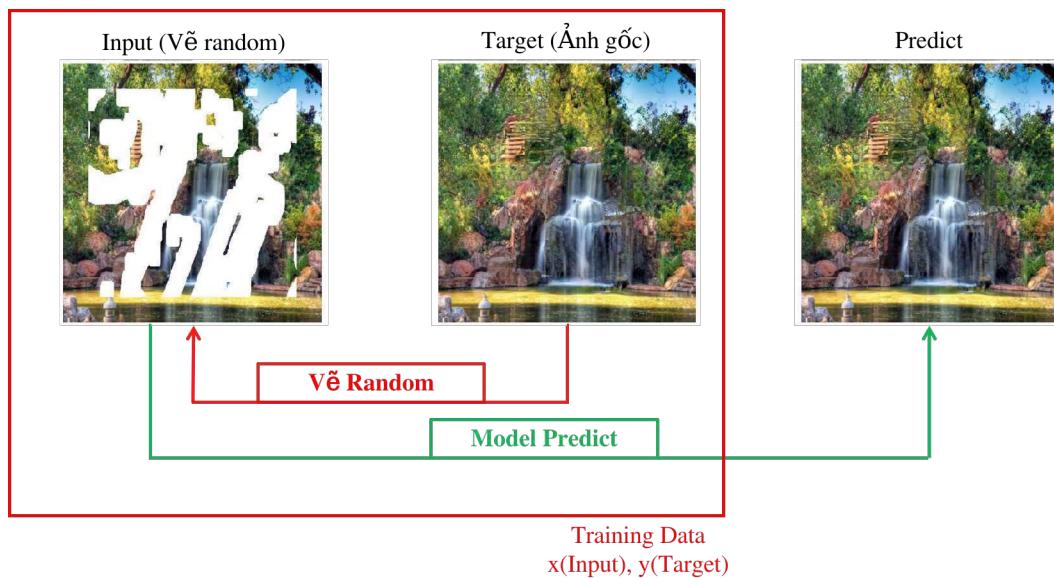
- Xây dựng UNET model nhận input là ảnh bị hư tổn, output ra ảnh có cùng kích thước với input nhưng đã được khôi phục
- Lựa chọn loss phù hợp cho việc output ra 1 ảnh từ model
- Config các hyperparameter, Train và test kết quả



Hình 40.6: Ứng dụng khôi phục ảnh cũ



Hình 40.7: Một số ví dụ khi tạo inpainting dataset



Các bạn có thể tham khảo đoạn code dưới đây để vẽ random lên ảnh

```

1 class ImageDataset(Dataset):
2 def __init__(self, img_dir, is_train=True):
3 self.is_train = is_train
4 self.img_dir = img_dir
5 self.images = os.listdir(img_dir)
6
7 def __len__(self):
8 return len(self.images)
9
10 def normalize(self, input_image, target_image):
11 input_image = input_image*2 - 1
12 target_image = target_image*2 - 1
13
14 return input_image, target_image
15
16 def random_jitter(self, input_image, target_image):
17 if torch.rand([]) < 0.5:
18 input_image = transforms.functional.hflip(
19 input_image)
20 target_image = transforms.functional.hflip(
21 target_image)
22
23 return input_image, target_image

```

```

23 def create_mask(self, image):
24 masked_image = image.copy()
25 ## Prepare masking matrix
26 mask = np.full((IMG_WIDTH,IMG_HEIGHT,3), 0, np.uint8)
27 for _ in range(np.random.randint(1, 5)):
28 # Get random x locations to start line
29 x1, x2 = np.random.randint(1, IMG_WIDTH), np.
random.randint(1, IMG_WIDTH)
30 # Get random y locations to start line
31 y1, y2 = np.random.randint(1, IMG_HEIGHT), np.
random.randint(1, IMG_HEIGHT)
32 # Get random thickness of the line drawn
33 thickness = np.random.randint(1, 15)
34 # Draw line on the black mask
35 cv2.line(mask,(x1,y1),(x2,y2),(1,1,1),thickness)
36
37 masked_image = np.where(mask, 255*np.ones_like(mask),
38 masked_image)
39 return masked_image
40
41 def __getitem__(self, idx):
42 img_path = os.path.join(self.img_dir, self.images[idx])
43
44 image = np.array(Image.open(img_path).convert("RGB"))
45
46 input_image = self.create_mask(image)
47 input_image = transforms.functional.to_tensor(
48 input_image)
49 target_image = transforms.functional.to_tensor(image)
50
51 input_image, target_image = self.normalize(
52 input_image, target_image)
53
54 if self.is_train:
55 input_image, target_image = self.random_jitter(
56 input_image, target_image)
57
58 return input_image, target_image

```

**Khởi Tạo Class (`__init__` method):**

- Hàm khởi tạo nhận `img_dir` và tham số tùy chọn `is_train`. `img_dir` là

thư mục chứa ảnh, và `is_train` chỉ ra liệu tập dữ liệu có được sử dụng cho việc huấn luyện hay không (mặc định là `True`).

- `self.is_train`: Cho biết liệu tập dữ liệu có dùng cho huấn luyện không.
- `self.img_files`: Danh sách đường dẫn tệp cho tất cả ảnh trong `images`.

#### **len method:**

- Trả về số lượng ảnh trong tập dữ liệu, cần thiết để PyTorch hiểu kích thước của tập dữ liệu.

#### **Chuẩn Hóa (normalize method):**

- Normalize ảnh đầu vào và target. Các ảnh được chuyển từ  $[0, 255]$  sang  $[-1, 1]$ . Đây là normalize phổ biến để ổn định quá trình huấn luyện.

#### **random\_jitter method:**

- Thực hiện tăng cường dữ liệu. Ngẫu nhiên áp dụng lật ngang cho cả ảnh đầu vào và target với xác suất 50%.

#### **create\_mask method:**

- Tạo và áp dụng một mask ngẫu nhiên lên hình ảnh.
- Vẽ các đường ngẫu nhiên trên hình ảnh, từ 1 đến 5 đường

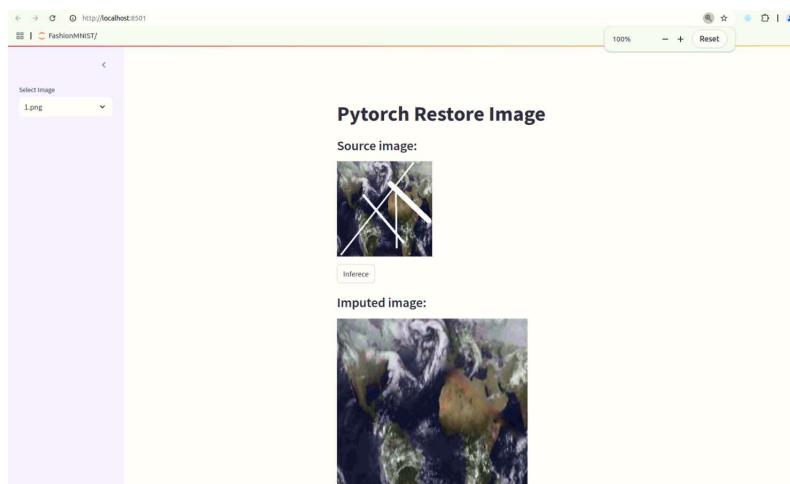
#### **getitem method**

- `__getitem__` được sử dụng bởi PyTorch để lấy từng ảnh từ tập dữ liệu.
- `img_path`: Lấy đường dẫn của ảnh tại `idx` nhất định.
- `image`: Đọc ảnh theo chế độ RGB.
- `input_image` và `target_image`: Ảnh gốc bị vẽ lên các đường ngẫu nhiên để tạo `input_image`, và ảnh gốc cũng được giữ lại làm `target_image`. Cả hai được chuyển đổi sang kiểu dữ liệu `torch.float32`.

- Các ảnh sau đó được chuẩn hóa bằng phương thức normalize.
- Nếu tập dữ liệu được sử dụng cho huấn luyện (self.is\_train là True), 'random\_jitter' được áp dụng cho cả input\_image và target\_image.

**Triển khai mô hình:** Dựa trên mô hình đã huấn luyện và thư viện streamlit để triển khai ứng dụng:

- [Github](#)
- Giao Diện



Hình 40.8: Giao diện ứng dụng

### 40.3 Trắc Nghiệm

1. Chức năng của các skip connection trong U-Net là gì?
  - (A). Tăng chi phí tính toán của mạng
  - (B). Kết hợp các feature map từ nhánh dowsampling với các feature map đã được upsample tương ứng trong nhánh mở rộng
  - (C). Đưa tính phi tuyến vào mạng
  - (D). Giảm số lượng tham số trong mạng
2. Ứng dụng chính của mô hình UNET là gì?
  - (A). Phân loại văn bản
  - (B). Phân đoạn hình ảnh
  - (C). Nhận dạng giọng nói
  - (D). Dự đoán giá cổ phiếu
3. Nếu không dùng skip connection trong U-Net có được không? Skip connection trong model U-Net có tác dụng gì?
  - (A) Được. Không có tác dụng gì
  - (B) Không. Giúp mô hình chạy được
  - (C) Được. Nó giúp ảnh thu được nhiều thông tin hơn
  - (D) Không. Nó giúp ảnh thu được không thay đổi gì
4. Điều gì làm cho U-Net khác biệt với Mạng Convolutional Neural Network (CNN) thông thường được sử dụng để phân loại hình ảnh?
  - (A). U-Net có nhánh upsampling kết hợp với các skip connection
  - (B). U-Net không sử dụng các lớp convolutional
  - (C). U-Net chỉ được sử dụng cho ảnh grayscale
  - (D). U-Net chỉ được sử dụng cho ảnh màu
5. Phương pháp nào trong các phương pháp sau đây thực hiện nội suy tuyến tính tuần tự trên hai trục của hình ảnh và có khả năng tạo ra kết quả tốt hơn so với Nearest-neighbor Interpolation?
  - (A). Bilinear Interpolation
  - (B). Bicubic Interpolation
  - (C). Nearest-neighbor Interpolation
  - (D). Trilinear Interpolation

6. Super-resolution dùng để làm gì?  
(A). Phân loại hình ảnh  
(C). Làm mờ hình ảnh  
(B). Tăng độ phân giải hình ảnh  
(D). Tạo hình ảnh 3D

7. Trong quá trình huấn luyện super resolution model, data thường được tạo thành các cặp ảnh như?  
(A). Hình ảnh màu và hình ảnh đen trắng  
(B). Cặp ảnh thuộc 2 class khác nhau  
(C). Hình ảnh độ phân giải cao và hình ảnh độ phân giải thấp  
(D). Tất cả đều đúng

8. Vấn đề phổ biến nào có thể xảy ra trong các mạng deep learning mà kiến trúc của U-Net giúp giảm thiểu? ?  
(A). Overfitting  
(B). Underfitting  
(C). Mất thông tin không gian trong quá trình downsampling  
(D). Độ phức tạp tính toán

9. Mục tiêu chính của image inpainting là gì??  
(A). Nén ảnh mà không làm giảm đáng kể chất lượng  
(B) Tăng cường độ tương phản và màu sắc của ảnh  
(C) Phát hiện và xóa các đối tượng khỏi ảnh  
(D) Điền vào các vùng bị thiếu hoặc bị hỏng của ảnh bằng nội dung phù hợp

10. Ưu điểm của việc sử dụng phương pháp deep learning cho inpainting so với các phương pháp truyền thống là gì? ?  
(A). Các phương pháp deep learning luôn nhanh hơn.  
(B). Các phương pháp deep learning không yêu cầu bất kỳ dữ liệu huấn luyện nào  
(C). Các phương pháp deep learning xử lý tốt hơn các vùng khuyết thiếu lớn và texture phức tạp.  
(D). Các phương pháp deep learning đơn giản hơn để triển khai

## 40.4 Phụ Lục

1. **Hint:** Các bạn được khuyến khích nên tham khảo code hướng dẫn (link bên dưới) trước. Sau đó tự xây dựng pipeline của riêng mình. Link data đã bao gồm trong file code. Các bạn lưu ý điều chỉnh batch size phù hợp với phần cứng mà mình có :
  - [Link Bài tập 1](#)
  - [Link Bài tập 2](#)
  - [Link Bài tập 3](#)
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể tải tại [Link](#) (**Lưu ý** Sáng thứ 3 khi hết deadline phần bài tập ad mới copy các nội dung bài giải nêu trên vào đường dẫn)
3. **Rubric:**

| Câu | Kiến Thức                                                                                                                                                                                                                                                                                                                                     | Đánh Giá                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1   | <ul style="list-style-type: none"> <li>- Ôn tập về kiến trúc Unet</li> <li>- Ôn tập về các thành phần strong kiến trúc Unet, encoder, decoder, upsampling technique</li> <li>- Các ứng dụng của kiến trúc Unet</li> </ul>                                                                                                                     | <ul style="list-style-type: none"> <li>- Hiểu về khái niệm vanishing gradient</li> <li>- Có thể tự code và custom các thành phần trong kiến trúc Unet như encoder, decoder, upsampling</li> <li>- Có thể kết hợp và custom các thành phần để hình thành nên kiến trúc Unet theo từng task với yêu cầu cụ thể.</li> </ul>                                                                                   |
| 2   | <ul style="list-style-type: none"> <li>- Khái niệm về Image super resolution</li> <li>- Tự tạo data dùng cho Image super resolution</li> <li>- Ôn tập và hiểu thêm tầm quan trọng của skip connection trong xây dựng model cho Image super resolution</li> <li>- Cách xây dựng và lựa chọn model trong Image super resolution task</li> </ul> | <ul style="list-style-type: none"> <li>- Xây dựng được model (dựa trên kiến trúc Unet) cơ bản cho task Image super resolution</li> <li>- Biết cách code và linh hoạt tinh chỉnh skip connection trong Unet</li> <li>- Biết cách can thiệp vào model để chỉnh sửa theo yêu cầu của Image super resolution task dựa vào kiến trúc Unet</li> </ul>                                                            |
| 3   | <ul style="list-style-type: none"> <li>- Khái niệm về Image inpainting</li> <li>- Tự tạo data dùng cho Image inpainting</li> <li>- Ôn tập và hiểu thêm tầm quan trọng của skip connection trong xây dựng model cho Image inpainting</li> <li>- Cách xây dựng và lựa chọn model trong Image inpainting task</li> </ul>                         | <ul style="list-style-type: none"> <li>- Xây dựng được model (dựa trên kiến trúc Unet) cơ bản cho task Image inpainting</li> <li>- Biết cách code và linh hoạt tinh chỉnh skip connection trong Unet</li> <li>- Biết cách can thiệp vào model để chỉnh sửa theo yêu cầu của Image inpainting task dựa vào kiến trúc Unet</li> <li>- Biết cách áp dụng chiến thuật pretrain cho Image inpainting</li> </ul> |

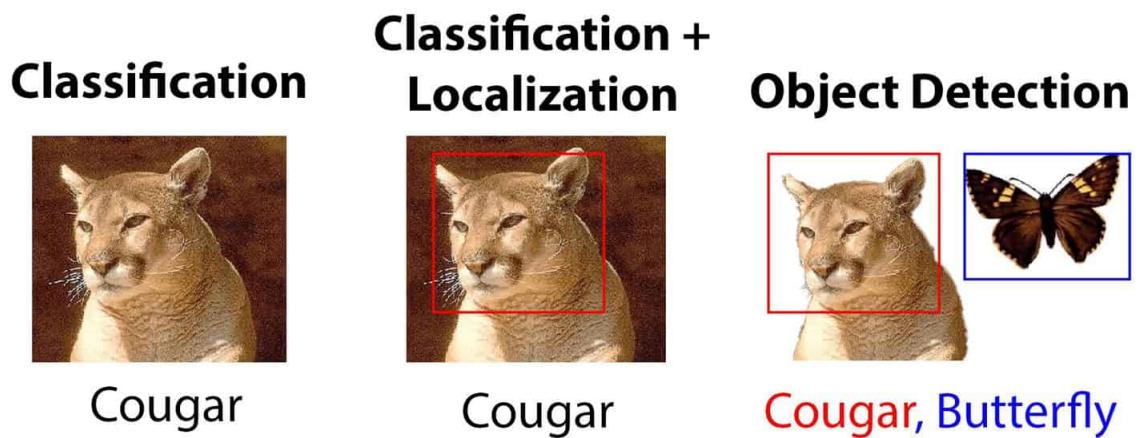
# Chương 41

## Object Detection 1: Thảo luận YOLO v1 đến v3

### 41.1 Giới thiệu

Trong buổi hôm nay, chúng ta sẽ cùng nhau xây dựng model YOLO cho bài toán Object Detection. Tuy nhiên, thay vì đi thẳng vào YOLO, ta sẽ xây dựng từ bài toán Classification, sau đó thêm vào task Localization và cuối cùng là Object Detection (YOLO). Chúng ta sẽ đi qua 4 phần chính:

1. Classification (1 object trong ảnh)
2. Classification (1 object trong ảnh) + Localization (bounding box)
3. Classification (nhiều hơn 1 object trong ảnh) + Localization (bounding box)
4. YOLOv1



Hình 41.1: Hình ảnh minh họa cho 3 bài toán Classification, Localization và Object Detection.

Trong hình ảnh minh họa ở Figure 60.1, chúng ta có thể thấy sự khác biệt giữa ba bài toán chính: **Classification** chỉ đơn giản là phân loại một object trong ảnh, **Localization** không chỉ phân loại mà còn xác định vị trí của object đó bằng bounding box, và cuối cùng là **Object Detection** kết hợp cả hai yếu tố này để phát hiện và định vị nhiều object trong một hình ảnh.

## 41.2 Nội dung chính

### 41.2.1 Bài toán 1: Classification (Phân loại một object trong hình ảnh)

#### Giới thiệu

Đầu tiên, chúng ta sẽ xây dựng một mô hình **Classification** đơn giản để làm nền tảng cho các bài toán phức tạp hơn sau này. Mục tiêu của bài toán này là phân loại hình ảnh chứa một object duy nhất thuộc về một trong hai class: **cat** (mèo) hoặc **dog** (chó).

#### Import và Tải Dữ liệu

Để bắt đầu, chúng ta cần import các thư viện cần thiết và tải xuống bộ dữ liệu. Bộ dữ liệu này bao gồm các hình ảnh của mèo và chó, được sử dụng rộng rãi trong các bài toán **Image Classification** và **Object Detection**. Việc sử dụng bộ dữ liệu từ **kagglehub** giúp chúng ta dễ dàng quản lý và truy cập dữ liệu một cách hiệu quả.

```
1 import kagglehub
2
3 # Download the latest dataset version
4 data_dir = kagglehub.dataset_download("andrewmvd/dog-and-
 cat-detection")
5 print("Path to dataset files:", data_dir)
6
```

#### Import Các Thư Viện Cần Thiết

Tiếp theo, chúng ta sẽ import các thư viện cần thiết để xử lý dữ liệu và xây dựng mô hình.

```
1 import os
2 import torch
3 import numpy as np
```

```
4 import pandas as pd
5 import seaborn as sns
6 import torch.nn as nn
7 import torch.optim as optim
8 import matplotlib.pyplot as plt
9 import xml.etree.ElementTree as ET
10
11 from PIL import Image
12 from torchvision import transforms, models
13 from torch.utils.data import Dataset, DataLoader
14 from sklearn.metrics import confusion_matrix
15 from sklearn.model_selection import train_test_split
16 from torchvision.models.resnet import ResNet18_Weights
17
```

## Định Nghĩa Class Dataset

Để quản lý và xử lý dữ liệu hiệu quả, chúng ta sẽ định nghĩa một class `ImageDataset`. Class này sẽ chịu trách nhiệm tải hình ảnh, xử lý các label tương ứng và thực hiện các bước tiền xử lý dữ liệu cần thiết trước khi đưa vào model.

```
1 # Dataset Class
2 class ImageDataset(Dataset):
3 def __init__(self, annotations_dir, image_dir,
4 transform=None):
5 self.annotations_dir = annotations_dir
6 self.image_dir = image_dir
7 self.transform = transform
8 self.image_files = self.
9 filter_images_with_multiple_objects()
10
11 def filter_images_with_multiple_objects(self):
12 valid_image_files = []
13 for f in os.listdir(self.image_dir):
14 if os.path.isfile(os.path.join(self.image_dir
15 , f)):
16 img_name = f
17 annotation_name = os.path.splitext(
18 img_name)[0] + ".xml"
```

```
15 annotation_path = os.path.join(self.
16 annotations_dir, annotation_name)
17
18 # Keep images that have single object
19 if self.count_objects_in_annotation(
20 annotation_path) <= 1:
21 valid_image_files.append(img_name)
22 else:
23 print(
24 f"Image {img_name} has multiple
25 objects and will be excluded from the dataset"
26)
27 return valid_image_files
28
29 def count_objects_in_annotation(self, annotation_path):
30 try:
31 tree = ET.parse(annotation_path)
32 root = tree.getroot()
33 count = 0
34 for obj in root.findall("object"):
35 count += 1
36 return count
37 except FileNotFoundError:
38 return 0
39
40 def __len__(self):
41 return len(self.image_files)
42
43 def __getitem__(self, idx):
44 # Image path
45 img_name = self.image_files[idx]
46 img_path = os.path.join(self.image_dir, img_name)
47
48 # Load image
49 image = Image.open(img_path).convert("RGB")
50
51 # Annotation path
52 annotation_name = os.path.splitext(img_name)[0] +
53 ".xml"
54 annotation_path = os.path.join(self.
55 annotations_dir, annotation_name)
56
57 # Parse annotation
```

```

53 label = self.parse_annotation(annotation_path)
54
55 if self.transform:
56 image = self.transform(image)
57
58 return image, label
59
60 def parse_annotation(self, annotation_path):
61 tree = ET.parse(annotation_path)
62 root = tree.getroot()
63
64 label = None
65 for obj in root.findall("object"):
66 name = obj.find("name").text
67 if (
68 label is None
69): # Take the first label for now. We are
70 working with 1 label per image
71 label = name
72
73 # Convert label to numerical representation (0
74 for cat, 1 for dog)
75 label_num = 0 if label == "cat" else 1 if label
76 == "dog" else -1
77
78 return label_num
79
80

```

Trong class `ImageDataset`, chúng ta thực hiện các bước sau:

- **Khởi tạo:** Xác định thư mục chứa annotations và hình ảnh, cũng như các biến đổi dữ liệu nếu có.
- **Lọc dữ liệu:** Loại bỏ những hình ảnh chứa nhiều hơn một object để đảm bảo tính nhất quán của bài toán **Classification**.
- **Đếm object:** Đếm số lượng object trong mỗi hình ảnh dựa trên các annotations.
- **Lấy mẫu:** Định nghĩa method `__getitem__` để truy xuất hình ảnh và nhãn tương ứng.

- **Phân tích annotations:** Chuyển đổi nhãn văn bản thành số để dễ dàng xử lý trong mô hình.

## Phân Tích và Chuẩn Bị Dữ Liệu

Sau khi định nghĩa class dataset, chúng ta tiến hành phân tích và chuẩn bị dữ liệu.

```

1 # Data directory
2 annotations_dir = os.path.join(data_dir, 'annotations')
3 image_dir = os.path.join(data_dir, 'images')
4
5 # Get list of image files and create a dummy dataframe to
 split the data
6 image_files = [f for f in os.listdir(image_dir) if os.
 path.isfile(os.path.join(image_dir, f))]
7 df = pd.DataFrame({'image_name': image_files})
8
9 # Split data
10 train_df, val_df = train_test_split(df, test_size=0.2,
 random_state=42)
11

```

## Chuẩn Bị và Split Dữ Liệu

Chúng ta sẽ áp dụng các biến đổi (`transforms`) cho dữ liệu hình ảnh và chia tách dữ liệu thành tập training (train set) và tập testing (test set).

```

1 # Transforms
2 transform = transforms.Compose([
3 transforms.Resize((224, 224)),
4 transforms.ToTensor(),
5 transforms.Normalize(mean=[0.485, 0.456, 0.406], std
 =[0.229, 0.224, 0.225])
6])
7
8 # Datasets
9 train_dataset = ImageDataset(annotations_dir, image_dir,
 transform=transform)

```

```
10 val_dataset = ImageDataset(annotations_dir, image_dir,
 transform=transform)
11
12 # Filter datasets based on train_df and val_df
13 train_dataset.image_files = [f for f in train_dataset.
 image_files if f in train_df['image_name'].values]
14 val_dataset.image_files = [f for f in val_dataset.
 image_files if f in val_df['image_name'].values]
15
16 # Dataloaders
17 train_loader = DataLoader(train_dataset, batch_size=32,
 shuffle=True)
18 val_loader = DataLoader(val_dataset, batch_size=32,
 shuffle=False)
19
```

## Xây Dựng Model

Chúng ta sẽ sử dụng model ResNet18 và finetune để phù hợp với bài toán classification của chúng ta.

```
1 # Model
2 model = models.resnet18(weights=ResNet18_Weights.DEFAULT)
3 num_ftrs = model.fc.in_features
4 model.fc = nn.Linear(num_ftrs, 2) # 2 classes: cat and
 dog
5
6 # Device
7 device = torch.device("cuda" if torch.cuda.is_available()
 else "cpu")
8 model.to(device)
9
10 # Loss and Optimizer
11 criterion = nn.CrossEntropyLoss()
12 optimizer = optim.Adam(model.parameters(), lr=0.001)
13
14 # Show model summary
15 print(model)
16
```

## Training loop

Cuối cùng, chúng ta sẽ training model trên train set và evaluate trên test set.

```
1 # Training Loop
2 num_epochs = 10
3 for epoch in range(num_epochs):
4 model.train()
5 for batch_idx, (data, targets) in enumerate(
6 train_loader):
6 data = data.to(device)
7 targets = targets.to(device)
8
9 scores = model(data)
10 loss = criterion(scores, targets)
11
12 optimizer.zero_grad()
13 loss.backward()
14 optimizer.step()
15
16 # Validation
17 model.eval()
18 with torch.no_grad():
19 correct = 0
20 total = 0
21 for data, targets in val_loader:
22 data = data.to(device)
23 targets = targets.to(device)
24 scores = model(data)
25 _, predictions = scores.max(1)
26 correct += (predictions == targets).sum()
27 total += targets.size(0)
28
29 print(f'Epoch {epoch+1}/{num_epochs}, Validation
30 Accuracy: {float(correct)/float(total)*100:.2f}%')
```

Sau khi huấn luyện xong, kết quả đạt được của mô hình như sau:



Hình 41.2: Kết quả của mô hình sau khi huấn luyện 10 epoch.

### 41.2.2 Bài toán 2: Classification + Bounding Box Regression

Trong bài toán thứ hai, chúng ta không chỉ thực hiện **classification** object trong hình ảnh mà còn định vị vị trí của object đó bằng cách dự đoán **bounding box**.

#### Import và Tải Dữ liệu

Giống như bài toán trước, chúng ta bắt đầu bằng việc import các thư viện cần thiết và tải dữ liệu.

```
1 import kagglehub
2 # Download latest version
3 data_dir = kagglehub.dataset_download("andrewmvd/dog-and-
 cat-detection")
4 print("Path to dataset files:", data_dir)
5
```

## Import Các Thư Viện Cần Thiết

```
1 import os
2 import torch
3 import numpy as np
4 import pandas as pd
5 import seaborn as sns
6 import torch.nn as nn
7 import torch.optim as optim
8 import matplotlib.pyplot as plt
9 import matplotlib.patches as patches
10 import xml.etree.ElementTree as ET
11
12 from PIL import Image
13 from torchvision import transforms, models
14 from torch.utils.data import Dataset, DataLoader
15 from sklearn.metrics import confusion_matrix
16 from sklearn.model_selection import train_test_split
17 from torchvision.models.resnet import ResNet18_Weights
18
```

## Định Nghĩa Class Dataset với Bounding Box

Chúng ta sẽ mở rộng lớp `ImageDataset` để bao gồm cả thông tin về **bounding box**.

```
1 # Dataset Class
2 class ImageDataset(Dataset):
3 def __init__(self, annotations_dir, image_dir,
4 transform=None):
5 self.annotations_dir = annotations_dir
6 self.image_dir = image_dir
7 self.transform = transform
8 self.image_files = self.
9 filter_images_with_multiple_objects()
10
11 def filter_images_with_multiple_objects(self):
12 valid_image_files = []
13 for f in os.listdir(self.image_dir):
```

```
12 if os.path.isfile(os.path.join(self.image_dir
13 , f)):
14 img_name = f
15 annotation_name = os.path.splitext(
16 img_name)[0] + ".xml"
17 annotation_path = os.path.join(self.
18 annotations_dir, annotation_name)
19
20 if self.count_objects_in_annotation(
21 annotation_path) == 1:
22 valid_image_files.append(img_name)
23 return valid_image_files
24
25 def count_objects_in_annotation(self, annotation_path):
26 try:
27 tree = ET.parse(annotation_path)
28 root = tree.getroot()
29 count = 0
30 for obj in root.findall('object'):
31 count += 1
32 return count
33 except FileNotFoundError:
34 return 0
35
36 def __len__(self):
37 return len(self.image_files)
38
39 def __getitem__(self, idx):
40 # Image path
41 img_name = self.image_files[idx]
42 img_path = os.path.join(self.image_dir, img_name)
43
44 # Load image
45 image = Image.open(img_path).convert("RGB")
46
47 # Annotation path
48 annotation_name = os.path.splitext(img_name)[0] +
49 ".xml"
50 annotation_path = os.path.join(self.
51 annotations_dir, annotation_name)
52
53 # Parse annotation
```

```
48 label, bbox = self.parse_annotation(
49 annotation_path) # Get both label and bbox
50
51 if self.transform:
52 image = self.transform(image)
53
54 return image, label, bbox
55
56 def parse_annotation(self, annotation_path):
57 tree = ET.parse(annotation_path)
58 root = tree.getroot()
59
60 # Get image size for normalization
61 image_width = int(root.find('size/width').text)
62 image_height = int(root.find('size/height').text)
63
64 label = None
65 bbox = None
66 for obj in root.findall('object'):
67 name = obj.find('name').text
68 if label is None: # Take the first label
69 label = name
70 # Get bounding box coordinates
71 xmin = int(obj.find('bndbox/xmin').text)
72 ymin = int(obj.find('bndbox/ymin').text)
73 xmax = int(obj.find('bndbox/xmax').text)
74 ymax = int(obj.find('bndbox/ymax').text)
75
76 # Normalize bbox coordinates to [0, 1]
77 bbox = [
78 xmin / image_width,
79 ymin / image_height,
80 xmax / image_width,
81 ymax / image_height,
82]
83
84 # Convert label to numerical representation (0 for
85 # cat, 1 for dog)
86 label_num = 0 if label == 'cat' else 1 if label ==
87 'dog' else -1
88
89 return label_num, torch.tensor(bbox, dtype=torch.
float32)
90
91
```

## Phân Tích và Chuẩn Bị Dữ Liệu

Tương tự như bài toán trước, chúng ta sẽ chuẩn bị dữ liệu cho bài toán này.

```
1 # Data directory
2 annotations_dir = os.path.join(data_dir, 'annotations')
3 image_dir = os.path.join(data_dir, 'images')
4
5 # Get list of image files and create a dummy dataframe to
6 # split the data
7 image_files = [f for f in os.listdir(image_dir) if os.
8 path.isfile(os.path.join(image_dir, f))]
9 df = pd.DataFrame({'image_name': image_files})
10
11 # Split data
12 train_df, val_df = train_test_split(df, test_size=0.2,
13 random_state=42)
14
```

## Chuẩn Bị và Split Dữ Liệu

Chúng ta áp dụng các biến đổi (`transforms`) cho dữ liệu và chia tách dữ liệu thành các `train set` và `test set`.

```
1 # Transforms
2 transform = transforms.Compose([
3 transforms.Resize((224, 224)),
4 transforms.ToTensor(),
5 transforms.Normalize(mean=[0.485, 0.456, 0.406], std
6 =[0.229, 0.224, 0.225])
7])
8
9 # Datasets
10 train_dataset = ImageDataset(annotations_dir, image_dir,
11 transform=transform)
12 val_dataset = ImageDataset(annotations_dir, image_dir,
13 transform=transform)
14
15 # Filter datasets based on train_df and val_df
```

```
13 train_dataset.image_files = [f for f in train_dataset.
 image_files if f in train_df['image_name'].values]
14 val_dataset.image_files = [f for f in val_dataset.
 image_files if f in val_df['image_name'].values]
15
16 # Dataloaders
17 train_loader = DataLoader(train_dataset, batch_size=32,
 shuffle=True)
18 val_loader = DataLoader(val_dataset, batch_size=32,
 shuffle=False)
19
```

## Xây Dựng Model với 2 head

Chúng ta sẽ xây dựng một mô hình có hai head: một để **classification** và một để dự đoán **bounding box**.

```
1 # Model with Two Heads
2 class TwoHeadedModel(nn.Module):
3 def __init__(self, num_classes=2):
4 super(TwoHeadedModel, self).__init__()
5 self.base_model = models.resnet18(weights=
6 ResNet18_Weights.DEFAULT)
7 self.num_ftrs = self.base_model.fc.in_features
8
9 # Remove the original fully connected layer
10 self.base_model.fc = nn.Identity()
11
12 # Classification head
13 self.classifier = nn.Linear(self.num_ftrs,
14 num_classes)
15
16 # Bounding box regression head
17 self.regressor = nn.Linear(self.num_ftrs, 4)
18
19 def forward(self, x):
20 x = self.base_model(x)
21 class_logits = self.classifier(x)
22 bbox_coords = torch.sigmoid(self.regressor(x))
23 return class_logits, bbox_coords
```

Trong mô hình này, chúng ta thực hiện các bước sau:

- **Base model:** Sử dụng pre-trained ResNet18
- **Classification head:** Thêm một lớp fully connected mới để thực hiện nhiệm vụ phân loại.
- **Regression head:** Thêm một lớp fully connected khác để dự đoán tọa độ bounding box, với mỗi tọa độ được chuẩn hóa về khoảng [0, 1] thông qua hàm sigmoid.

## Cài Đặt Model

Chúng ta sẽ khởi tạo mô hình, thiết lập thiết bị, loss function và optimizer.

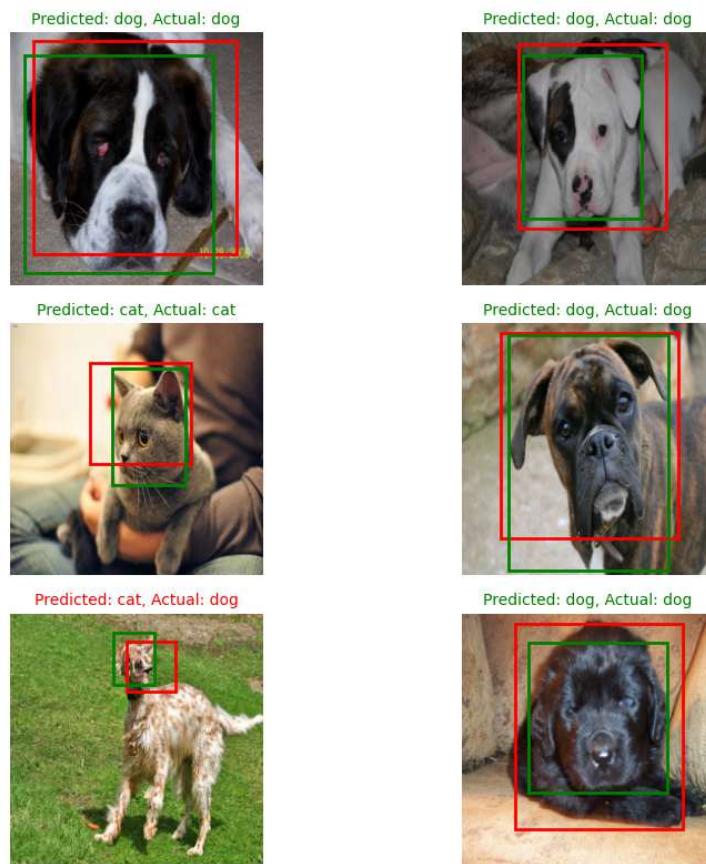
```
1 # Model
2 model = TwoHeadedModel()
3
4 # Device
5 device = torch.device("cuda" if torch.cuda.is_available()
6 else "cpu")
7 model.to(device)
8
9 # Loss and Optimizer
10 criterion_class = nn.CrossEntropyLoss()
11 criterion_bbox = nn.MSELoss()
12 optimizer = optim.Adam(model.parameters(), lr=0.001)
13
```

## Training loop

Chúng ta sẽ huấn luyện mô hình với cả hai head và theo dõi hiệu suất trên test set.

```
1 # Training Loop
2 num_epochs = 10
3 for epoch in range(num_epochs):
4 model.train()
```

```
5 for batch_idx, (data, targets, bboxes) in enumerate(
6 train_loader):
7 data = data.to(device)
8 targets = targets.to(device)
9 bboxes = bboxes.to(device)
10
11 scores, pred_bboxes = model(data)
12 loss_class = criterion_class(scores, targets)
13 loss_bbox = criterion_bbox(pred_bboxes, bboxes)
14 loss = loss_class + loss_bbox # Combine losses
15
16 optimizer.zero_grad()
17 loss.backward()
18 optimizer.step()
19
20 # Validation
21 model.eval()
22 with torch.no_grad():
23 correct = 0
24 total = 0
25 total_loss_bbox = 0
26 total_samples = 0
27 for data, targets, bboxes in val_loader:
28 data = data.to(device)
29 targets = targets.to(device)
30 bboxes = bboxes.to(device)
31
32 scores, pred_bboxes = model(data)
33 _, predictions = scores.max(1)
34 correct += (predictions == targets).sum()
35 total += targets.size(0)
36
37 # Calculate bbox loss for monitoring (
38 # optional)
39 total_loss_bbox += criterion_bbox(pred_bboxes,
40 bboxes).item() * data.size(0)
41 total_samples += data.size(0)
42
43 avg_loss_bbox = total_loss_bbox / total_samples
44
45 print(f'Epoch {epoch+1}/{num_epochs}, Validation
46 Accuracy: {float(correct)/float(total)*100:.2f}%, '
47 f'Avg. Bbox Loss: {avg_loss_bbox:.4f}')
```



Hình 41.3: Kết quả của mô hình sau khi huấn luyện 10 epoch.

### 41.2.3 Bài toán 3: Classification (> 2 objects in an image) + Bounding Box Regression

#### Giới thiệu

Trong bài toán thứ ba, chúng ta mở rộng phạm vi của **classification** bằng cách xử lý hình ảnh **chứa nhiều hơn hai object** và thực hiện **bounding box regression** để định vị các object này. Bài toán này đòi hỏi mô hình không chỉ phân loại các object mà còn xác định chính xác vị trí của từng object trong hình ảnh.

#### Import và Tải Dữ liệu

Đầu tiên, chúng ta sẽ import các thư viện cần thiết và tải bộ dữ liệu từ kagglehub.

```
1 import kagglehub
2
3 # Download latest version
4 data_dir = kagglehub.dataset_download("andrewmvd/dog-and-
 cat-detection")
5 print("Path to dataset files:", data_dir)
6
```

#### Import Các Thư Viện Cần Thiết

Tiếp theo, chúng ta sẽ import các thư viện cần thiết để xử lý dữ liệu, xây dựng mô hình và thực hiện các thao tác đánh giá.

```
1 import os
2 import torch
3 import random
4 import numpy as np
5 import pandas as pd
6 import seaborn as sns
7 import torch.nn as nn
8 import torch.optim as optim
```

```
9 import matplotlib.pyplot as plt
10 import torch.nn.functional as F
11 import matplotlib.patches as patches
12 import xml.etree.ElementTree as ET
13 import tqdm.notebook as tqdm
14
15 from PIL import Image
16 from torchvision import transforms, models
17 from torch.utils.data import Dataset, DataLoader
18 from sklearn.metrics import confusion_matrix
19 from sklearn.model_selection import train_test_split
20 from torchvision.models.resnet import ResNet18_Weights,
 ResNet50_Weights
21
```

## Định Nghĩa Class Dataset với Nhiều object

Chúng ta sẽ định nghĩa một lớp MyDataset để xử lý các hình ảnh chứa nhiều object và thực hiện các thao tác như merge hình ảnh, chia thành các patch, và chuẩn bị dữ liệu cho mô hình.

```
1 class MyDataset(Dataset):
2 def __init__(self, annotations_dir, image_dir,
3 transform=None):
4 self.annotations_dir = annotations_dir
5 self.image_dir = image_dir
6 self.transform = transform
7 self.image_files = self.
8 filter_images_with_multiple_objects()
9
10 def filter_images_with_multiple_objects(self):
11 valid_image_files = []
12 for f in os.listdir(self.image_dir):
13 if os.path.isfile(os.path.join(self.image_dir
14 , f)):
15 img_name = f
16 annotation_name = os.path.splitext(
17 img_name)[0] + ".xml"
18 annotation_path = os.path.join(self.
19 annotations_dir, annotation_name)
20
```

```
16 if self.count_objects_in_annotation(
17 annotation_path) == 1:
18 valid_image_files.append(img_name)
19 return valid_image_files
20
21 def count_objects_in_annotation(self, annotation_path):
22 try:
23 tree = ET.parse(annotation_path)
24 root = tree.getroot()
25 count = 0
26 for obj in root.findall("object"):
27 count += 1
28 return count
29 except FileNotFoundError:
30 return 0
31
32 def parse_annotation(self, annotation_path):
33 tree = ET.parse(annotation_path)
34 root = tree.getroot()
35
36 # Get image size for normalization
37 image_width = int(root.find("size/width").text)
38 image_height = int(root.find("size/height").text)
39
40 label = None
41 bbox = None
42 for obj in root.findall("object"):
43 name = obj.find("name").text
44 if label is None: # Take the first label
45 label = name
46 # Get bounding box coordinates
47 xmin = int(obj.find("bndbox/xmin").text)
48 ymin = int(obj.find("bndbox/ymin").text)
49 xmax = int(obj.find("bndbox/xmax").text)
50 ymax = int(obj.find("bndbox/ymax").text)
51
52 # Normalize bbox coordinates to [0, 1]
53 bbox = [
54 xmin / image_width,
55 ymin / image_height,
56 xmax / image_width,
57 ymax / image_height,
58]
```

```
58 # Convert label to numerical representation (0
59 for cat, 1 for dog)
60 label_num = 0 if label == "cat" else 1 if label
61 == "dog" else -1
62
63 return label_num, torch.tensor(bbox, dtype=torch.
64 float32)
65
66 def __len__(self):
67 return len(self.image_files)
68
69 def __getitem__(self, idx):
70 img1_file = self.image_files[idx]
71 img1_path = os.path.join(self.image_dir,
72 img1_file)
73
74 annotation_name = os.path.splitext(img1_file)[0]
75 + ".xml"
76 img1_annotations = self.parse_annotation(
77 os.path.join(self.annotations_dir,
78 annotation_name)
79)
80
81 idx2 = random.randint(0, len(self.image_files) -
82 1)
83 img2_file = self.image_files[idx2]
84 img2_path = os.path.join(self.image_dir,
85 img2_file)
86
87 annotation_name = os.path.splitext(img2_file)[0]
88 + ".xml"
89 img2_annotations = self.parse_annotation(
90 os.path.join(self.annotations_dir,
91 annotation_name)
92)
93
94 img1 = Image.open(img1_path).convert("RGB")
95 img2 = Image.open(img2_path).convert("RGB")
96
97 # Horizontal merge
98 merged_image = Image.new(
99 "RGB", (img1.width + img2.width, max(img1.
height, img2.height))
```

```
91)
92 merged_image.paste(img1, (0, 0))
93 merged_image.paste(img2, (img1.width, 0))
94 merged_w = img1.width + img2.width
95 merged_h = max(img1.height, img2.height)
96
97 merged_annotations = []
98
99 # No change for objects from img1, already
100 normalized
101 merged_annotations.append(
102 {"bbox": img1_annotations[1].tolist(), "label":
103 img1_annotations[0]}
104)
105
106 # Adjust bbox coordinates for objects from img2
107 AND normalize
108 new_bbox = [
109 (img2_annotations[1][0] * img2.width + img1.
110 width) / merged_w, # Normalize xmin
111 img2_annotations[1][1] * img2.height /
112 merged_h, # Normalize ymin
113 (img2_annotations[1][2] * img2.width + img1.
114 width) / merged_w, # Normalize xmax
115 img2_annotations[1][3] * img2.height /
116 merged_h, # Normalize ymax
117]
118
119 merged_annotations.append({"bbox": new_bbox, "
120 "label": img2_annotations[0]})
121
122 # Convert merged image to tensor
123 if self.transform:
124 merged_image = self.transform(merged_image)
125 else:
126 merged_image = transforms.ToTensor()(merged_image)
127
128 # Convert annotations to 1D tensors, with shape
129 # (4,) for bbox and (1,) for label
130 annotations = torch.zeros((len(merged_annotations),
131 5))
132 for i, ann in enumerate(merged_annotations):
```

```

123 annotations[i] = torch.cat((torch.tensor(ann[
124 "bbox"]),
125 torch.tensor([ann["label"]]))))
126
127 return merged_image, annotations
128

```

Trong class MyDataset, chúng ta thực hiện các bước sau:

- **Merge hình ảnh:** Chọn ngẫu nhiên hai hình ảnh và ghép chúng lại thành một hình ảnh duy nhất bằng cách dán hai hình ảnh cạnh nhau.
- **Điều chỉnh annotations:** Điều chỉnh tọa độ bounding box của object trong patch mới sau khi ghép hình ảnh.
- **Trả về dữ liệu:** Mỗi mẫu dữ liệu bao gồm 4 patch hình ảnh và các annotations tương ứng cho từng patch.
- **Trình bày dữ liệu:** Dữ liệu trả về sẽ có dạng (bounding box, class), với shape là [2, 5].

## Phân Tích và Chuẩn Bị Dữ Liệu

Sau khi định nghĩa lớp dataset, chúng ta tiến hành phân tích và chuẩn bị dữ liệu cho bài toán này.

```

1 # Data directory
2 annotations_dir = os.path.join(data_dir, 'annotations')
3 image_dir = os.path.join(data_dir, 'images')
4
5 # Define transformations
6 transform = transforms.Compose([
7 transforms.Resize((224, 224)),
8 transforms.ToTensor()
9])
10
11 # Create dataset and dataloaders
12 dataset = MyDataset(annotations_dir, image_dir, transform
13 =transform)
14 train_dataset, val_dataset = train_test_split(dataset,
15 test_size=0.2, random_state=42)

```

```
14 train_loader = DataLoader(train_dataset, batch_size=8,
15 shuffle=True)
15 val_loader = DataLoader(val_dataset, batch_size=8,
16 shuffle=False)
16
```

## Xây Dựng Model với Hai Head

Chúng ta sẽ xây dựng một mô hình có hai head: một để **classification** và một để dự đoán **bounding box**. Trong trường hợp này, chúng ta sử dụng ResNet50 làm backbone cho mô hình.

```
1 class SimpleYOLO(nn.Module):
2 def __init__(self, num_classes):
3 super(SimpleYOLO, self).__init__()
4 self.backbone = models.resnet50(weights=
ResNet50_Weights.DEFAULT)
5 self.num_classes = num_classes
6
7 # Remove the final classification layer of ResNet
8 self.backbone = nn.Sequential(*list(self.backbone
9 .children())[:-2])
10
11 # Add the YOLO head
12 self.fcs = nn.Linear(
13 2048, 2 * 2 * (4 + self.num_classes)
14) # 2 is for the number of grid cell
15
16 def forward(self, x):
17 # x shape: (batch_size, C, H, W)
18 features = self.backbone(x)
19 features = F.adaptive_avg_pool2d(features, (1, 1))
20 # shape: (batch_size, 2048, 1, 1)
21 features = features.view(features.size(0), -1) #
22 features = self.fcs(features)
23
24
25 return features
```

## Cài Đặt Model

Chúng ta sẽ khởi tạo mô hình, thiết lập thiết bị, loss function và optimizer.

```
1 # Initialize model, criterion, and optimizer
2 device = torch.device('cuda' if torch.cuda.is_available()
3 else 'cpu')
4 num_classes = 2 # Assuming two classes: dog and cat
5 class_to_idx = {'dog': 0, 'cat': 1}
6
7 model = SimpleYOLO(num_classes=num_classes).to(device)
8 optimizer = optim.Adam(model.parameters(), lr=0.001)
```

## Training Loop

Chúng ta sẽ huấn luyện mô hình với cả hai head và theo dõi hiệu suất trên tập kiểm tra (validation set).

```
1 def calculate_loss(output, targets, device, num_classes):
2 mse_loss = nn.MSELoss()
3 ce_loss = nn.CrossEntropyLoss()
4
5 batch_size = output.shape[0]
6 total_loss = 0
7
8 output = output.view(batch_size, 2, 2, 4 +
9 num_classes) # Reshape to (batch_size, grid_y, grid_x,
10 4 + num_classes)
11
12 for i in range(batch_size): # Iterate through each
13 image in the batch
14 for j in range(len(targets[i])): # Iterate
15 through objects in the image
16
17 # Determine which grid cell the object's
18 center falls into
19 # Assuming bbox coordinates are normalized to
20 [0, 1]
```

```

15 bbox_center_x = (targets[i][j][0] + targets[i]
16][j][2]) / 2
17 bbox_center_y = (targets[i][j][1] + targets[i]
18][j][3]) / 2
19
20 grid_x = int(bbox_center_x * 2) # Multiply
21 by number of grid cells (2 in this case)
22 grid_y = int(bbox_center_y * 2)
23
24 # 1. Classification Loss for the responsible
25 grid cell
26 # Convert label to one-hot encoding only for
27 this example
28 # One hot encoding
29 label_one_hot = torch.zeros(num_classes,
device=device)
30 label_one_hot[int(targets[i][j][4])] = 1
31
32 # Classification loss (using CrossEntropyLoss
33)
34 classification_loss = ce_loss(output[i,
grid_y, grid_x, 4:], label_one_hot)
35
36 # 2. Regression Loss for the responsible grid
37 cell
38 bbox_target = targets[i][j][:4].to(device)
39 regression_loss = mse_loss(output[i, grid_y,
grid_x, :4], bbox_target)
40
41 # 3. No Object Loss (for other grid cells)
42 no_obj_loss = 0
43 for other_grid_y in range(2):
44 for other_grid_x in range(2):
45 if other_grid_y != grid_y or
46 other_grid_x != grid_x:
47 # MSE loss for predicting no
48 object (all zeros)
49 no_obj_loss += mse_loss(output[i,
50 other_grid_y, other_grid_x, :4], torch.zeros(4,
device=device))
51
52 total_loss += classification_loss +
53 regression_loss + no_obj_loss

```

```
44 return total_loss / batch_size # Average loss over
45 the batch
46
46 def evaluate_model(model, data_loader, device,
47 num_classes):
48 model.eval()
49 running_loss = 0.0
50 all_predictions = []
51 all_targets = []
52
52 with torch.no_grad():
53 for images, targets in tqdm.tqdm(data_loader,
54 desc="Validation", leave=False):
55 images = images.to(device)
56
56 output = model(images)
57
58 total_loss = calculate_loss(output, targets,
59 device, num_classes)
60 running_loss += total_loss.item()
61
61 # Reshape output to (batch_size, grid_y,
62 grid_x, 4 + num_classes)
63 output = output.view(images.shape[0], 2, 2, 4
63 + num_classes)
64
64 # Collect predictions and targets for mAP
65 # calculation
65 for batch_idx in range(images.shape[0]):
66 for target in targets[batch_idx]:
67 # Determine responsible grid cell
68 bbox_center_x = (target[0] + target
69 [2]) / 2
70 bbox_center_y = (target[1] + target
71 [3]) / 2
72 grid_x = int(bbox_center_x * 2)
73 grid_y = int(bbox_center_y * 2)
74
74 # Class prediction (index of max
75 probability)
75 prediction = output[batch_idx, grid_y
76 , grid_x, 4:].argmax().item()
76 all_predictions.append(prediction)
```

```
77 all_targets.append(target[4].item())
78
79 val_loss = running_loss / len(data_loader)
80
81 # Convert lists to tensors for PyTorch's metric
82 # functions
83 all_predictions = torch.tensor(all_predictions,
84 device=device)
85 all_targets = torch.tensor(all_targets, device=device)
86
87 # Calculate accuracy
88 val_accuracy = (all_predictions == all_targets).float()
89 .mean()
90
91 return val_loss, val_accuracy.item()
92
93 def train_model(
94 model, train_loader, val_loader, optimizer,
95 num_epochs, device, num_classes
96):
97 best_val_accuracy = 0.0
98 train_losses = []
99 val_losses = []
100 train_accuracies = []
101 val_accuracies = []
102
103 for epoch in tqdm.tqdm(range(num_epochs), desc="Epochs"):
104 model.train()
105 running_loss = 0.0
106
107 for images, targets in tqdm.tqdm(train_loader,
108 desc="Batches", leave=False):
109 images = images.to(device)
110
111 optimizer.zero_grad()
112 output = model(images)
113
114 total_loss = calculate_loss(output, targets,
115 device, num_classes)
116
117 total_loss.backward()
118 optimizer.step()
```

```

113 running_loss += total_loss.item()
114
115 epoch_loss = running_loss / len(train_loader)
116 train_losses.append(epoch_loss)
117
118 # Validation
119 val_loss, val_accuracy = evaluate_model(model,
120 val_loader, device, num_classes)
121 val_losses.append(val_loss)
122 val_accuracies.append(val_accuracy)
123
124 print(
125 f"Epoch {epoch+1}/{num_epochs}, Train Loss: {epoch_loss:.4f}, Validation Loss: {val_loss:.4f},
126 Validation Accuracy: {val_accuracy:.4f}"
127)
128
129 # Save the best model
130 if val_accuracy > best_val_accuracy:
131 best_val_accuracy = val_accuracy
132 torch.save(model.state_dict(), "best_model.
133 pth")
134
135 return train_losses, val_losses, train_accuracies,
136 val_accuracies
137

```

## Inference

Sau khi huấn luyện, chúng ta có thể sử dụng mô hình để thực hiện dự đoán trên các hình ảnh mới.

```

1 def inference(model, image_path, transform, device,
2 class_to_idx, threshold=0.5):
3 model.eval()
4 image = Image.open(image_path).convert("RGB")
5 original_width, original_height = image.size
6
6 # Resize the image to match the input size expected
7 # by the model (e.g., 448x448)
8 resized_image = image.resize((448, 448))

```

```
8 resized_width, resized_height = resized_image.size
9
10 # Apply the same transformations used during training
11 transformed_image = transform(resized_image).
12 unsqueeze(0).to(device)
13
14 with torch.no_grad():
15 output = model(transformed_image)
16 output = output.view(1, 2, 2, 4 + len(
17 class_to_idx)) # Reshape for 2x2 grid
18
19 fig, ax = plt.subplots(1)
20 ax.axis("off")
21 ax.imshow(resized_image) # Display resized image
22
23 for grid_y in range(2):
24 for grid_x in range(2):
25 # Get the class prediction and bounding box
26 # for the current grid cell
27 class_pred = output[0, grid_y, grid_x, 4:].
28 argmax().item()
29 bbox = output[0, grid_y, grid_x, :4].tolist()
30 # Predicted bbox
31
32 # Confidence (probability of the predicted
33 # class)
34 confidence = torch.softmax(output[0, grid_y,
35 grid_x, 4:], dim=0)[
36 class_pred
37].item()
38
38
39 # Scale the bounding box back to the resized
40 # image size
41 # Assuming bbox coordinates are normalized to
42 # [0, 1] within the grid cell
43 x_min = bbox[0] * (resized_width / 2) +
44 grid_x * (resized_width / 2)
45 y_min = bbox[1] * (resized_height / 2) +
46 grid_y * (resized_height / 2)
47 x_max = bbox[2] * (resized_width / 2) +
48 grid_x * (resized_width / 2)
49 y_max = bbox[3] * (resized_height / 2) +
50 grid_y * (resized_height / 2)
```

```
39 # Draw the bounding box and label on the
40 image if confidence is above threshold
41 if confidence > threshold:
42 rect = patches.Rectangle(
43 (x_min, y_min),
44 x_max - x_min,
45 y_max - y_min,
46 linewidth=1,
47 edgecolor="r",
48 facecolor="none",
49)
50 ax.add_patch(rect)
51 plt.text(
52 x_min,
53 y_min,
54 f"{list(class_to_idx.keys())[class_pred]}: {confidence:.2f}",
55 color="white",
56 fontsize=12,
57 bbox=dict(facecolor="red", alpha=0.5)
58 ,
59)
60
61 plt.show()
62
63 # Load the best model
64 model.load_state_dict(torch.load("best_model.pth"))
65
66 # Inference on a sample image
67 # image_path = os.path.join(image_dir, "cat.100.jpg")
68 image_path = "/mnt/c/Study/OD Project/good_1.jpg"
69 inference(model, image_path, transform, device,
70 class_to_idx, threshold=0.5)
71
```

## Kết luận

Thông qua bài toán thứ ba, chúng ta đã mở rộng khả năng của mô hình trong việc xử lý các hình ảnh chứa nhiều object. Bằng cách kết hợp **classification** và **bounding box regression**, chúng ta có thể không chỉ phân loại các object mà còn xác định chính xác vị trí của chúng trong hình ảnh. Việc sử



Hình 41.4: Dự đoán của mô hình sau khi huấn luyện 10 epoch.

dụng các kỹ thuật như patching và two-headed models giúp cải thiện hiệu suất và khả năng mở rộng của hệ thống.

#### 41.2.4 Bài toán 4: YOLOv1

Trong bài toán thứ tư, chúng ta sẽ triển khai mô hình **YOLOv1** cho bài toán **Object Detection**. YOLOv1 là một trong những mô hình tiên tiến nhất cho việc phát hiện và định vị đối tượng trong hình ảnh với tốc độ nhanh và hiệu quả cao. Mô hình này sử dụng một mạng neural đơn giản để dự đoán các bounding box và lớp của chúng trong một lần duy nhất, giúp giảm thiểu thời gian xử lý so với các phương pháp truyền thống.

##### Giới thiệu về YOLOv1

YOLOv1 (You Only Look Once version 1) là một kiến trúc mạng neural được thiết kế để thực hiện **Object Detection** một cách hiệu quả. Thay vì phân chia hình ảnh thành nhiều vùng và xử lý từng vùng một cách riêng biệt như các phương pháp trước đây, YOLOv1 xử lý toàn bộ hình ảnh cùng một lúc. Điều này không chỉ tăng tốc độ xử lý mà còn cải thiện độ chính xác trong việc phát hiện và định vị các đối tượng.

Mô hình YOLOv1 được xây dựng dựa trên kiến trúc **Darknet**, một mạng convolutional mạnh mẽ với các lớp CNN sâu, kết hợp với các lớp fully connected để dự đoán các bounding box và xác suất lớp cho từng grid cell trong hình ảnh.

##### Import và Tải Dữ liệu

Đầu tiên, chúng ta cần import các thư viện cần thiết và tải dữ liệu từ bộ dữ liệu VOC (Visual Object Classes). Bộ dữ liệu này cung cấp các hình ảnh cùng với các annotations về các đối tượng trong hình ảnh, phù hợp cho việc huấn luyện và đánh giá mô hình **Object Detection**.

```
1 import os
2
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 from matplotlib import patches
7 from collections import Counter
8 import cv2
```

```
9 from glob import glob
10 from tqdm import tqdm
11 from termcolor import colored
12
13 import torch
14 from torch import nn
15 from torch import optim
16 from torch.utils.data import DataLoader
17
18 import torchvision
19 from torchvision import transforms
20
21 import albumentations as A
22 from albumentations.pytorch import ToTensorV2
23
```

## Định Nghĩa Dataset Custom cho YOLOv1

Chúng ta sẽ định nghĩa một lớp dataset tùy chỉnh `CustomVOCDataset` kế thừa từ `torchvision.datasets.VOCDetection`. Lớp này sẽ xử lý việc chuyển đổi annotations từ định dạng VOC sang định dạng YOLO, chuẩn bị dữ liệu cho mô hình YOLOv1.

```
1 class CustomVOCDataset(torchvision.datasets.VOCDetection):
2 :
3 def __init__(self, class_mapping, S=7, B=2, C=20, custom_transforms=None):
4 # Initialize YOLO-specific configuration parameters.
5 self.S = S # Grid size S x S
6 self.B = B # Number of bounding boxes
7 self.C = C # Number of classes
8 self.class_mapping = class_mapping # Mapping of class names to class indices
9 self.custom_transforms = custom_transforms
10
11 def __getitem__(self, index):
12 # Get an image and its target (annotations) from the VOC dataset.
```

```
12 image, target = super(CustomVOCDataset, self).
13 __getitem__(index)
14 img_width, img_height = image.size
15
16 # Convert target annotations to YOLO format
17 # bounding boxes.
18 boxes = convert_to_yolo_format(target, img_width,
19 img_height, self.class_mapping)
20
21 just_boxes = boxes[:,1:]
22 labels = boxes[:,0]
23
24 # Tranforme
25 if self.custom_transforms:
26 sample = {
27 'image': np.array(image),
28 'bboxes': just_boxes,
29 'labels': labels
30 }
31
32 sample = self.custom_transforms(**sample)
33 image = sample['image']
34 boxes = sample['bboxes']
35 labels = sample['labels']
36
37 # Create an empty label matrix for YOLO ground
38 # truth.
39 label_matrix = torch.zeros((self.S, self.S, self.
40 C + 5 * self.B))
41
42 boxes = torch.tensor(boxes, dtype=torch.float32)
43 labels = torch.tensor(labels, dtype=torch.float32)
44)
45 image = torch.as_tensor(image, dtype=torch.
46 float32)
47
48 # Iterate through each bounding box in YOLO
49 # format.
50 for box, class_label in zip(boxes, labels):
51 x, y, width, height = box.tolist()
52 class_label = int(class_label)
53
54 # Calculate the grid cell (i, j) that this
55 # box belongs to.
```

```
47 i, j = int(self.S * y), int(self.S * x)
48 x_cell, y_cell = self.S * x - j, self.S * y -
49 i
50
51 # Calculate the width and height of the box
52 # relative to the grid cell.
53 width_cell, height_cell = (
54 width * self.S,
55 height * self.S,
56)
57
58 # If no object has been found in this
59 # specific cell (i, j) before:
60 if label_matrix[i, j, 20] == 0:
61 # Mark that an object exists in this cell
62
63 label_matrix[i, j, 20] = 1
64
65 # Store the box coordinates as an offset
66 # from the cell boundaries.
67 box_coordinates = torch.tensor(
68 [x_cell, y_cell, width_cell,
69 height_cell]
70)
71
72 # Set the box coordinates in the label
73 # matrix.
74 label_matrix[i, j, 21:25] =
75 box_coordinates
76
77 # Set the one-hot encoding for the class
78 # label.
79 label_matrix[i, j, class_label] = 1
80
81 return image, label_matrix
82
83
```

Trong lớp CustomVOCdataset, chúng ta thực hiện các bước sau:

- **Khởi tạo:** Xác định các tham số đặc thù của YOLO như kích thước grid (S), số lượng bounding boxes mỗi grid cell (B), và số lớp (C).
- **Chuyển đổi annotations:** Chuyển đổi annotations từ định dạng VOC

sang định dạng YOLO, bao gồm việc tính toán tọa độ trung tâm và kích thước của bounding box, cũng như ánh xạ lớp đối tượng.

- **Áp dụng biến đổi:** Sử dụng các biến đổi dữ liệu (transforms) để tăng cường dữ liệu và chuẩn bị hình ảnh cho mô hình.
- **Tạo ma trận nhãn:** Tạo một ma trận nhãn rỗng cho YOLO, sau đó điền thông tin về các bounding box và lớp đối tượng vào ma trận này.

### Chuyển Đổi Annotations sang Định Dạng YOLO

Hàm `convert_to_yolo_format` chịu trách nhiệm chuyển đổi annotations từ định dạng VOC sang định dạng YOLO, bao gồm việc tính toán tọa độ trung tâm, chiều rộng, chiều cao của bounding box và ánh xạ lớp đối tượng.

```
1 def convert_to_yolo_format(target, img_width, img_height,
2 class_mapping):
3 """
4 Convert annotation data from VOC format to YOLO
5 format.
6
7 Parameters:
8 target (dict): Annotation data from VOCDetection
9 dataset.
10 img_width (int): Width of the original image.
11 img_height (int): Height of the original image.
12 class_mapping (dict): Mapping from class names to
13 integer IDs.
14
15 Returns:
16 torch.Tensor: Tensor of shape [N, 5] for N bounding
17 boxes,
18 each with [class_id, x_center, y_center
19 , width, height].
20 """
21 # Extract the list of annotations from the target
22 # dictionary.
23 annotations = target['annotation']['object']
24
25 # Get the real width and height of the image from the
26 # annotation.
```

```
19 real_width = int(target['annotation']['size'][‘width’])
20 real_height = int(target['annotation']['size'][‘height’])
21
22 # Ensure that annotations is a list, even if there’s
23 # only one object.
24 if not isinstance(annotations, list):
25 annotations = [annotations]
26
27 # Initialize an empty list to store the converted
28 # bounding boxes.
29 boxes = []
30
31 # Loop through each annotation and convert it to YOLO
32 # format.
33 for anno in annotations:
34 xmin = int(anno[‘bndbox’][‘xmin’]) / real_width
35 xmax = int(anno[‘bndbox’][‘xmax’]) / real_width
36 ymin = int(anno[‘bndbox’][‘ymin’]) / real_height
37 ymax = int(anno[‘bndbox’][‘ymax’]) / real_height
38
39 # Calculate the center coordinates, width, and
40 # height of the bounding box.
41 x_center = (xmin + xmax) / 2
42 y_center = (ymin + ymax) / 2
43 width = xmax - xmin
44 height = ymax - ymin
45
46 # Retrieve the class name from the annotation and
47 # map it to an integer ID.
48 class_name = anno[‘name’]
49 class_id = class_mapping[class_name] if
50 class_name in class_mapping else 0
51
52 # Append the YOLO formatted bounding box to the
53 # list.
54 boxes.append([class_id, x_center, y_center, width,
55 height])
56
57 # Convert the list of boxes to a torch tensor.
58 return np.array(boxes)
```

## Định Nghĩa Hàm Tính IoU và Non-Maximum Suppression

Các hàm `intersection_over_union` và `non_max_suppression` được sử dụng để tính toán **Intersection over Union (IoU)** giữa các bounding boxes và thực hiện **Non-Maximum Suppression (NMS)** để loại bỏ các bounding boxes trùng lặp, chỉ giữ lại các bounding boxes có độ tin cậy cao nhất.

```
1 def intersection_over_union(boxes_preds, boxes_labels,
2 box_format="midpoint"):
3 """
4 Calculate the Intersection over Union (IoU) between
5 bounding boxes.
6
7 Parameters:
8 boxes_preds (tensor): Predicted bounding boxes (
9 BATCH_SIZE, 4)
10 boxes_labels (tensor): Ground truth bounding
11 boxes (BATCH_SIZE, 4)
12 box_format (str): Box format, can be "midpoint"
13 or "corners".
14
15 Returns:
16 tensor: Intersection over Union scores for each
17 example.
18 """
19
20 # Check if the box format is "midpoint"
21 if box_format == "midpoint":
22 # Calculate coordinates of top-left (x1, y1) and
23 # bottom-right (x2, y2) points for predicted boxes
24 box1_x1 = boxes_preds[..., 0:1] - boxes_preds
25 [..., 2:3] / 2
26 box1_y1 = boxes_preds[..., 1:2] - boxes_preds
27 [..., 3:4] / 2
28 box1_x2 = boxes_preds[..., 0:1] + boxes_preds
29 [..., 2:3] / 2
30 box1_y2 = boxes_preds[..., 1:2] + boxes_preds
31 [..., 3:4] / 2
32
33 # Calculate coordinates of top-left (x1, y1) and
34 # bottom-right (x2, y2) points for ground truth boxes
35 box2_x1 = boxes_labels[..., 0:1] - boxes_labels
36 [..., 2:3] / 2
```

```
24 box2_y1 = boxes_labels[..., 1:2] - boxes_labels
25 [..., 3:4] / 2
26 box2_x2 = boxes_labels[..., 0:1] + boxes_labels
27 [..., 2:3] / 2
28 box2_y2 = boxes_labels[..., 1:2] + boxes_labels
29 [..., 3:4] / 2
30
31 # Check if the box format is "corners"
32 if box_format == "corners":
33 # Extract coordinates for predicted boxes
34 box1_x1 = boxes_preds[..., 0:1]
35 box1_y1 = boxes_preds[..., 1:2]
36 box1_x2 = boxes_preds[..., 2:3]
37 box1_y2 = boxes_preds[..., 3:4]
38
39 # Extract coordinates for ground truth boxes
40 box2_x1 = boxes_labels[..., 0:1]
41 box2_y1 = boxes_labels[..., 1:2]
42 box2_x2 = boxes_labels[..., 2:3]
43 box2_y2 = boxes_labels[..., 3:4]
44
45 # Calculate coordinates of the intersection rectangle
46 x1 = torch.max(box1_x1, box2_x1)
47 y1 = torch.max(box1_y1, box2_y1)
48 x2 = torch.min(box1_x2, box2_x2)
49 y2 = torch.min(box1_y2, box2_y2)
50
51 # Compute the area of the intersection rectangle,
52 # clamp(0) to handle cases where they do not overlap
53 intersection = (x2 - x1).clamp(0) * (y2 - y1).clamp(0)
54
55 # Calculate the areas of the predicted and ground
56 # truth boxes
57 box1_area = abs((box1_x2 - box1_x1) * (box1_y2 - box1_y1))
58 box2_area = abs((box2_x2 - box2_x1) * (box2_y2 - box2_y1))
59
60 # Calculate the Intersection over Union, adding a
61 # small epsilon to avoid division by zero
62 return intersection / (box1_area + box2_area -
63 intersection + 1e-6)
```



```
30 # Remove bounding boxes with IoU greater than the
31 # specified threshold with the chosen box
32 bboxes = [
33 box
34 for box in bboxes
35 if box[0] != chosen_box[0]
36 or intersection_over_union(
37 torch.tensor(chosen_box[2:]),
38 torch.tensor(box[2:]),
39 box_format=box_format,
40)
41 < iou_threshold
42]
43
44 # Add the chosen bounding box to the list after
45 # NMS
46 bboxes_after_nms.append(chosen_box)
47
48 # Return the list of bounding boxes after NMS
49 return bboxes_after_nms
```

## Tính Toán Trung Bình Đô Chính Xác (mAP)

Hàm `mean_average_precision` được sử dụng để tính toán giá trị trung bình của độ chính xác (mAP) cho mô hình, một chỉ số quan trọng đánh giá hiệu quả của các mô hình **Object Detection**.

```
1 def mean_average_precision(
2 pred_boxes, true_boxes, iou_threshold=0.5, box_format=
3 "midpoint", num_classes=20):
4 """
5 Calculate the mean average precision (mAP).
6
7 Parameters:
8 pred_boxes (list): A list containing predicted
9 bounding boxes with each box defined as [train_idx,
10 class_pred, prob_score, x1, y1, x2, y2].
11 true_boxes (list): Similar to pred_boxes but
12 containing information about true boxes.
13
14 Returns:
15 float: mAP value.
16
17 """
18
19 # Implement mAP calculation logic here.
20
21 return mAP
```

```

9 iou_threshold (float): IoU threshold, where
10 predicted boxes are considered correct.
11 box_format (str): "midpoint" or "corners" used to
12 specify the format of the boxes.
13 num_classes (int): Number of classes.
14
15 Returns:
16 float: The mAP value across all classes with a
17 specific IoU threshold.
18 """
19
20
21 # List to store mAP for each class
22 average_precisions = []
23
24 # Small epsilon to stabilize division
25 epsilon = 1e-6
26
27 for c in range(num_classes):
28 detections = []
29 ground_truths = []
30
31 # Iterate through all predictions and targets,
32 # and only add those belonging to
33 # the current class 'c'.
34 for detection in pred_boxes:
35 if detection[1] == c:
36 detections.append(detection)
37
38 for true_box in true_boxes:
39 if true_box[1] == c:
40 ground_truths.append(true_box)
41
42 # Find the number of boxes for each training
43 # example.
44 # The Counter here counts the number of target
45 # boxes we have
46 # for each training example, so if image 0 has 3,
47 # and image 1 has 5,
48 # we'll have a dictionary like:
49 # amount_bboxes = {0: 3, 1: 5}
50 amount_bboxes = Counter([gt[0] for gt in
51 ground_truths])

```

```
44 # We then loop through each key, val in this
45 # dictionary and convert it to the following (for the
46 # same example):
47 # amount_bboxes = {0: torch.tensor([0, 0, 0]), 1:
48 # torch.tensor([0, 0, 0, 0, 0])}
49 for key, val in amount_bboxes.items():
50 amount_bboxes[key] = torch.zeros(val)
51
52 # Sort by box probability, index 2 is the
53 # probability
54 detections.sort(key=lambda x: x[2], reverse=True)
55 TP = torch.zeros((len(detections)))
56 FP = torch.zeros((len(detections)))
57 total_true_bboxes = len(ground_truths)
58
59 # If there are no ground truth boxes for this
60 # class, it can be safely skipped
61 if total_true_bboxes == 0:
62 continue
63
64 for detection_idx, detection in enumerate(
65 detections):
66 # Only consider ground truth boxes with the
67 # same training index as the prediction
68 ground_truth_img = [
69 bbox for bbox in ground_truths if bbox[0]
70 == detection[0]
71]
72
73 num_gts = len(ground_truth_img)
74 best_iou = 0
75
76 for idx, gt in enumerate(ground_truth_img):
77 iou = intersection_over_union(
78 torch.tensor(detection[3:]),
79 torch.tensor(gt[3:]),
80 box_format=box_format,
81)
82
83 if iou > best_iou:
84 best_iou = iou
85 best_gt_idx = idx
86
87 if best_iou > iou_threshold:
```

```
80 # Only detect ground truth once
81 if amount_bboxes[detection[0]][
82 best_gt_idx] == 0:
83 # True positive and mark this
84 bounding_box as seen
85 TP[detection_idx] = 1
86 amount_bboxes[detection[0]][
87 best_gt_idx] = 1
88 else:
89 FP[detection_idx] = 1
90
91
92 # If IOU is lower, the detection result is
93 # false positive
94 else:
95 FP[detection_idx] = 1
96
97 TP_cumsum = torch.cumsum(TP, dim=0)
98 FP_cumsum = torch.cumsum(FP, dim=0)
99 recalls = TP_cumsum / (total_true_bboxes +
100 epsilon)
101 precisions = torch.divide(TP_cumsum, (TP_cumsum +
102 FP_cumsum + epsilon))
103 precisions = torch.cat((torch.tensor([1]),
104 precisions))
105 recalls = torch.cat((torch.tensor([0]), recalls))
106 # Use torch.trapz for numerical integration
107 average_precisions.append(torch.trapz(precisions,
108 recalls))
109
110 return sum(average_precisions) / len(
111 average_precisions)
```

## Định Nghĩa Kiến Trúc Mạng YOLOv1

Mô hình YOLOv1 được xây dựng dựa trên kiến trúc Darknet, bao gồm các lớp convolutional và max-pooling, kết hợp với các lớp fully connected để dự đoán bounding boxes và lớp đối tượng.

```
1 """
2 Information about the architectural configuration:
3 A Tuple is structured as (kernel_size, number of filters,
4 stride, padding).
5 "M" simply represents max-pooling with a 2x2 pool size
6 and 2x2 kernel.
7 The list is structured according to the data blocks, and
8 ends with an integer representing the number of
9 repetitions.
10 """
11
12 # Describing convolutional and max-pooling layers, as
13 # well as the number of repetitions for convolutional
14 # blocks.
15 architecture_config = [
16 (7, 64, 2, 3), # Convolutional block 1
17 "M", # Max-pooling layer 1
18 (3, 192, 1, 1), # Convolutional block 2
19 "M", # Max-pooling layer 2
20 (1, 128, 1, 0), # Convolutional block 3
21 (3, 256, 1, 1), # Convolutional block 4
22 (1, 256, 1, 0), # Convolutional block 5
23 (3, 512, 1, 1), # Convolutional block 6
24 "M", # Max-pooling layer 3
25 [(1, 256, 1, 0), (3, 512, 1, 1), 4], # Convolutional
26 # block 7 (repeated 4 times)
27 (1, 512, 1, 0), # Convolutional block 8
28 (3, 1024, 1, 1), # Convolutional block 9
29 "M", # Max-pooling layer 4
30 [(1, 512, 1, 0), (3, 1024, 1, 1), 2], # Convolutional
31 # block 10 (repeated 2 times)
32 (3, 1024, 1, 1), # Convolutional block 11
33 (3, 1024, 2, 1), # Convolutional block 12
34 (3, 1024, 1, 1), # Convolutional block 13
35 (3, 1024, 1, 1), # Convolutional block 14
36]
37
38 # A convolutional block is defined with Conv2d,
39 # BatchNorm2d, and LeakyReLU layers.
40 class CNNBlock(nn.Module):
41 def __init__(self, in_channels, out_channels, **kwargs):
42 super(CNNBlock, self).__init__()
```

```
34 self.conv = nn.Conv2d(in_channels, out_channels,
35 bias=False, **kwargs)
36 self.batchnorm = nn.BatchNorm2d(out_channels)
37 self.leakyrelu = nn.LeakyReLU(0.1)
38
39 def forward(self, x):
40 return self.leakyrelu(self.batchnorm(self.conv(x)))
41
42 # The YOLOv1 model is defined with convolutional layers
43 # and fully connected layers (fcs).
43 class Yolov1(nn.Module):
44 def __init__(self, in_channels=3, **kwargs):
45 super(Yolov1, self).__init__()
46 self.architecture = architecture_config
47 self.in_channels = in_channels
48 self.darknet = self._create_conv_layers(self.
49 architecture)
50 self.fcs = self._create_fcs(**kwargs)
51
52 def forward(self, x):
53 x = self.darknet(x)
54 return self.fcs(torch.flatten(x, start_dim=1))
55
56 # Function to create convolutional layers based on
57 # the predefined architecture.
58 def _create_conv_layers(self, architecture):
59 layers = []
60 in_channels = self.in_channels
61
62 for x in architecture:
63 if type(x) == tuple:
64 layers += [
65 CNNBlock(
66 in_channels, x[1], kernel_size=x
67 [0], stride=x[2], padding=x[3],
68)
69]
70 in_channels = x[1]
71
72 elif type(x) == str:
73 layers += [nn.MaxPool2d(kernel_size=(2,
74 2), stride=(2, 2))]
```

```
71 elif type(x) == list:
72 conv1 = x[0]
73 conv2 = x[1]
74 num_repeats = x[2]
75
76 for _ in range(num_repeats):
77 layers += [
78 CNNBlock(
79 in_channels,
80 conv1[1],
81 kernel_size=conv1[0],
82 stride=conv1[2],
83 padding=conv1[3],
84)
85]
86 layers += [
87 CNNBlock(
88 conv1[1],
89 conv2[1],
90 kernel_size=conv2[0],
91 stride=conv2[2],
92 padding=conv2[3],
93)
94]
95 in_channels = conv2[1]
96
97 return nn.Sequential(*layers)
98
99 # Function to create fully connected layers based on
100 # input parameters such as grid size, number of boxes,
101 # and number of classes.
102 def _create_fcs(self, split_size, num_boxes,
103 num_classes):
104 S, B, C = split_size, num_boxes, num_classes
105
106 return nn.Sequential(
107 nn.Flatten(),
108 nn.Linear(1024 * S * S, 4096),
109 nn.Dropout(0.0),
110 nn.LeakyReLU(0.1),
111 nn.Linear(4096, S * S * (C + B * 5)),
112)
```

Trong kiến trúc này:

- **Convolutional Layers:** Các lớp convolutional được xây dựng theo cấu hình đã định nghĩa trong `architecture_config`, bao gồm các lớp `CNNBlock` kết hợp với `BatchNorm2d` và `LeakyReLU`.
- **Fully Connected Layers:** Sau các lớp convolutional, các lớp fully connected được sử dụng để dự đoán các bounding boxes và lớp của chúng. Lớp đầu tiên `Linear` kết nối từ các đặc trưng đã trích xuất sang không gian cao hơn, sau đó sử dụng lớp `Dropout` để giảm thiểu overfitting và lớp `LeakyReLU` cho phép gradient flow tốt hơn.

## Định Nghĩa Hàm Mất Của YOLOv1

Hàm mất (`YoloLoss`) được thiết kế để tối ưu hóa các dự đoán của mô hình YOLOv1, bao gồm việc tính toán mất mát cho các bounding boxes, xác suất đối tượng và lớp của đối tượng.

```

1 class YoloLoss(nn.Module):
2 """
3 Calculate the loss for the YOLO (v1) model.
4 """
5
6 def __init__(self, S=7, B=2, C=20):
7 super(YoloLoss, self).__init__()
8 self.mse = nn.MSELoss(reduction="sum")
9
10 """
11 S is the grid size of the image (7),
12 B is the number of bounding boxes (2),
13 C is the number of classes (in VOC dataset, it's
14 20).
15 """
16 self.S = S
17 self.B = B
18 self.C = C
19
20 # These are YOLO-specific constants, representing
21 # the weight
22 # for no object loss (lambda_noobj) and box
23 # coordinates loss (lambda_coord).
```

```
21 self.lambda_noobj = 0.5
22 self.lambda_coord = 5
23
24 def forward(self, predictions, target):
25 # Reshape the predictions to the shape (
26 # BATCH_SIZE, S*S*(C+B*5))
27 predictions = predictions.reshape(-1, self.S,
28 self.S, self.C + self.B * 5)
29
30 # Calculate Intersection over Union (IoU) for the
31 # two predicted bounding boxes
32 # with the target bounding box.
33 iou_b1 = intersection_over_union(predictions[...,
34 21:25], target[..., 21:25])
35 iou_b2 = intersection_over_union(predictions[...,
36 26:30], target[..., 21:25])
37 ious = torch.cat([iou_b1.unsqueeze(0), iou_b2.
38 unsqueeze(0)], dim=0)
39
40 # Get the box with the highest IoU among the two
41 # predictions.
42 # Note that bestbox will have an index of 0 or 1,
43 # indicating which box is better.
44 iou_maxes, bestbox = torch.max(ious, dim=0)
45 exists_box = target[..., 20].unsqueeze(3) # This
46 # represents Iobj_i in the paper
47
48 # ===== #
49 # FOR BOX COORDINATES #
50 # ===== #
51
51 # Set the boxes with no objects to zero. Choose
52 # one of the two predictions
53 # based on the bestbox index calculated earlier.
54 box_predictions = exists_box * (
55 (
56 bestbox * predictions[..., 26:30]
57 + (1 - bestbox) * predictions[..., 21:25]
58)
59)
60
61 box_targets = exists_box * target[..., 21:25]
```

```
54 # Take the square root of width and height to
55 # ensure positive values.
56 box_predictions[..., 2:4] = torch.sign(
57 box_predictions[..., 2:4]) * torch.sqrt(
58 torch.abs(box_predictions[..., 2:4] + 1e-6)
59)
60 box_targets[..., 2:4] = torch.sqrt(box_targets
61 [..., 2:4])
62
63 box_loss = self.mse(
64 torch.flatten(box_predictions, end_dim=-2),
65 torch.flatten(box_targets, end_dim=-2),
66)
67
68 # ===== #
69 # FOR OBJECT LOSS #
70 # ===== #
71
72 # pred_box represents the confidence score of the
73 # box with the highest IoU.
74 pred_box =
75 bestbox * predictions[..., 25:26] + (1 -
76 bestbox) * predictions[..., 20:21]
77)
78
79 object_loss = self.mse(
80 torch.flatten(exists_box * pred_box),
81 torch.flatten(exists_box * target[...,
82 20:21]),
83)
84
85 # ===== #
86 # FOR NO OBJECT LOSS #
87 # ===== #
88
88 no_object_loss = self.mse(
89 torch.flatten((1 - exists_box) * predictions
90 [..., 20:21], start_dim=1),
91 torch.flatten((1 - exists_box) * target[...,
92 20:21], start_dim=1),
93)
94
95 no_object_loss += self.mse(
```

```
89 torch.flatten((1 - exists_box) * predictions
90 [..., 25:26], start_dim=1),
91 torch.flatten((1 - exists_box) * target[..., 20:21], start_dim=1)
92)
93
94 # ===== #
95 # FOR CLASS LOSS #
96 # ===== #
97
98 class_loss = self.mse(
99 torch.flatten(exists_box * predictions[..., :20], end_dim=-2),
100 torch.flatten(exists_box * target[..., :20], end_dim=-2),
101)
102
103 # Calculate the final loss by combining the above
104 # components.
105 loss = (
106 self.lambda_coord * box_loss # First term
107 + object_loss # Second term
108 + self.lambda_noobj * no_object_loss # Third
109 term
110 + class_loss # Fourth term
111)
112
113 return loss
```

## Cài Đặt và Huấn Luyện Mô Hình YOLOv1

Chúng ta sẽ cài đặt mô hình YOLOv1, thiết lập các siêu tham số, và tiến hành huấn luyện mô hình trên bộ dữ liệu VOC.

```
1 # Set the random seed for reproducibility.
2 seed = 123
3 torch.manual_seed(seed)
4
5 # Hyperparameters and configurations
6 # Learning rate for the optimizer.
```

```

7 LEARNING_RATE = 2e-5
8 # Specify whether to use "cuda" (GPU) or "cpu" for
 training.
9 DEVICE = "cuda"
10 # Originally 64 in the research paper, but using a
 smaller batch size due to GPU limitations.
11 BATCH_SIZE = 16
12 # Number of training epochs.
13 EPOCHS = 300
14 # Number of worker processes for data loading.
15 NUM_WORKERS = 2
16 # If True, DataLoader will pin memory to transfer data to
 the GPU faster.
17 PIN_MEMORY = True
18 # If False, the training process will not load a pre-
 trained model.
19 LOAD_MODEL = False
20 # Specify the file name for the pre-trained model if
 LOAD_MODEL is True.
21 LOAD_MODEL_FILE = "yolov1.pth.tar"
22

```

## Định Nghĩa Các Biến Đổi Dữ Liệu

Chúng ta sẽ sử dụng thư viện `albumentations` để áp dụng các biến đổi dữ liệu nhằm tăng cường dữ liệu và cải thiện khả năng tổng quát hóa của mô hình.

```

1 WIDTH = 448
2 HEIGHT = 448
3
4 def get_train_transforms():
5 return A.Compose([
6 A.OneOf([
7 A.HueSaturationValue(
8 hue_shift_limit=0.2, sat_shift_limit= 0.2,
9 val_shift_limit=0.2, p=0.9),
10 A.RandomBrightnessContrast(
11 brightness_limit=0.2, contrast_limit=0.2, p=0.9)],
12 p=0.9),
13 A.ToGray(p=0.01),
14 A.HorizontalFlip(p=0.2),
15]
16)
17
18

```

## Định Nghĩa Mapping Lớp Đối Tượng

Chúng ta định nghĩa một từ điển `class_mapping` để ánh xạ các tên lớp đối tượng trong bộ dữ liệu VOC sang các chỉ số số học, giúp dễ dàng xử lý trong quá trình huấn luyện.

```
1 class_mapping = {
2 'aeroplane': 0,
3 'bicycle': 1,
4 'bird': 2,
5 'boat': 3,
6 'bottle': 4,
7 'bus': 5,
8 'car': 6,
9 'cat': 7,
10 'chair': 8,
11 'cow': 9,
```

```
12 'diningtable': 10,
13 'dog': 11,
14 'horse': 12,
15 'motorbike': 13,
16 'person': 14,
17 'pottedplant': 15,
18 'sheep': 16,
19 'sofa': 17,
20 'train': 18,
21 'tvmonitor': 19
22 }
23
```

## Định Nghĩa Hàm Huấn Luyện và Đánh Giá

Chúng ta sẽ định nghĩa các hàm `train_fn` và `test_fn` để thực hiện quá trình huấn luyện và đánh giá mô hình trên tập dữ liệu huấn luyện và tập dữ liệu kiểm tra.

```
1 def train_fn(train_loader, model, optimizer, loss_fn,
2 epoch):
3 mean_loss = []
4 mean_mAP = []
5
6 total_batches = len(train_loader)
7 display_interval = total_batches // 5 # Update after
8 20% of the total batches.
9
10 for batch_idx, (x, y) in enumerate(train_loader):
11 x, y = x.to(DEVICE), y.to(DEVICE)
12 out = model(x)
13 loss = loss_fn(out, y)
14
15 optimizer.zero_grad()
16 loss.backward()
17 optimizer.step()
18
19 pred_boxes, true_boxes = get_bboxes_training(out,
20 y, iou_threshold=0.5, threshold=0.4)
```

```
18 mAP = mean_average_precision(pred_boxes,
19 true_boxes, iou_threshold=0.5, box_format="midpoint")
20
21 mean_loss.append(loss.item())
22 mean_mAP.append(mAP.item())
23
24 if batch_idx % display_interval == 0 or batch_idx
25 == total_batches - 1:
26 print(f"Epoch: {epoch:3} \t Iter: {batch_idx
27 :3}/{total_batches:3} \t Loss: {loss.item():3.10f} \t
28 mAP: {mAP.item():3.10f}")
29
30 avg_loss = sum(mean_loss) / len(mean_loss)
31 avg_mAP = sum(mean_mAP) / len(mean_AP)
32 print(colored(f"Train \t loss: {avg_loss:3.10f} \t
33 mAP: {avg_mAP:3.10f}", 'green'))
34
35 return avg_mAP
36
37 def test_fn(test_loader, model, loss_fn, epoch):
38 model.eval()
39 mean_loss = []
40 mean_mAP = []
41
42 for batch_idx, (x, y) in enumerate(test_loader):
43 x, y = x.to(DEVICE), y.to(DEVICE)
44 out = model(x)
45 loss = loss_fn(out, y)
46
47 pred_boxes, true_boxes = get_bboxes_training(out,
48 y, iou_threshold=0.5, threshold=0.4)
49 mAP = mean_average_precision(pred_boxes,
50 true_boxes, iou_threshold=0.5, box_format="midpoint")
51
52 mean_loss.append(loss.item())
53 mean_mAP.append(mAP.item())
54
55 avg_loss = sum(mean_loss) / len(mean_loss)
56 avg_mAP = sum(mean_mAP) / len(mean_AP)
57 print(colored(f"Test \t loss: {avg_loss:3.10f} \t mAP
58 : {avg_mAP:3.10f}", 'yellow'))
59
60 model.train()
```

```
54 return avg_mAP
55
```

## Định Nghĩa Hàm Main để Huấn Luyện Mô Hình

Chúng ta sẽ định nghĩa hàm `train` để khởi tạo mô hình, thiết lập các lớp dữ liệu, và tiến hành huấn luyện mô hình YOLOv1 trên bộ dữ liệu VOC.

```
1 # Main function
2 def train():
3 # Initialize model, optimizer, loss
4 model = Yolov1(split_size=7, num_boxes=2, num_classes
5 =20).to(DEVICE)
6 optimizer = optim.Adam(model.parameters(), lr=
7 LEARNING_RATE)
8 loss_fn = YoloLoss()
9
10 # Load checkpoint if necessary
11 if LOAD_MODEL:
12 load_checkpoint(torch.load(LOAD_MODEL_FILE),
13 model, optimizer)
14
15 # Create the full dataset
16 train_dataset = CustomVOCDataset(
17 root='./data',
18 year='2012',
19 image_set='train',
20 download=True,
21)
22
23 train_dataset.init_config_yolo(class_mapping=
24 class_mapping, custom_transforms=get_train_transforms
25 ())
26
27 testval_dataset = CustomVOCDataset(
28 root='./data',
29 year='2012',
30 image_set='val',
31 download=True,
32)
```

```
29 testval_dataset.init_config_yolo(class_mapping=
30 class_mapping, custom_transforms=get_val_transforms())
31
32 # Split dataset into train, validation, and test sets
33 # using indices
34 dataset_size = len(testval_dataset)
35 val_size = int(0.15 * dataset_size)
36 test_size = dataset_size - val_size
37
38 val_indices = list(range(val_size))
39 test_indices = list(range(val_size, val_size +
40 test_size))
41
42 # Create SubsetRandomSamplers
43 val_sampler = SubsetRandomSampler(val_indices)
44 test_sampler = SubsetRandomSampler(test_indices)
45
46 # Create DataLoaders using the samplers
47 train_loader = DataLoader(
48 dataset=train_dataset,
49 batch_size=BATCH_SIZE,
50 num_workers=NUM_WORKERS,
51 pin_memory=PIN_MEMORY,
52 drop_last=True,
53)
54
55 val_loader = DataLoader(
56 dataset=testval_dataset,
57 batch_size=BATCH_SIZE,
58 num_workers=NUM_WORKERS,
59 pin_memory=PIN_MEMORY,
60 sampler=val_sampler, # Use the sampler here
61 drop_last=False,
62)
63
64 test_loader = DataLoader(
65 dataset=testval_dataset,
66 batch_size=BATCH_SIZE,
67 num_workers=NUM_WORKERS,
68 pin_memory=PIN_MEMORY,
69 sampler=test_sampler, # Use the sampler here
70 drop_last=False,
71)
```

```
70 best_mAP_train = 0
71 best_mAP_val = 0
72 best_mAP_test = 0
73
74 # Training loop
75 for epoch in range(EPOCHS):
76 train_mAP = train_fn(train_loader, model,
77 optimizer, loss_fn, epoch)
78 val_mAP = val_test_fn(val_loader, model, loss_fn,
79 epoch)
80 test_mAP = val_test_fn(test_loader, model,
81 loss_fn, epoch, is_test=True) # Pass is_test=True for
82 # test set
83
84 # Update best mAP values
85 if train_mAP > best_mAP_train:
86 best_mAP_train = train_mAP
87 if val_mAP > best_mAP_val:
88 best_mAP_val = val_mAP
89 # Save checkpoint when validation mAP
90 improves
91 checkpoint = {
92 "state_dict": model.state_dict(),
93 "optimizer": optimizer.state_dict(),
94 }
95 save_checkpoint(checkpoint, filename=
LOAD_MODEL_FILE)
96 if test_mAP > best_mAP_test:
97 best_mAP_test = test_mAP
98
99 print(colored(f"Best Train mAP: {best_mAP_train:.3f}",
100 'green'))
101 print(colored(f"Best Val mAP: {best_mAP_val:.3f}",
102 'blue'))
103 print(colored(f"Best Test mAP: {best_mAP_test:.3f}"',
104 'yellow'))
```

## Hàm test

Sau khi huấn luyện, chúng ta có thể sử dụng hàm `test` để thực hiện dự đoán trên các hình ảnh mới, hiển thị các bounding boxes và lớp đối tượng được

phát hiện.

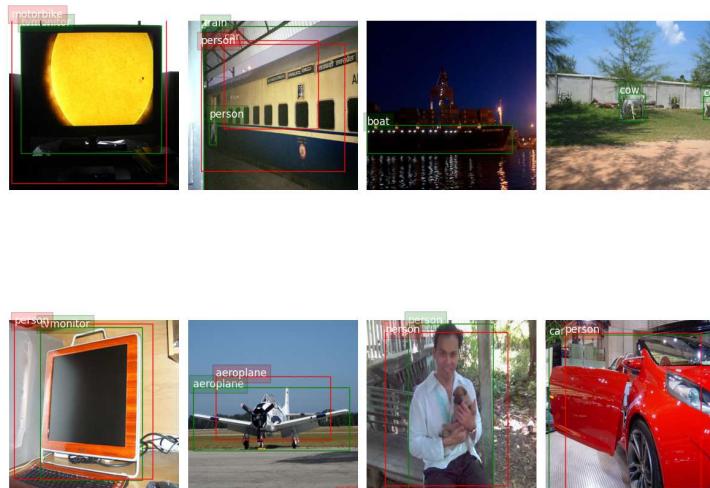
```
1 def plot_image_with_labels(image, ground_truth_boxes,
2 predicted_boxes, class_mapping):
3 """Draw both ground truth and predicted bounding
4 boxes on an image, with labels."""
5
6 # Inverting the class mapping for easy access of
7 # class names based on indices
8 inverted_class_mapping = {v: k for k, v in
9 class_mapping.items()}
10
11 # Convert the image to a numpy array and get its
12 # dimensions
13 im = np.array(image)
14 height, width, _ = im.shape
15
16 # Create a figure and axis for plotting
17 fig, ax = plt.subplots(1)
18 # Display the image
19 ax.imshow(im)
20
21 # Plot each ground truth box in green
22 for box in ground_truth_boxes:
23 # Extract label index and bounding box
24 # coordinates
25 label_index, box = box[0], box[2:]
26 # Calculate upper left coordinates
27 upper_left_x = box[0] - box[2] / 2
28 upper_left_y = box[1] - box[3] / 2
29 # Create a rectangle patch
30 rect = patches.Rectangle(
31 (upper_left_x * width, upper_left_y * height),
32 box[2] * width,
33 box[3] * height,
34 linewidth=1,
35 edgecolor="green",
36 facecolor="none",
37)
38 # Add the rectangle to the plot
39 ax.add_patch(rect)
```

```
34 # Retrieve the class name and add it as text to
35 # the plot
36 class_name = inverted_class_mapping.get(
37 label_index, "Unknown")
38 ax.text(upper_left_x * width, upper_left_y * height,
39 class_name, color='white', fontsize=12, bbox=
40 dict(facecolor='green', alpha=0.2))
41
42 # Plot each predicted box in red
43 for box in predicted_boxes:
44 # Similar processing as for ground truth boxes
45 label_index, box = box[0], box[2:]
46 upper_left_x = box[0] - box[2] / 2
47 upper_left_y = box[1] - box[3] / 2
48 rect = patches.Rectangle(
49 (upper_left_x * width, upper_left_y * height),
50 box[2] * width,
51 box[3] * height,
52 linewidth=1,
53 edgecolor="r",
54 facecolor="none",
55)
56 ax.add_patch(rect)
57 class_name = inverted_class_mapping.get(
58 label_index, "Unknown")
59 ax.text(upper_left_x * width, upper_left_y * height,
60 class_name, color='white', fontsize=12, bbox=
61 dict(facecolor='red', alpha=0.2))
62
63 plt.show()
64
65 def test():
66 # Create a YOLO model object with specific
67 # hyperparameters.
68 model = Yolov1(split_size=7, num_boxes=2, num_classes
69 =20).to(DEVICE)
70
71 # Load saved model weights and optimizer information
72 # from a file, if applicable.
73 if LOAD_MODEL:
74 model.load_state_dict(torch.load(LOAD_MODEL_FILE)
75 ['state_dict'])
```

```
66 # Prepare the test dataset and DataLoader for model
67 # evaluation
68 test_dataset = CustomVOCDataset(root='./data',
69 image_set='val', download=False)
70 test_dataset.init_config_yolo(class_mapping=
71 class_mapping, custom_transforms=get_valid_transforms()
72)
73 test_loader = DataLoader(dataset=test_dataset,
74 batch_size=BATCH_SIZE,
75 num_workers=NUM_WORKERS,
76 pin_memory=PIN_MEMORY, shuffle=False, drop_last=False)
77
78 model.eval()
79 # Iterate over the test dataset and process each
80 # batch
81 for x, y in test_loader:
82 x = x.to(DEVICE)
83 out = model(x)
84
85 # Convert model output to bounding boxes and
86 # apply non-max suppression
87 pred_bboxes = cellboxes_to_boxes(out)
88 gt_bboxes = cellboxes_to_boxes(y)
89
90 # Plot the first 8 images with their ground truth
91 # and predicted bounding boxes
92 for idx in range(8):
93 pred_box = non_max_suppression(pred_bboxes[
94 idx], iou_threshold=0.5, threshold=0.4, box_format="
95 midpoint")
96 gt_box = non_max_suppression(gt_bboxes[idx],
97 iou_threshold=0.5, threshold=0.4, box_format="midpoint"
98 ")
99
100 image = x[idx].permute(1,2,0).to("cpu")/255
101 plot_image_with_labels(image, gt_box,
102 pred_box, class_mapping)
103
104 break # Stop after processing the first batch
```

## Kết luận

Thông qua bài toán thứ tư, chúng ta đã hoàn thiện quá trình xây dựng và huấn luyện mô hình **YOLOv1** cho bài toán **Object Detection**. Mô hình này không chỉ phân loại các đối tượng trong hình ảnh mà còn định vị chính xác vị trí của chúng bằng các bounding boxes. Việc áp dụng các kỹ thuật như **Non-Maximum Suppression** và tính toán **mean Average Precision (mAP)** giúp cải thiện độ chính xác và hiệu quả của hệ thống.



Hình 41.5: Kết quả dự đoán của mô hình YOLOv1 sau khi train 10 epoch. Kết quả đúng có đường viền màu xanh, kết quả dự đoán có đường viền màu đỏ.

Mô hình YOLOv1 đã chứng minh được khả năng phát hiện đối tượng nhanh chóng và chính xác, mở đường cho các phiên bản tiếp theo như YOLOv2, YOLOv3, và YOLOv4 với nhiều cải tiến hơn nữa về kiến trúc và hiệu suất.

### 41.3 Câu hỏi trắc nghiệm

1. **Object Detection** nhằm mục đích gì?
  - (a) Phân loại hình ảnh.
  - (b) Nhận diện và định vị các object trong hình ảnh.
  - (c) Phân tích dữ liệu.
  - (d) Xử lý ngôn ngữ tự nhiên.
2. Một trong những mô hình phổ biến nhất cho **Object Detection** là:
  - (a) YOLO (You Only Look Once).
  - (b) ResNet50.
  - (c) BERT.
  - (d) GPT-4.
3. Trong **Object Detection**, **bounding box** được sử dụng để:
  - (a) Phân loại loại object.
  - (b) Tăng độ phân giải của hình ảnh.
  - (c) Xử lý màu sắc của hình ảnh.
  - (d) Định vị vị trí của object trong hình ảnh.
4. **Intersection over Union (IoU)** được sử dụng để:
  - (a) Đánh giá hiệu suất của mô hình phân loại.
  - (b) Đo lường độ chính xác của **bounding box**.
  - (c) Tăng tốc độ huấn luyện.
  - (d) Giảm thiểu độ lệch dữ liệu.
5. Một thách thức chính trong **Object Detection** là:
  - (a) Xử lý ngôn ngữ tự nhiên.
  - (b) Phân loại văn bản.
  - (c) Nhận diện các object trong điều kiện ánh sáng kém.

- (d) Tăng độ phân giải của hình ảnh.

6. **Non-Maximum Suppression (NMS)** được sử dụng để:

- (a) Tăng độ chính xác của phân loại.
- (b) Loại bỏ các **bounding box** trùng lặp.
- (c) Tối ưu hóa hàm loss.
- (d) Điều chỉnh tỷ lệ học.

7. **Ý tưởng chính của YOLO V1 là gì?**

- (a) Sử dụng một mô hình CNN để dự đoán box và score của các objects trong ảnh.
- (b) Sử dụng nhiều mô hình CNN để dự đoán box và score của các objects trong ảnh.
- (c) Sử dụng mô hình RNN để dự đoán box và score của các objects trong ảnh.
- (d) Sử dụng một mô hình MLP để dự đoán box và score của các objects trong ảnh.

8. **YOLO V1 chia hình ảnh đầu vào như thế nào?**

- (a) Chia thành một lưới các vùng chồng chéo (Grid of overlapping regions).
- (b) Chia thành một lưới các vùng không chồng chéo.
- (c) Chia thành một vùng đơn.
- (d) Chia thành nhiều vùng dựa trên số lượng objects trong ảnh.

9. **Nhược điểm của YOLO V1 là gì?**

- (a) Không thể phát hiện được các objects nhỏ.
- (b) Không có khả năng xử lý đa objects trong cùng một vùng.
- (c) Không có khả năng xử lý các objects có kích thước khác nhau.
- (d) Tất cả các ý trên.

10. **Sự khác nhau giữa tăng cường dữ liệu và tiền xử lý dữ liệu là gì?**

- (a) Tăng cường dữ liệu dùng để tăng kích thước tập dữ liệu trong khi tiền xử lý dữ liệu dùng để làm sạch dữ liệu.
- (b) Tăng cường dữ liệu dùng để cải thiện chất lượng model trong khi tiền xử lý dữ liệu dùng để giảm overfitting.
- (c) Tăng cường dữ liệu dùng để tăng kích thước tập dữ liệu trong khi tiền xử lý dữ liệu dùng để chuẩn bị dữ liệu cho giai đoạn training.
- (d) Tăng cường dữ liệu dùng để giảm overfitting trong khi tiền xử lý dữ liệu dùng để tăng chất lượng model.

## 41.4 Phụ lục

1. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần bài tập, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
2. **Rubric:**

| Câu   | Kiến Thức                                                                                                                                                                                                                                                                                  | Đánh Giá                                                                                                                                                                                                                                                                                   |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| II.2. | <ul style="list-style-type: none"> <li>- Kiến thức về YOLOv1.</li> <li>- Kiến thức về cách huấn luyện YOLOv1 cho bài toán Object Detection.</li> </ul>                                                                                                                                     | <ul style="list-style-type: none"> <li>- Biết cách ứng dụng YOLOv1 để xây dựng mô hình Object Detection.</li> </ul>                                                                                                                                                                        |
| II.3. | <ul style="list-style-type: none"> <li>- Kiến thức về Convolutional Neural Networks (CNN).</li> <li>- Kiến thức về Classification và Localization trong Object Detection.</li> <li>- Khả năng xây dựng mô hình CNN sử dụng PyTorch cho bài toán Classification và Localization.</li> </ul> | <ul style="list-style-type: none"> <li>- Hiểu được các khái niệm liên quan đến mô hình CNN và các nhiệm vụ Classification và Localization.</li> <li>- Khả năng lập trình và ứng dụng mô hình CNN vào giải quyết bài toán Classification và Localization trong Object Detection.</li> </ul> |

- *Hết* -

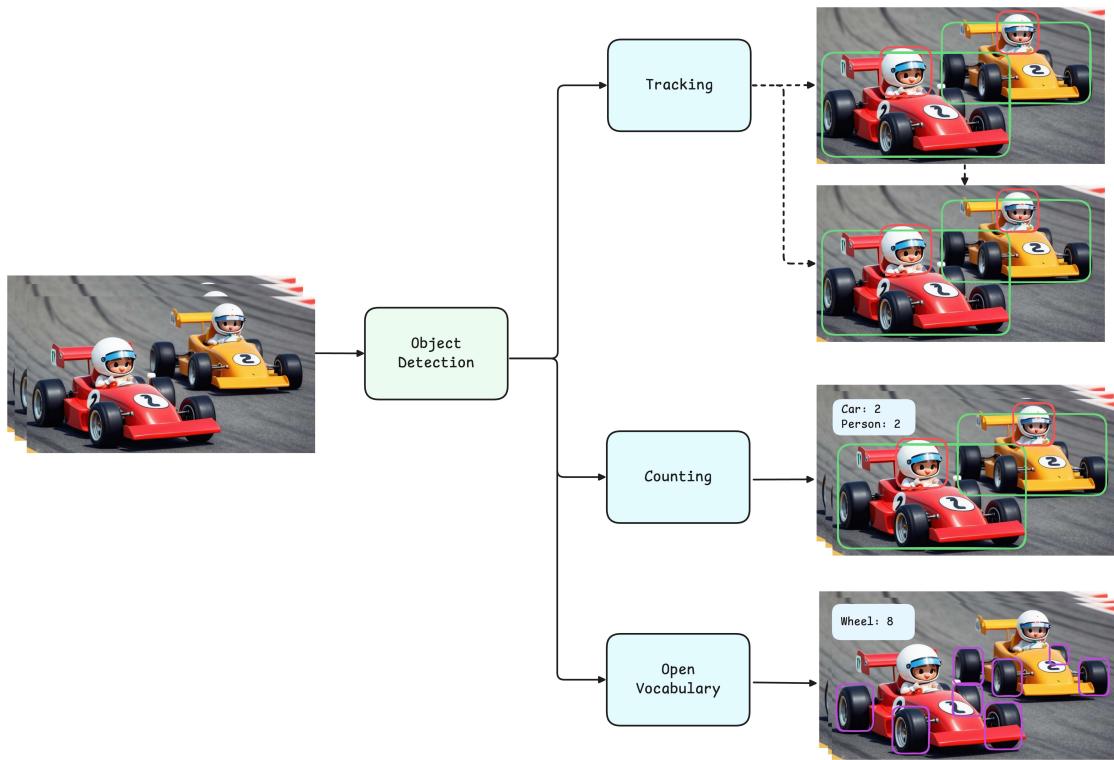
## Chương 42

# Object Detection 2: Thảo luận YOLO v4 đến v10

### 42.1 Giới thiệu

Trong buổi hôm nay, chúng ta sẽ cùng nhau tìm hiểu 3 bài toán phổ biến ứng dụng Object Detection: Object Tracking, Object Counting, và Open Vocab Detection.

- **Tracking:** Theo dõi vị trí các đối tượng qua các khung hình liên tiếp.
- **Counting:** Đếm số lượng các đối tượng trong ảnh.
- **Open Vocabulary:** Nhận diện các đối tượng không nằm trong danh sách label cố định.



Hình 42.1: Hình ảnh minh họa cho 3 bài toán Tracking, Counting và Open Vocab Detection.

## 42.2 Nội dung chính

### 42.2.1 Bài toán 1: Object Tracking

#### Version đơn giản

Trong bài toán này, chúng ta sẽ triển khai một hệ thống **Object Tracking** đơn giản sử dụng mô hình YOLO để theo dõi các đối tượng trong video. Dưới đây là các bước thực hiện:

**Import các thư viện cần thiết** Đầu tiên, chúng ta cần import các thư viện để xử lý video và thực hiện việc theo dõi đối tượng.

```
1 from collections import defaultdict
2 import cv2
3 import numpy as np
4 from ultralytics import YOLO
5
```

**Tải mô hình YOLO và mở video** Chúng ta sẽ tải mô hình YOLO và mở file video cần xử lý.

```
1 # Load the YOLO11 model
2 model = YOLO("yolo11.pt")
3
4 # Open the video file
5 video_path = "samples/vietnam.mp4"
6 cap = cv2.VideoCapture(video_path)
7
```

**Thiết lập VideoWriter để lưu video kết quả** Tiếp theo, chúng ta thiết lập đối tượng VideoWriter để lưu video đã được xử lý với các đối tượng được theo dõi.

```
1 # Get video properties
2 width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
3 height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
4 fps = int(cap.get(cv2.CAP_PROP_FPS))
5
6 # Create VideoWriter object
7 video_name = video_path.split("/")[-1]
8 output_path = f"run/{video_name.split('.')[0]}_tracked.
 mp4"
9 fourcc = cv2.VideoWriter_fourcc(*"mp4v")
10 out = cv2.VideoWriter(output_path, fourcc, fps, (width,
 height))
```

**Khởi tạo lịch sử theo dõi và vòng lặp xử lý frames** Chúng ta sẽ sử dụng một defaultdict để lưu trữ lịch sử theo dõi các đối tượng và thực hiện vòng lặp qua từng frame của video để thực hiện việc theo dõi.

```
1 # Store the track history
2 track_history = defaultdict(lambda: [])
3
4 # Loop through the video frames
5 while cap.isOpened():
6 # Read a frame from the video
7 success, frame = cap.read()
8
9 if success:
10 # Run YOLOv1 tracking on the frame, persisting
11 # tracks between frames
12 results = model.track(frame, persist=True, show=False)
13
14 # Get the boxes and track IDs (with error
15 # handling)
16 boxes = results[0].boxes.xywh.cpu()
17 try:
18 track_ids = results[0].boxes.id
19 if track_ids is not None:
20 track_ids = track_ids.int().cpu().tolist()
21 ()
```

```
19 else:
20 track_ids = [] # No tracks found in this
frame
21 except AttributeError:
22 track_ids = [] # Handle case where tracking
fails
23
24 # Visualize the results on the frame
25 annotated_frame = results[0].plot()
26
27 # Plot the tracks only if we have valid tracking
data
28 if track_ids:
29 for box, track_id in zip(boxes, track_ids):
30 x, y, w, h = box
31 track = track_history[track_id]
32 track.append((float(x), float(y))) # x,
y center point
33
34 if len(track) > 120: # retain 30 tracks
for 30 frames
35 track.pop(0)
36
37 # Draw the tracking lines
38 points = np.hstack(track).astype(np.int32
).reshape((-1, 1, 2))
39 cv2.polylines(
40 annotated_frame,
41 [points],
42 isClosed=False,
43 color=(230, 230, 230),
44 thickness=4,
45)
46 # Write the frame to output video
47 out.write(annotated_frame)
48 else:
49 # Break the loop if the end of the video is
reached
50 break
51
52 # Release everything
53 cap.release()
54 out.release()
55 print(f"Video has been saved to {output_path}")
```

56

**Kết quả sau khi xử lý** Sau khi chạy chương trình, video kết quả sẽ được lưu tại đường dẫn đã chỉ định.

### Version optimized

Phiên bản này được tối ưu hóa bằng cách sử dụng batching và cải thiện hiệu suất xử lý. Các bước thực hiện như sau:

**Import các thư viện cần thiết** Chúng ta sẽ import thêm các thư viện hỗ trợ như argparse, tqdm và thay đổi cách import logger.

```
1 import argparse
2 from collections import defaultdict
3 import cv2
4 import numpy as np
5 from tqdm import tqdm
6 from ultralytics import YOLO
7 from loguru import logger
8
```

**Định nghĩa cấu hình và khởi tạo video** Chúng ta sẽ định nghĩa một hàm để tải cấu hình và một hàm để khởi tạo các đối tượng VideoCapture và VideoWriter.

```
1 def load_config():
2 """Load and return configuration settings"""
3 return {
4 "model_path": "yolo11x.pt",
5 "track_history_length": 120,
6 "batch_size": 64,
7 "line_thickness": 4,
8 "track_color": (230, 230, 230),
9 }
```

```
10
11 def initialize_video(video_path):
12 """Initialize video capture and writer objects"""
13 cap = cv2.VideoCapture(video_path)
14 width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
15 height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
16 fps = int(cap.get(cv2.CAP_PROP_FPS))
17
18 video_name = video_path.split("/")[-1]
19 output_path = f"run/{video_name.split('.')[0]}_{video_name}.mp4"
20 fourcc = cv2.VideoWriter_fourcc(*"mp4v")
21 out = cv2.VideoWriter(output_path, fourcc, fps, (
22 width, height))
23
24 return cap, out, output_path
25
```

**Cập nhật lịch sử theo dõi** Tiếp theo, chúng ta sẽ cập nhật lịch sử theo dõi và loại bỏ các track cũ.

```
1 def update_track_history(
2 track_history,
3 last_seen,
4 track_ids,
5 frame_count,
6 batch_size,
7 frame_idx,
8 history_length,
9) :
10 """Update tracking history and remove old tracks"""
11 current_tracks = set(track_ids)
12 for track_id in list(track_history.keys()):
13 if track_id in current_tracks:
14 last_seen[track_id] = frame_count - (
15 batch_size - frame_idx - 1)
16 elif frame_count - last_seen[track_id] >
17 history_length:
18 del track_history[track_id]
19 del last_seen[track_id]
```

**Vẽ các đường theo dõi trên frame** Chúng ta sẽ thêm các đường theo dõi các đối tượng lên từng frame video.

```

1 def draw_tracks(frame, boxes, track_ids, track_history,
2 config):
3 """Draw tracking lines on frame"""
4 if not track_ids:
5 return frame
6
7 for box, track_id in zip(boxes, track_ids):
8 x, y, w, h = box
9 track = track_history[track_id]
10 track.append((float(x), float(y)))
11 if len(track) > config["track_history_length"]:
12 track.pop(0)
13
14 points = np.hstack(track).astype(np.int32).
15 reshape((-1, 1, 2))
16 cv2.polylines(
17 frame,
18 [points],
19 isClosed=False,
20 color=config["track_color"],
21 thickness=config["line_thickness"],
22)
23 return frame
24

```

**Xử lý một batch các frames** Để tối ưu hóa hiệu suất, chúng ta sẽ xử lý nhiều frames cùng một lúc.

```

1 def process_batch(model, batch_frames, track_history,
2 last_seen, frame_count, config):
3 """Process a batch of frames through YOLO model"""
4 results = model.track(
5 batch_frames,
6 persist=True,
7 tracker="botsort.yaml",
8 show=False,
9 verbose=False,
10

```

```

9 iou=0.5 ,
10)
11
12 processed_frames = []
13 for frame_idx, result in enumerate(results):
14 boxes = result.boxes.xywh.cpu()
15 track_ids = (
16 result.boxes.id.int().cpu().tolist() if
17 result.boxes.id is not None else []
18)
19
20 update_track_history(
21 track_history,
22 last_seen,
23 track_ids,
24 frame_count,
25 len(batch_frames),
26 frame_idx,
27 config["track_history_length"],
28)
29
30 annotated_frame = result.plot(font_size=4,
31 line_width=2)
32 annotated_frame = draw_tracks(
33 annotated_frame, boxes, track_ids,
34 track_history, config
35)
36 processed_frames.append(annotated_frame)

 return processed_frames

```

**Hàm chính để xử lý video** Cuối cùng, chúng ta sẽ kết hợp tất cả các bước trên vào một hàm chính để xử lý toàn bộ video.

```

1 def main(video_path):
2 """Main function to process video"""
3 CONFIG = load_config()
4 model = YOLO(CONFIG.get("model_path", "yolo11x.pt"))
5
6 cap, out, output_path = initialize_video(video_path)

```

```
7 track_history = defaultdict(lambda: [])
8 last_seen = defaultdict(int)
9 total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
10
11 with tqdm(
12 total=total_frames,
13 desc="Processing frames",
14 colour="green",
15) as pbar:
16 frame_count = 0
17 batch_frames = []
18
19 while cap.isOpened():
20 success, frame = cap.read()
21 if not success:
22 break
23
24 frame_count += 1
25 batch_frames.append(frame)
26
27 if len(batch_frames) == CONFIG["batch_size"]
28 or frame_count == total_frames:
29 try:
30 processed_frames = process_batch(
31 model,
32 batch_frames,
33 track_history,
34 last_seen,
35 frame_count,
36 CONFIG,
37)
38 for frame in processed_frames:
39 out.write(frame)
40 pbar.update(1)
41 batch_frames = []
42
43 except Exception as e:
44 logger.error(
45 f"Error when handling frames {frame_count - len(batch_frames) + 1} to {frame_count}: {str(e)}"
46)
47 batch_frames = []
48 continue
```

```

48
49 try:
50 cap.release()
51 out.release()
52 cv2.destroyAllWindows()
53 logger.info(f"{output_path}")
54 except Exception as e:
55 logger.error(f"{str(e)}")
56
57 if __name__ == "__main__":
58 parser = argparse.ArgumentParser()
59 parser.add_argument("--video-path", type=str, default =
60 "samples/vietnam-2.mp4")
61 args = parser.parse_args()
62
63 main(args.video_path)
64

```

### Giải thích các thay đổi và tối ưu hóa

- **Sử dụng batching:** Thay vì xử lý từng frame một, chúng ta xử lý nhiều frames cùng lúc để tăng hiệu suất.
- **Cấu hình linh hoạt:** Định nghĩa các tham số cấu hình trong hàm `load_config` để dễ dàng điều chỉnh.
- **Xử lý lỗi:** Sử dụng `try-except` để xử lý các lỗi có thể xảy ra trong quá trình xử lý frames.
- **Thay thế logger:** Sử dụng `loguru` thay cho `logger` từ `src.utils` để đơn giản hóa việc logging.

**Chạy chương trình** Để chạy chương trình, bạn có thể sử dụng lệnh sau trong terminal:

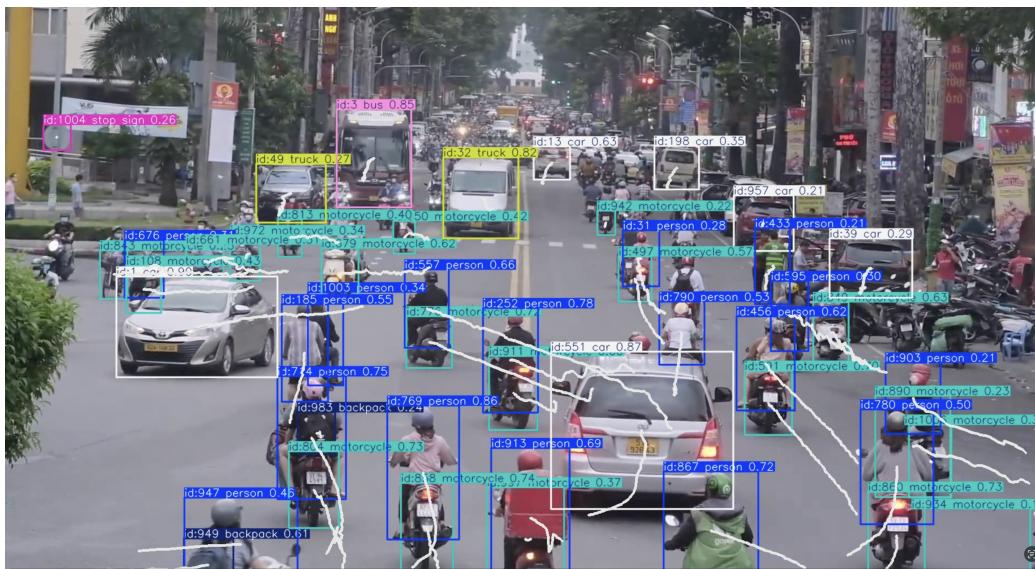
```

1 python your_script.py --video-path "samples/vietnam-2.mp4"
2

```

**Kết quả sau khi xử lý** Sau khi chạy phiên bản tối ưu hóa, video kết quả sẽ được lưu tại đường dẫn đã chỉ định với hiệu suất xử lý cao hơn và các đối tượng được theo dõi mượt mà hơn.

Hình dưới đây minh họa output:



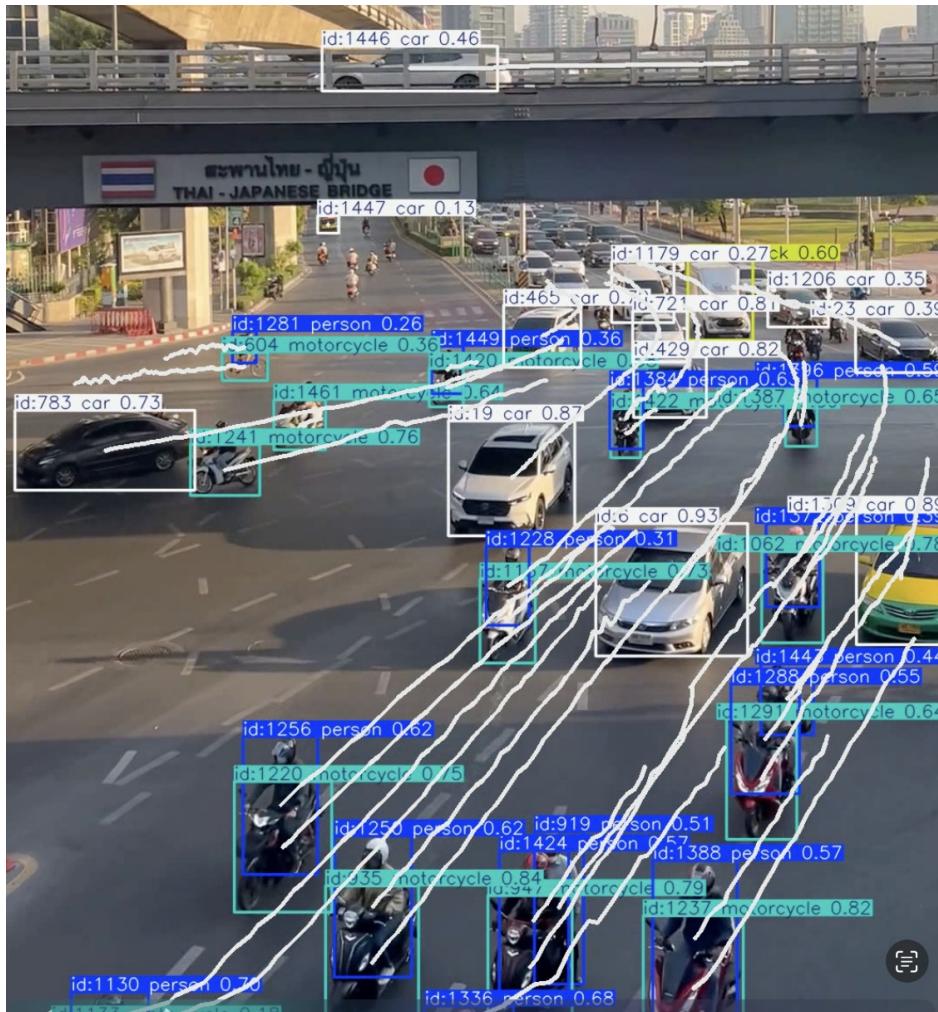
Hình 42.2: Output bài toán tracking (1).

#### 42.2.2 Bài toán 2: Object Counting

Trong bài toán này, chúng ta sẽ triển khai một hệ thống **Object Counting** để đếm số lượng đối tượng xuất hiện trong một khu vực cụ thể của video. Sử dụng mô hình YOLO, chúng ta sẽ xác định và đếm các đối tượng trong vùng được định nghĩa trước. Dưới đây là các bước thực hiện:

### Import các thư viện cần thiết

Đầu tiên, chúng ta cần import các thư viện cần thiết để xử lý video và thực hiện việc đếm đối tượng.



Hình 42.3: Output bài toán tracking (2).

```
1 import cv2
2 from ultralytics import solutions
3
```

## Khởi tạo video và định nghĩa vùng đếm

Chúng ta sẽ mở file video cần xử lý và định nghĩa các điểm để tạo thành vùng đếm đối tượng.

```
1 cap = cv2.VideoCapture("samples/highway.mp4")
2 assert cap.isOpened(), "Error reading video file"
3 w, h, fps = (
4 int(cap.get(x))
5 for x in (cv2.CAP_PROP_FRAME_WIDTH, cv2.
6 CAP_PROP_FRAME_HEIGHT, cv2.CAP_PROP_FPS)
7)
8 # Define region points
9 # region_points = [(20, 400), (1080, 400)] # For line
10 region_points = [
11 (430, 700),
12 (1600, 700),
13 (1600, 1080),
14 (430, 1080),
15] # For rectangle region counting: top left, top right,
16 bottom right, bottom left
```

### Giải thích:

- `cv2.VideoCapture`: Mở file video để xử lý.
- `region_points`: Định nghĩa các điểm để tạo thành vùng đếm đối tượng. Trong trường hợp này, chúng ta sử dụng một hình chữ nhật.

## Thiết lập VideoWriter để lưu video kết quả

Chúng ta cần thiết lập đối tượng `VideoWriter` để lưu video đã được xử lý với các đối tượng được đếm.

```
1 # Video writer
2 video_writer = cv2.VideoWriter(
```

```

3 "./run/highway_counted.mp4", cv2.VideoWriter_fourcc(*
4 "mp4v"), fps, (w, h)
5

```

## Khởi tạo ObjectCounter

Chúng ta sẽ sử dụng `ObjectCounter` từ thư viện `ultralytics.solutions` để đếm các đối tượng trong vùng đã định nghĩa.

```

1 # Init ObjectCounter
2 counter = solutions.ObjectCounter(
3 show=False, # Display the output
4 region=region_points, # Pass region points
5 model="yolo11x.pt", # model="yolo11n-obb.pt" for
6 # object counting using YOLO11 OBB model.
7

```

## Giải thích:

- `show=False`: Không hiển thị cửa sổ kết quả.
- `region`: Các điểm định nghĩa vùng đếm đối tượng.
- `model`: Đường dẫn tới mô hình YOLO được sử dụng để phát hiện đối tượng.

## Xử lý video và đếm đối tượng

Chúng ta sẽ đọc từng frame của video, áp dụng `ObjectCounter` để đếm các đối tượng và ghi lại kết quả vào video đầu ra.

```

1 # Process video
2 while cap.isOpened():
3 success, im0 = cap.read()
4 if not success:

```

```
5 print(
6 "Video frame is empty or video processing has
7 been successfully completed."
8)
9 break
10 im0 = counter.count(im0)
11 video_writer.write(im0)
12
13 cap.release()
14 video_writer.release()
15 cv2.destroyAllWindows()
```

### Giải thích:

- Vòng lặp while: Đọc từng frame của video cho đến khi hết.
- counter.count(im0): Áp dụng ObjectCounter để đếm đối tượng trong frame hiện tại.
- video\_writer.write(im0): Ghi frame đã được đếm đối tượng vào video đầu ra.

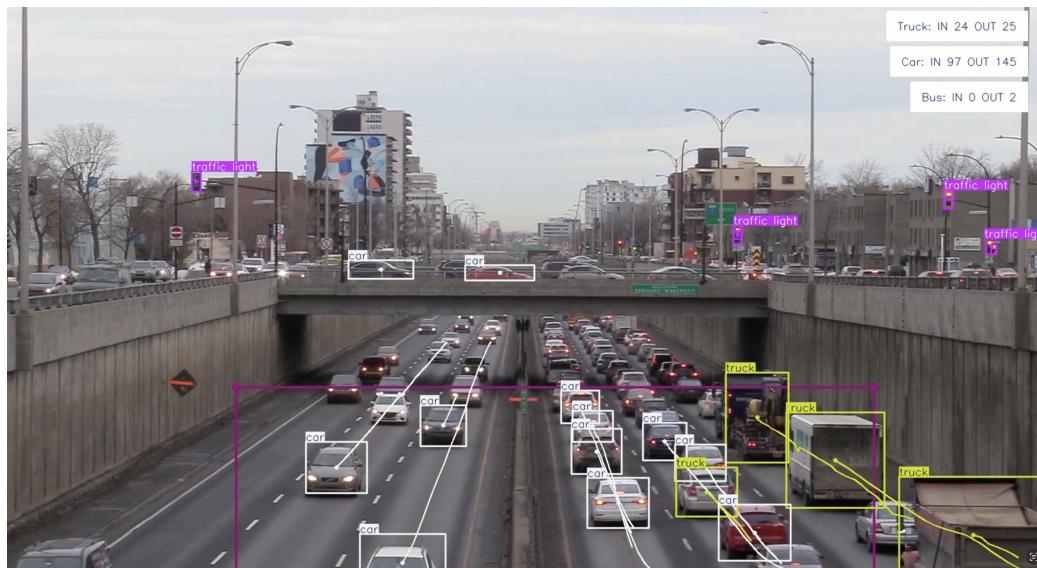
**Chạy chương trình** Để chạy chương trình, bạn có thể sử dụng lệnh sau trong terminal:

```
1 python your_counting_script.py
2
```

### Kết quả sau khi xử lý

Sau khi chạy chương trình, video kết quả sẽ được lưu tại đường dẫn đã chỉ định với các đối tượng được đếm và đánh dấu trong vùng đã định nghĩa.

Hình dưới đây minh họa kết quả của mô hình sau khi đếm các đối tượng trong video.



Hình 42.4: Kết quả của mô hình Object Counting sau khi xử lý video.

### 42.2.3 Bài toán 3: Open Vocab Detection

Trong bài toán này, chúng ta sẽ triển khai một hệ thống **Open Vocab Detection** để phát hiện các đối tượng trong hình ảnh dựa trên các lớp tùy chỉnh. Sử dụng mô hình YOLO-World, chúng ta có thể dễ dàng định nghĩa và phát hiện các lớp đối tượng theo nhu cầu cụ thể. Dưới đây là các bước thực hiện:

#### Import các thư viện cần thiết

Đầu tiên, chúng ta cần import các thư viện cần thiết để thực hiện việc phát hiện đối tượng.

```

1 from ultralytics import YOLOWorld
2 from ultralytics.engine.results import Boxes
3
4 from src.utils import save_detection_results
5

```

## Khởi tạo mô hình YOLO-World

Chúng ta sẽ khởi tạo mô hình YOLO-World với trọng số được huấn luyện trước.

```
1 # Initialize a YOLO-World model
2 model = YOLOWorld("yolov8s-world.pt")
```

### Giải thích:

- **YOLOWorld**: Một phiên bản mở rộng của mô hình YOLO, hỗ trợ phát hiện đối tượng với các lớp tùy chỉnh.
- **"yolov8s-world.pt"**: Đường dẫn tới file trọng số của mô hình đã được huấn luyện trước.

## Định nghĩa các lớp tùy chỉnh

Chúng ta có thể định nghĩa các lớp đối tượng mà chúng ta muốn mô hình phát hiện. Trong ví dụ này, chúng ta sẽ chỉ định lớp **bus**.

```
1 # Define custom classes
2 model.set_classes(["bus"]) # ----- Change this to
 the class you want to detect
3
```

### Giải thích:

- **set\_classes**: Hàm này cho phép chúng ta định nghĩa các lớp đối tượng mà mô hình sẽ phát hiện. Bạn có thể thay đổi danh sách này để phù hợp với nhu cầu của mình.

## Thực hiện dự đoán trên một hình ảnh

Chúng ta sẽ sử dụng mô hình đã được cấu hình để thực hiện dự đoán trên một hình ảnh cụ thể.

```
1 # Execute prediction on an image
2 results: Boxes = model.predict("samples/bus.jpg")
3
```

### Giải thích:

- `model.predict`: Hàm này thực hiện việc dự đoán các đối tượng trong hình ảnh đầu vào.
- `"samples/bus.jpg"`: Đường dẫn tới hình ảnh mà chúng ta muốn phát hiện đối tượng.
- `results`: Kết quả trả về bao gồm các bounding boxes và các thông tin liên quan đến các đối tượng được phát hiện.

### Lưu kết quả phát hiện đối tượng

Chúng ta sẽ lưu kết quả phát hiện đối tượng dưới dạng hình ảnh với các bounding boxes được vẽ lên.

```
1 # Save detection results as images
2 save_detection_results(results)
3
```

### Giải thích:

- `save_detection_results`: Hàm này chịu trách nhiệm lưu kết quả phát hiện đối tượng vào các file hình ảnh. Hàm này có thể được định nghĩa trong module `src.utils`.

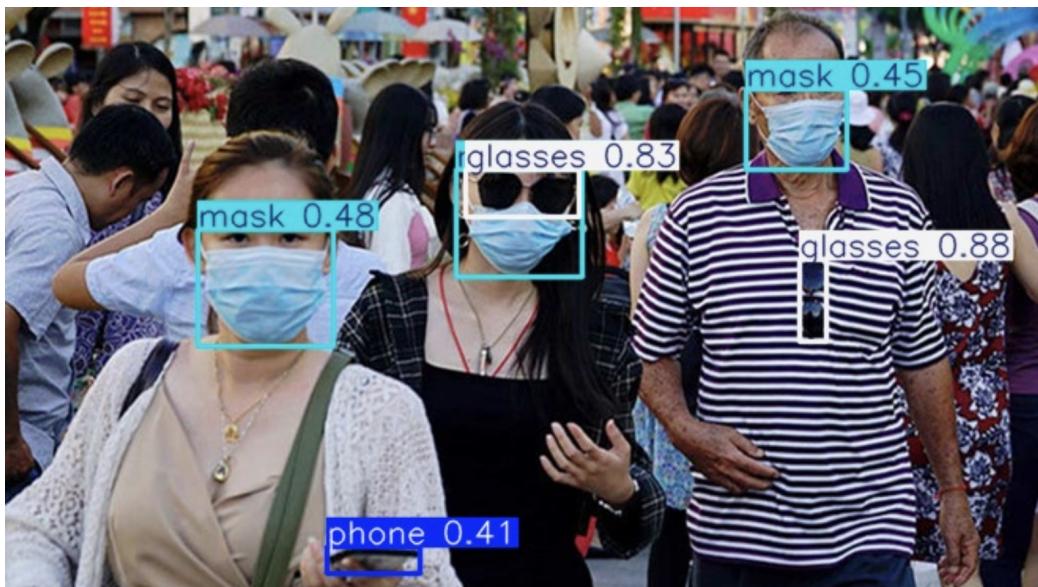
**Chạy chương trình** Để chạy chương trình, bạn có thể sử dụng lệnh sau trong terminal:

```
1 python your_open_vocab_detection_script.py
2
```

### Kết quả sau khi phát hiện

Sau khi chạy chương trình, hình ảnh kết quả sẽ được lưu với các bounding boxes được vẽ lên các đối tượng được phát hiện.

Hình dưới đây minh họa kết quả của mô hình sau khi phát hiện đối tượng **bus** trong hình ảnh.



Hình 42.5: Kết quả của mô hình Open Vocab Detection sau khi phát hiện đối tượng **phone**, **glasses**, **mask** trong hình ảnh.

### Giải thích thêm:

- Đảm bảo rằng mô hình YOLO-World đã được tải xuống và đường dẫn `model` trong `YOLOWorld` là chính xác.
- Các lớp đối tượng có thể được thay đổi bằng cách chỉnh sửa danh sách trong `set_classes`.
- Kết quả phát hiện đối tượng sẽ được lưu trong thư mục được chỉ định bởi hàm `save_detection_results`.

### 42.3 Câu hỏi trắc nghiệm

1. Trong quá trình **Object Tracking** (Bài toán 1), thông tin `track_id` được sử dụng với mục đích gì?
  - (a) Hiển thị màu sắc của bounding box.
  - (b) Tự động thay đổi kích thước khung hình.
  - (c) Phân biệt các đối tượng khác nhau khi di chuyển qua nhiều khung hình.
  - (d) Tăng tốc độ xử lý video.
2. **Bài toán 2 (Object Counting)** thường được ứng dụng để làm gì trong thực tế?
  - (a) Làm nổi bật các vật thể có kích thước lớn trong ảnh.
  - (b) Phân loại toàn bộ hình ảnh.
  - (c) Đếm số lượng các đối tượng trong một vùng hoặc khung hình.
  - (d) Tăng độ phân giải của ảnh.
3. Trong **Bài toán 1 (Object Tracking)**, hàm `draw_tracks` trong mã nguồn có tác dụng gì?
  - (a) Vẽ thêm chữ watermark lên khung hình.
  - (b) Tô màu toàn bộ đối tượng được phát hiện.
  - (c) Vẽ đường nối theo quỹ đạo di chuyển của đối tượng qua các khung hình.
  - (d) Giảm nhiễu cho video.
4. Khi sử dụng **batching** (Bài toán 1 phiên bản *optimized*) trong **Object Tracking**, lợi ích chính là gì?
  - (a) Giảm chất lượng đầu ra.
  - (b) Tăng tốc độ và hiệu suất xử lý.
  - (c) Tạo ra nhiều bounding box bị trùng lặp.
  - (d) Làm méo khung hình video.

5. Trong **Bài toán 2 (Object Counting)**, biến `region_points` được sử dụng để:
  - (a) Xác định khu vực hoặc đường kẻ dùng để đếm đối tượng.
  - (b) Xóa nền của hình ảnh.
  - (c) Thay đổi kích thước của từng đối tượng.
  - (d) Gán nhãn cho các đối tượng hiếm.
6. **Bài toán 3 (Open Vocab Detection)** cho phép chúng ta:
  - (a) Chỉ nhận diện các lớp đã được huấn luyện cố định.
  - (b) Phát hiện các đối tượng không có trong danh sách nhãn cố định.
  - (c) Giảm số lượng nhãn cần huấn luyện.
  - (d) Bỏ qua các bounding box có kích thước nhỏ hơn ngưỡng.
7. Trong **Bài toán 2 (Object Counting)**, thư viện `ultralytics.solutions` được sử dụng để:
  - (a) Hiển thị tất cả các mô hình YOLO có sẵn.
  - (b) Giảm độ phân giải của video xuống mức tối thiểu.
  - (c) Huấn luyện mô hình từ đầu (scratch).
  - (d) Triển khai `ObjectCounter` giúp đếm số lượng đối tượng trong một vùng.
8. **Đường dẫn** đến video đầu vào và video kết quả (xuất ra) trong các **bài toán xử lý video** (như Bài toán 1 và Bài toán 2) được thiết lập chủ yếu thông qua:
  - (a) `cv2.VideoCapture` và `cv2.VideoWriter`.
  - (b) `numpy.load` và `numpy.save`.
  - (c) `matplotlib.imread` và `matplotlib.imsave`.
  - (d) `cv2.imshow` và `cv2.imwrite`.
9. Trong **hàm process\_batch** (Bài toán 1 phiên bản *optimized*), biến `results` được trả về dùng để:
  - (a) Lưu video đã chỉnh sửa.

- (b) Ghi lại lịch sử các đường tracking.
  - (c) Hiển thị trực tiếp video ra màn hình.
  - (d) Chứa thông tin về boxes và track\_ids sau khi chạy mô hình.
10. Một yếu tố quan trọng giúp nâng cao **hiệu suất tính toán** trong **Bài toán 1 (Object Tracking)**, phiên bản **optimized** là:
- (a) Dừng theo dõi đối tượng ngay sau khi phát hiện lần đầu.
  - (b) Sử dụng cùng lúc nhiều mô hình YOLO khác nhau.
  - (c) Áp dụng **batching** để xử lý nhiều khung hình trong một lần suy luận.
  - (d) Tăng độ phân giải của video lên gấp đôi.

## 42.4 Phụ lục

1. **Code và data:** Code và data sample các bạn có thể tìm ở github tại [đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần bài tập, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

| Bài toán | Kiến Thức                                                                                                                                                                                                                                                                                                                                                                                  | Đánh Giá                                                                                                                                                                                                                                                                                                                |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1        | <p><b>Object Tracking:</b></p> <ul style="list-style-type: none"> <li>- Kiến thức về cách sử dụng mô hình YOLO (phiên bản track) để phát hiện và theo dõi đối tượng trong video.</li> <li>- Hiểu cách thức lưu trữ lịch sử đối tượng, <b>track_id</b>, và vẽ đường di chuyển (<b>polylines</b>).</li> <li>- Biết áp dụng kỹ thuật <b>batching</b> để tối ưu tốc độ xử lý video.</li> </ul> | <ul style="list-style-type: none"> <li>- Nắm vững quá trình xử lý từng khung hình, gắn <b>track_id</b> cho đối tượng.</li> <li>- Triển khai thành công vòng lặp đọc video, gán ID và vẽ quỹ đạo di chuyển.</li> <li>- Sử dụng hiệu quả batching để tăng hiệu suất.</li> </ul>                                           |
| 2        | <p><b>Object Counting:</b></p> <ul style="list-style-type: none"> <li>- Kiến thức về xác định khu vực đếm (định nghĩa <b>region_points</b>) và cách tích hợp với mô hình phát hiện đối tượng (YOLO, <b>ultralytics.solutions</b>).</li> <li>- Sử dụng <b>ObjectCounter</b> để đếm số lượng đối tượng trong một vùng/video.</li> </ul>                                                      | <ul style="list-style-type: none"> <li>- Thiết lập được <b>region_points</b> hoặc đường line để xác định khu vực đếm.</li> <li>- Biết cách đọc/ghi video và ghi nhận số lượng đối tượng đếm được theo thời gian.</li> <li>- Áp dụng mô hình YOLO kết hợp <b>ObjectCounter</b> để hiển thị kết quả chính xác.</li> </ul> |
| 3        | <p><b>Open Vocab Detection:</b></p> <ul style="list-style-type: none"> <li>- Kiến thức về mô hình <b>YOLOWorld</b> và cách <b>set_classes</b> để phát hiện những lớp chưa có trong danh sách nhãn cố định.</li> <li>- Hiểu cơ chế nhận diện nhiều đối tượng mới mà không cần huấn luyện lại hoàn toàn. 1388</li> </ul>                                                                     | <ul style="list-style-type: none"> <li>- Sử dụng được <b>YOLOWorld</b> với các lớp tùy chỉnh.</li> <li>- Thiết lập mô hình dự đoán, phân tích kết quả phát hiện, và lưu kết quả (bounding boxes) thành file.</li> <li>- Đảm bảo tính mở rộng cho các lớp đối tượng mới.</li> </ul>                                      |

## **Phần X**

# **Module 10: Deep learning cho dữ liệu text**

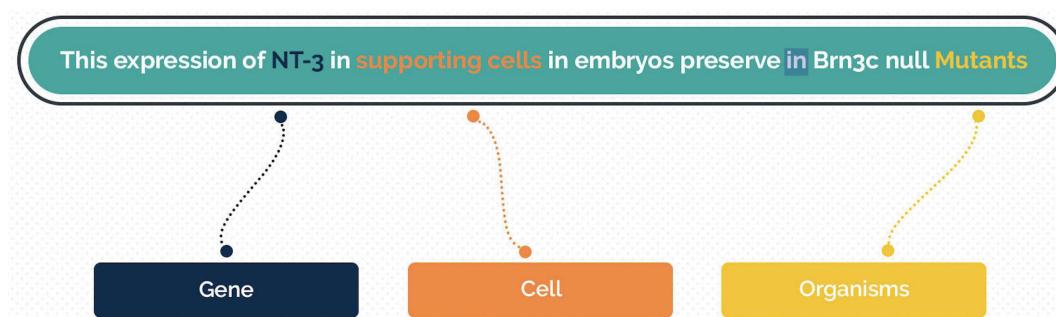
# Chương 43

## Bài toán POS và Medical NER

### 43.1 Giới thiệu



Hình 43.1: Bài toán gán nhãn từ loại (Part-of-Speech Tagging).



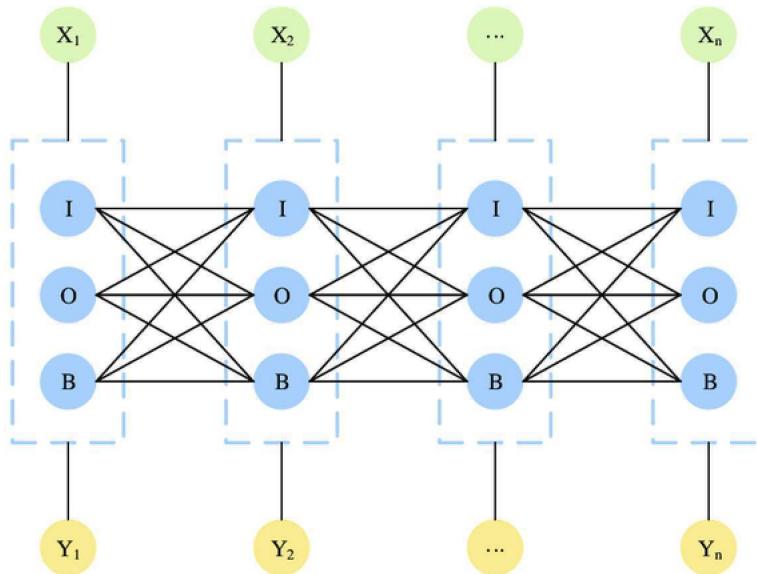
Hình 43.2: Bài toán nhận dạng thực thể cho dữ liệu y học (Medical Named Entity Recognition).

Part-of-Speech (POS) Tagging (Gán nhãn từ loại): là bài toán gán nhãn các từ trong câu tương ứng với từ loại thực hiện chức năng ngữ pháp của nó. Các từ loại được xây dựng và mở rộng phụ thuộc vào đặc trưng ngôn ngữ. Trong đó, một số từ loại điển hình thường xuất hiện trong hầu hết các ngôn ngữ trên thế giới hiện nay như: danh từ (Noun), tính từ (Adjective), động từ (Verb),... Ngoài các từ loại phổ biến này ra, dựa vào đặc trưng ngôn ngữ có thể xác định được tập các từ loại nhỏ hơn. Ví dụ với động từ trong tiếng

anh (VERB) có thể phân chia thành: VB (Verb, base form - động từ nguyên thể), VBD (Verb, past tense - động từ quá khứ),... hoặc danh từ trong tiếng anh (NOUN) có thể được phân chia thành: NN (Noun, singular or mass - Danh từ số ít), NNS (Noun, plural - Danh từ số nhiều),... Để hiểu rõ thêm về tập nhãn từ loại, các chuyên gia thường xây dựng và thống nhất cho các ngôn ngữ khác nhau, tập nhãn từ loại cơ bản thường là tập nhãn Penn Tree Bank. Ví dụ về POS Tagging được minh họa trong hình 1. Với câu đầu vào: "She sells seashells on the seashore", thông qua các mô hình gán nhãn sẽ gán nhãn từ loại cho các từ thành: "She" có nhãn từ loại là "PRP-Personal pronoun", "sells" có nhãn từ loại là "VBZ-Verb, 3rd person singular present",... đến "seashore" có nhãn từ loại là "NN-Noun, singular or mass".

Named Entity Recognition (NER - Nhận dạng thực thể): là bài toán xác định từ hoặc chuỗi từ trong văn bản là tên của một thực thể xác định, các thực thể điển hình như: tên người, tên địa danh, giới tính,... NER là bài toán có nhiều ứng dụng trong trích xuất các thông tin quan trọng thuộc nhiều lĩnh vực khác nhau. Đặc biệt là việc khai thác và trích xuất thông tin trong các bản ghi hồ sơ bệnh lâm vực y học (Medical NER). Ví dụ về Medical NER được minh họa trong hình 2. Với đoạn văn bản đầu vào: "This expression of NT-3 in supporting cells in embryos preserve in Brn3c null Mutants" thông qua mô hình nhận dạng thực thể sẽ xác định và trích xuất các thông tin quan trọng như: "NT-3" sẽ là thực thể "Gene", "supporting cells" sẽ là thực thể "Cell" và "Mutants" sẽ có thực thể là "Mutants".

Bài toán POS Tagging và NER thuộc nhóm bài toán phân loại mức từ (Token-level Text Classification). Nghĩa là với mỗi đơn vị văn bản đầu vào (các từ) sẽ được phân loại vào một lớp cụ thể (từ loại: danh từ, động từ,... trong bài POS Tagging hoặc thực thể trong bài NER)



Hình 43.3: Token-Level Text Classification.

## 43.2 Part-of-Speech Tagging

Ở trong phần này, chúng ta xây dựng mô hình POS Tagger trên bộ dữ liệu Penn Tree Bank của ngôn ngữ tiếng anh sử dụng pre-trained model như BERT,...

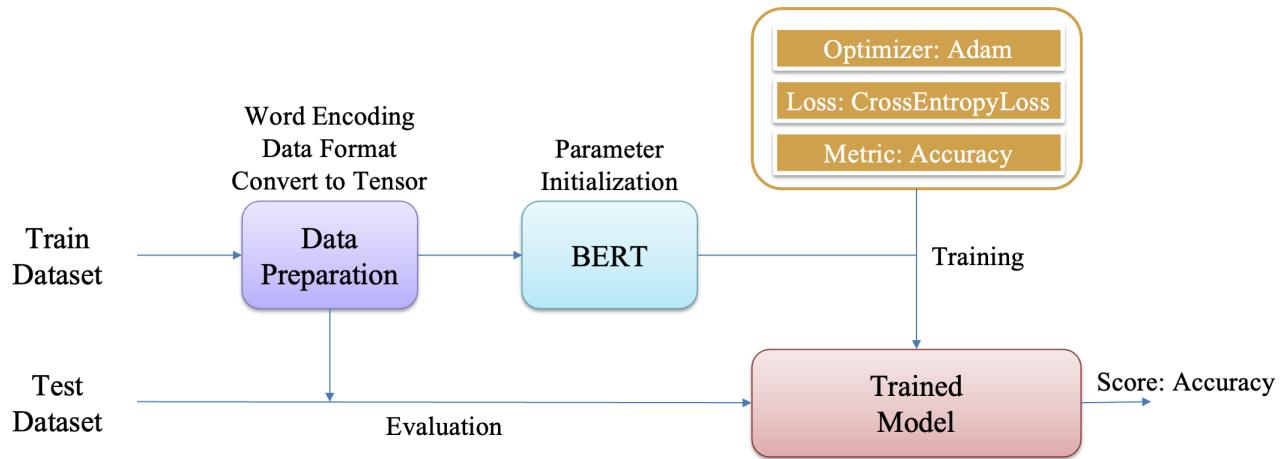
Quá trình huấn luyện và đánh giá mô hình POS Tagging dựa vào BERT được mô tả như sau"

### 1. Load Dataset

```

1 # install library
2 pip install evaluate
3
4 # import library
5 from typing import List
6 import numpy as np
7 import torch
8 import evaluate
9 from sklearn.model_selection import train_test_split
10 import nltk
11 nltk.download('treebank')

```



Hình 43.4: POS Tagging Pipeline.

```

12
13
14 # load tree bank dataset
15 tagged_sentences = nltk.corpus.treebank.tagged_sents()
16 print("Number of samples:", len(tagged_sentences))
17
18 # save sentences and tags
19 sentences, sentence_tags = [], []
20 for tagged_sentence in tagged_sentences:
21 sentence, tags = zip(*tagged_sentence)
22 sentences.append([word.lower() for word in sentence])
23 sentence_tags.append([tag for tag in tags])

```

## 2. Preprocessing

- Split dataset into train, validation and test set

```

1 train_sentences, test_sentences, train_tags, test_tags =
2 train_test_split(
3 sentences,
4 sentence_tags,
5 test_size=0.3
6)
7 valid_sentences, test_sentences, valid_tags, test_tags =
8 train_test_split(
9 test_sentences,
10 test_tags,
11 test_size=0.3
12)

```

```
10 test_size=0.5
11)
```

- Build dataset

```
1 # tokenization
2 from transformers import AutoTokenizer
3 from torch.utils.data import Dataset
4
5 model_name = "QCRI/bert-base-multilingual-cased-pos-english"
6
7 tokenizer = AutoTokenizer.from_pretrained(
8 model_name,
9 use_fast=True
10)
11 MAX_LEN = 256
12 class PostTagging_Dataset(Dataset):
13 def __init__(self,
14 sentences: List[List[str]],
15 tags: List[List[str]],
16 tokenizer,
17 label2id,
18 max_len=MAX_LEN
19):
20 super().__init__()
21 self.sentences = sentences
22 self.tags = tags
23 self.max_len = max_len
24 self.tokenizer = tokenizer
25 self.label2id = label2id
26
27 def __len__(self):
28 return len(self.sentences)
29
30 def __getitem__(self, idx):
31 input_token = self.sentences[idx]
32 label_token = self.tags[idx]
33
34 input_token = self.tokenizer.convert_tokens_to_ids(
35 input_token)
36 attention_mask = [1] * len(input_token)
37 labels = [self.label2id[token] for token in
38 label_token]
39
40 return {
41 "input_ids": self.pad_and_truncate(input_token,
```

```

40 pad_id=self.tokenizer.pad_token_id),
41 "labels": self.pad_and_truncate(labels, pad_id=
42 label2id["0"]),
43 "attention_mask": self.pad_and_truncate(
44 attention_mask, pad_id=0)
45 }
46
47 def pad_and_truncate(self, inputs: List[int], pad_id: int):
48 if len(inputs) < self.max_len:
49 padded_inputs = inputs + [pad_id] * (self.max_len -
len(inputs))
50 else:
51 padded_inputs = inputs[:self.max_len]
52 return torch.as_tensor(padded_inputs)

```

- Dataset loader

```

1 train_dataset = PostTagging_Dataset(train_sentences,
2 train_tags, tokenizer, label2id)
3 val_dataset = PostTagging_Dataset(valid_sentences, valid_tags,
4 tokenizer, label2id)
5 test_dataset = PostTagging_Dataset(test_sentences, test_tags,
6 tokenizer, label2id)

```

### 3. Modeling

```

1 from transformers import AutoTokenizer,
2 AutoModelForTokenClassification
3
4 model_name = "QCRIBert-base-multilingual-cased-pos-english"
5
6 model = AutoModelForTokenClassification.from_pretrained(
7 model_name)

```

### 4. Metric

```

1 accuracy = evaluate.load("accuracy")
2
3 ignore_label = len(label2id)
4
5 def compute_metrics(eval_pred):
6 predictions, labels = eval_pred
7 mask = labels != ignore_label
8 predictions = np.argmax(predictions, axis=-1)

```

```
9 return accuracy.compute(predictions=predictions[mask],
10 references=labels[mask])
```

## 5. Trainer

```
1 from transformers import TrainingArguments, Trainer
2
3 training_args = TrainingArguments(
4 output_dir="out_dir",
5 learning_rate=1e-5,
6 per_device_train_batch_size=16,
7 per_device_eval_batch_size=16,
8 num_train_epochs=10,
9 eval_strategy="epoch",
10 save_strategy="epoch",
11 load_best_model_at_end=True
12)
13
14 trainer = Trainer(
15 model=model,
16 args=training_args,
17 train_dataset=train_dataset,
18 eval_dataset=val_dataset,
19 tokenizer=tokenizer,
20 compute_metrics=compute_metrics,
21)
22
23 trainer.train()
```

## 6. Training

Kết quả training và accuracy trên tập test là 99.09

| Epoch | Training Loss | Validation Loss | Accuracy |
|-------|---------------|-----------------|----------|
| 1     | No log        | 0.051634        | 0.985287 |
| 2     | No log        | 0.041280        | 0.987875 |
| 3     | 0.150500      | 0.037712        | 0.988907 |
| 4     | 0.150500      | 0.035710        | 0.989532 |
| 5     | 0.150500      | 0.034171        | 0.989985 |
| 6     | 0.030800      | 0.033480        | 0.990071 |
| 7     | 0.030800      | 0.033291        | 0.990371 |
| 8     | 0.030800      | 0.032909        | 0.990437 |
| 9     | 0.024600      | 0.032835        | 0.990484 |
| 10    | 0.024600      | 0.032753        | 0.990457 |

Hình 43.5: Quá trình huấn luyện mô hình POS Tagging trên bộ dữ liệu Penn Tree Bank.

## 7. Inference

Sau quá trình huấn luyện và mô hình có độ đo đánh giá tốt nhất được sử dụng để gán nhãn cho các câu đầu vào

```

1 # tokenization
2 test_sentence = "We are exploring the topic of deep learning"
3 input = torch.as_tensor([tokenizer.convert_tokens_to_ids(
4 test_sentence.split())])
5 input = input.to("cuda")
6
7 # prediction
8 outputs = model(input)
9 _, preds = torch.max(outputs.logits, -1)
10 preds = preds[0].cpu().numpy()
11
12 # decode
13 pred_tags = ""
14 for pred in preds:
15 pred_tags += id2label[pred] + " "
15 pred_tags # => PRP VBP RB DT NN IN JJ NN

```

### 43.3 Medical NER

Ở trong phần này chúng ta sẽ xây dựng mô hình nhận dạng các thực thể trong y học dựa trên bộ dữ liệu [MACCROBAT2018](#).

Quay trở lại với ví dụ ở hình 2. Với đoạn văn bản đầu vào: "This expression of NT-3 in supporting cells in embryos preserve in Brn3c null Mutants" thông qua mô hình gán nhãn thực thể sẽ xác định và trích xuất các thông tin quan trọng như: "NT-3" sẽ là thực thể "Gene", "supporting cells" sẽ là thực thể "Cell" và "Mutants" sẽ có thực thể là "Mutants". Vì vậy, kết quả của mô hình sẽ có những từ không thuộc vào thực thể nào hoặc một thực thể có thể chứa nhiều từ. Nhưng mô hình giải quyết cho bài toán Token-Level Text Classification sẽ gán nhãn cho mỗi từ (hoặc gọi là token) thuộc vào một nhãn cụ thể. Cho nên, để giải quyết vấn đề này, chúng ta cần xây dựng một tập nhãn mới và quy bài toán NER về thành chuẩn bài toán Token-Level Text Classification.

Tập nhãn mới được xác định gồm:

1. Với những từ không thuộc vào thực thể nào sẽ được gán nhãn thành nhãn: "O"
2. Với những thực thể có một hoặc nhiều từ đi kèm, chúng ta tạo ra tập nhãn BI (hoặc I, BIE). Ví dụ, thực thể "Cell" sẽ tạo thành tập nhãn bao gồm: "B-Cell" (Xác định vị trí bắt đầu của thực thể), "I-Cell" (Xác định vị trí của các từ sau vị trí bắt đầu thực thể) hoặc có thêm "E-Cell" (Xác định vị trí kết thúc của thực thể).

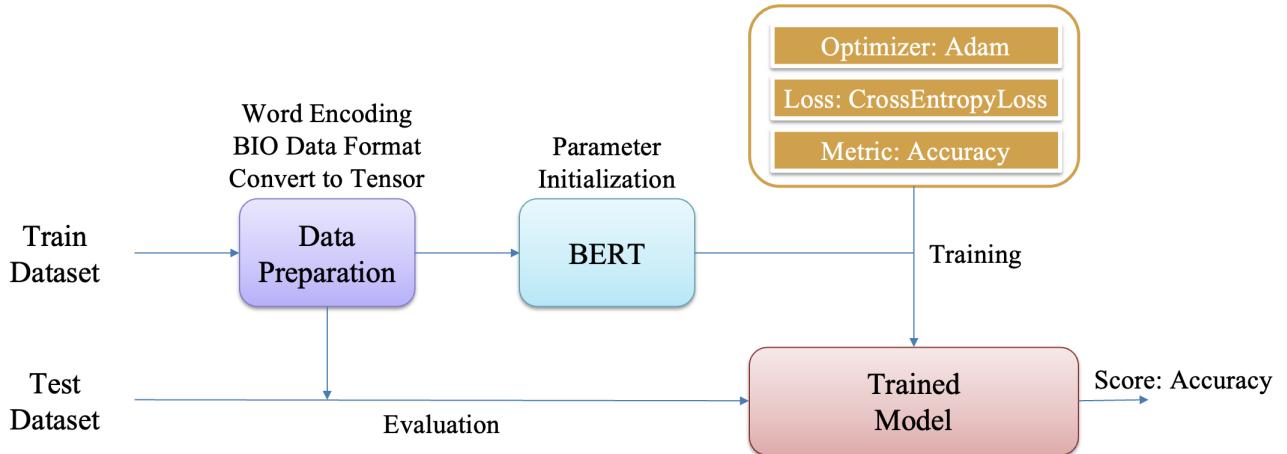
Vì vậy chúng ta có các phương pháp xác định tập nhãn mới có thể là BIO, hoặc IO, hoặc BIOE.

Vì vậy với bộ dữ liệu với  $N$  thực thể, chúng ta sẽ tạo thành bộ nhãn mới gồm:  $2*N$  (Nhãn B và I cho mỗi thực thể) + 1 (tương ứng nhãn O). Ví dụ với câu trên có 3 thực thể "Gene", "Cell", "Mutants" sẽ tạo thành 7 nhãn mới: "B-Gene", "I-Gene", "B-Cell", "I-Cell", "B-Mutants", "I-Mutants" và "O".

Do đó, bài toán chuyển thành, với đoạn văn bản đầu vào: "This expression of NT-3 in supporting cells in embryos preserve in Brn3c null Mutants" sẽ được gán nhãn thành: "O O O B-Gene O B-Cell I-Cell O O O O O B-Mutants". Vì vậy, chúng ta sẽ quy về bài toán Token-Level Text Classification và xây dựng mô hình tương tự như bài toán POS Tagging.

Tiếp theo, chúng ta sẽ xây dựng mô hình dựa trên pre-trained model như BERT,... sau khi đã tải về bộ dữ liệu [MACCROBAT2018](#).

Quá trình huấn luyện và đánh giá mô hình POS Tagging dựa vào BERT được mô tả như sau"



Hình 43.6: Named Entity Recognition Pipeline.

### 1. Load Dataset

Sau khi tải về bộ dữ liệu, chúng ta sẽ thu được folder có cấu trúc như sau:

```

15939911.ann
15939911.txt
16778410.ann
16778410.txt
17803823.ann
17803823.txt
18236639.ann
18236639.txt

```

Hình 43.7: Cấu trúc thư mục bộ dữ liệu MACCROBAT2018.

Mỗi sample tương ứng gồm 2 file: các file có đuôi "...txt", chứa đoạn văn bản và các file có đuôi "...ann" chứa các nhãn thực thể tương ứng được mô

tả theo chuẩn BioNLP Shared Task standoff format được mô tả cụ thể như hình sau:

T1 Organization 0 4 Sony  
T2 MERGE-ORG 14 27 joint venture  
T3 Organization 33 41 Ericsson  
E1 MERGE-ORG:T2 Org1:T1 Org2:T3  
T4 Country 75 81 Sweden  
R1 Origin Arg1:T3 Arg2:T4

Hình 43.8: Ví dụ về mô tả các thực thể và mối quan hệ tương ứng trong bộ dữ liệu MACCROBAT2018.

Với cột đầu tiên: T đại diện cho Entity, E đại diện cho Event và R đại diện cho Relation. Vì vậy, trong bài toán Medical NER chúng ta sẽ sử dụng các nhãn T.

Tương ứng với các thực thể T sẽ có cột thứ 2 là tên thực thể, ví dụ Organization, cột thứ 3 sẽ là vị trí trong văn bản đầu vào lấy từ file có đuôi ".txt", cột thứ 4 sẽ là đoạn văn bản tương ứng.

## 2. Preprocessing

Vì các nhãn thực thể được mô tả theo chuẩn standoff, nên trước tiên chúng ta cần tiền xử lý để lấy ra các vị trí tương ứng trong văn bản đầu vào và xây dựng tập nhãn "BIO" tương ứng.

```
1 import os
2 from typing import List, Dict, Tuple
3
4 class Preprocessing_Maccrobot:
5 def __init__(self, dataset_folder, tokenizer):
6 self.file_ids = [f.split(".")[0] for f in os.listdir(
dataset_folder) if f.endswith('.txt')]
7
8 self.text_files = [f + ".txt" for f in self.file_ids]
9 self.anno_files = [f + ".ann" for f in self.file_ids]
10
11 self.num_samples = len(self.file_ids)
12
13 self.texts: List[str] = []
14 for i in range(self.num_samples):
```

```
15 file_path = os.path.join(dataset_folder, self.
16 text_files[i])
17 with open(file_path, 'r') as f:
18 self.texts.append(f.read())
19
19 self.tags: List[Dict[str, str]] = []
20 for i in range(self.num_samples):
21 file_path = os.path.join(dataset_folder, self.
22 anno_files[i])
23 with open(file_path, 'r') as f:
24 text_bound_ann = [t.split("\t") for t in f.
25 read().split("\n") if t.startswith("T")]
26 text_bound_lst = []
27 for text_b in text_bound_ann:
28 label = text_b[1].split(" ")
29 try:
30 _ = int(label[1])
31 _ = int(label[2])
32 tag = {
33 "text": text_b[-1],
34 "label": label[0],
35 "start": label[1],
36 "end": label[2]
37 }
38 text_bound_lst.append(tag)
39 except:
40 pass
41
42 self.tags.append(text_bound_lst)
43 self.tokenizer = tokenizer
44
45 def process(self) -> Tuple[List[List[str]], List[List[str]]]:
46 input_texts = []
47 input_labels = []
48
49 for idx in range(self.num_samples):
50 full_text = self.texts[idx]
51 tags = self.tags[idx]
52
53 label_offset = []
54 continuous_label_offset = []
55 for tag in tags:
56 offset = list(range(int(tag["start"]),
57 int(tag["end"])+1))
```

```
55 label_offset.append(offset)
56 continuous_label_offset.extend(offset)
57
58 all_offset = list(range(len(full_text)))
59 zero_offset = [offset for offset in all_offset if
60 offset not in continuous_label_offset]
61 zero_offset = Preprocessing_Maccrobot.
62 find_continuous_ranges(zero_offset)
63
64 self.tokens = []
65 self.labels = []
66 self._merge_offset(full_text, tags, zero_offset,
67 label_offset)
68 assert len(self.tokens) == len(self.labels), f"
69 Length of tokens and labels are not equal"
70
71 input_texts.append(self.tokens)
72 input_labels.append(self.labels)
73
74 return input_texts, input_labels
75
76 def _merge_offset(self, full_text, tags, zero_offset,
77 label_offset):
78 i = j = 0
79 while i < len(zero_offset) and j < len(label_offset):
80 if zero_offset[i][0] < label_offset[j][0]:
81 self._add_zero(full_text, zero_offset, i)
82 i += 1
83 else:
84 self._add_label(full_text, label_offset, j,
85 tags)
86 j += 1
87
88 while i < len(zero_offset):
89 self._add_zero(full_text, zero_offset, i)
90 i += 1
91
92 while j < len(label_offset):
93 self._add_label(full_text, label_offset, j, tags)
94 j += 1
95
96 def _add_zero(self, full_text, offset, index):
97 start, *_ ,end = offset[index] if len(offset[index]) > 1 else (offset[index][0], offset[index][0]+1)
98 text = full_text[start:end]
```

```
93 text_tokens = self.tokenizer.tokenize(text)
94
95 self.tokens.extend(text_tokens)
96 self.labels.extend(
97 ["O"]*len(text_tokens)
98)
99
100 def _add_label(self, full_text, offset, index, tags):
101 start, *_ ,end = offset[index] if len(offset[index]) > 1 else (offset[index][0], offset[index][0]+1)
102 text = full_text[start:end]
103 text_tokens = self.tokenizer.tokenize(text)
104
105 self.tokens.extend(text_tokens)
106 self.labels.extend(
107 [f"B-{tags[index]['label']}"] + [f"I-{tags[index]['label']}"]*(len(text_tokens)-1)
108)
109
110 @staticmethod
111 def build_label2id(tokens: List[List[str]]):
112 label2id = {}
113 id_counter = 0
114 for token in [token for sublist in tokens for token in sublist]:
115 if token not in label2id:
116 label2id[token] = id_counter
117 id_counter += 1
118 return label2id
119
120 @staticmethod
121 def find_continuous_ranges(data: List[int]):
122 if not data:
123 return []
124 ranges = []
125 start = data[0]
126 prev = data[0]
127 for number in data[1:]:
128 if number != prev + 1:
129 ranges.append(list(range(start, prev + 1)))
130 start = number
131 prev = number
132 ranges.append(list(range(start, prev + 1)))
133 return ranges
134
```

```
135 # Preprocessing
136 from transformers import AutoTokenizer
137
138 tokenizer = AutoTokenizer.from_pretrained("d4data/biomedical-
139 ner-all")
140 dataset_folder = "./MACCROBAT2018"
141
142 Maccrobot_builder = Preprocessing_Maccrobot(dataset_folder,
143 tokenizer)
144 input_texts, input_labels = Maccrobot_builder.process()
145
146 label2id = Preprocessing_Maccrobot.build_label2id(
147 input_labels)
148 id2label = {v: k for k, v in label2id.items()}
149
150 # Split
151 from sklearn.model_selection import train_test_split
152 inputs_train, inputs_val, labels_train, labels_val =
153 train_test_split(
154 input_texts,
155 input_labels,
156 test_size=0.2,
157 random_state=42
158)
```

### 3. Dataloader

```
1 import torch
2 from torch.utils.data import Dataset
3
4 MAX_LEN = 512
5
6 class NER_Dataset(Dataset):
7 def __init__(self, input_texts, input_labels, tokenizer,
8 label2id, max_len=MAX_LEN):
9 super().__init__()
10 self.tokens = input_texts
11 self.labels = input_labels
12 self.tokenizer = tokenizer
13 self.label2id = label2id
14 self.max_len = max_len
```

```

15 def __len__(self):
16 return len(self.tokens)
17
18 def __getitem__(self, idx):
19 input_token = self.tokens[idx]
20 label_token = [self.label2id[label] for label in self
21 .labels[idx]]
22
23 input_token = self.tokenizer.convert_tokens_to_ids(
24 input_token)
25 attention_mask = [1] * len(input_token)
26
27 input_ids = self.pad_and_truncate(input_token, pad_id
28 = self.tokenizer.pad_token_id)
29 labels = self.pad_and_truncate(label_token, pad_id=0)
30 attention_mask = self.pad_and_truncate(
31 attention_mask, pad_id=0)
32
33 return {
34 "input_ids": torch.as_tensor(input_ids),
35 "labels": torch.as_tensor(labels),
36 "attention_mask": torch.as_tensor(attention_mask)
37 }
38
39 def pad_and_truncate(self, inputs: List[int], pad_id: int
40):
41 if len(inputs) < self.max_len:
42 padded_inputs = inputs + [pad_id] * (self.max_len
43 - len(inputs))
44 else:
45 padded_inputs = inputs[:self.max_len]
46 return padded_inputs
47
48 def label2id(self, labels: List[str]):
49 return [self.label2id[label] for label in labels]
50
51 train_set = NER_Dataset(inputs_train, labels_train, tokenizer
52 , label2id)
53 val_set = NER_Dataset(inputs_val, labels_val, tokenizer,
54 label2id)

```

#### 4. Modeling

```

1 from transformers import AutoModelForTokenClassification
2 model = AutoModelForTokenClassification.from_pretrained(

```

```
3 "d4data/biomedical-ner-all",
4 label2id=label2id,
5 id2label=id2label,
6 ignore_mismatched_sizes=True
7)
```

## 5. Metric

```
1 import evaluate
2 import numpy as np
3
4 accuracy = evaluate.load("accuracy")
5
6 def compute_metrics(eval_pred):
7 predictions, labels = eval_pred
8 mask = labels != 0
9 predictions = np.argmax(predictions, axis=-1)
10 return accuracy.compute(predictions=predictions[mask],
11 references=labels[mask])
```

## 6. Trainer

```
1 from transformers import TrainingArguments, Trainer
2
3 training_args = TrainingArguments(
4 output_dir="out_dir",
5 learning_rate=1e-4,
6 per_device_train_batch_size=16,
7 per_device_eval_batch_size=16,
8 num_train_epochs=20,
9 eval_strategy="epoch",
10 save_strategy="epoch",
11 load_best_model_at_end=True,
12 optim="adamw_torch"
13)
14
15 trainer = Trainer(
16 model=model,
17 args=training_args,
18 train_dataset=train_set,
19 eval_dataset=val_set,
20 tokenizer=tokenizer,
21 compute_metrics=compute_metrics,
22)
```

```

23
24 trainer.train()

```

## 7. Training

Kết quả training và accuracy trên tập validation là 78.5%

| Epoch | Training Loss | Validation Loss | Accuracy |
|-------|---------------|-----------------|----------|
| 1     | No log        | 1.474108        | 0.323930 |
| 2     | No log        | 0.900565        | 0.600812 |
| 3     | No log        | 0.721831        | 0.694619 |
| 4     | No log        | 0.610314        | 0.738286 |
| 5     | No log        | 0.573626        | 0.760023 |
| 6     | No log        | 0.567352        | 0.763211 |
| 7     | No log        | 0.563499        | 0.765916 |
| 8     | No log        | 0.558194        | 0.774804 |
| 9     | No log        | 0.562186        | 0.777026 |
| 10    | No log        | 0.565547        | 0.779828 |

Hình 43.9: Quá trình huấn luyện trên bộ dữ liệu MACCROBAT2018.

## 8. Inference

```

1 test_sentence = """A 48 year-old female presented with
 vaginal bleeding and abnormal Pap smears. Upon diagnosis
 of invasive non-keratinizing SCC of the cervix, she
 underwent a radical hysterectomy with salpingo-
 oophorectomy which demonstrated positive spread to the
 pelvic lymph nodes and the parametrium. Pathological
 examination revealed that the tumour also extensively
 involved the lower uterine segment. """
2
3 # tokenization
4 input = torch.as_tensor([tokenizer.convert_tokens_to_ids(
 test_sentence.split())])
5

```

```
6 input = input.to("cuda")
7
8 # prediction
9 outputs = model(input)
10 _, preds = torch.max(outputs.logits, -1)
11 preds = preds[0].cpu().numpy()
12
13 # decode
14 for token, pred in zip(test_sentence.split(), preds):
15 print(f"{token}\t{id2label[pred]}")
```

## 9. Deployment

Triển khai ứng dụng sử dụng streamlit tham khảo mã nguồn [tại đây](#), thử nghiệm ứng dụng [tại đây](#). Giao diện và kết quả:

## Model: DistilBERT. Dataset: MACCROBAT2018

Sentence:

A 48 year - old female presented with vaginal bleeding and abnormal Pap smears . Upon diagnosis of

A 48 year - old female presented with vaginal bleeding and abnormal Pap smears . Upon diagnosis of invasive non - keratinizing SCC of the cervix , she underwent a radical hysterectomy with salpingo - oophorectomy which demonstrated positive spread to the pelvic lymph nodes and the parametrium . Pathological examination revealed that the tumour also extensively involved the lower uterine segment .

```

▼ [
 ▼ 0 : {
 "entity_group" : "Age"
 "score" : "0.888535"
 "word" : "48 year - old"
 "start" : 2
 "end" : 15
 }
 ▼ 1 : {
 "entity_group" : "Sex"
 "score" : "0.7129116"
 "word" : "female"
 "start" : 16
 "end" : 22
 }
]

```

Hình 43.10: Triển khai ứng dụng.

### 43.4 Câu hỏi trắc nghiệm

**Câu hỏi 43** Nhiệm vụ của bài toán gán nhãn từ loại (Part-of-Speech Tagging) là?

- a) Phân tích cảm xúc
- b) Dịch máy
- c) Hỏi đáp

d) Xác định loại từ của các từ trong câu

**Câu hỏi 44** Bộ dữ liệu nào sau đây được sử dụng để huấn luyện mô hình POS Tagging?

- a) IMDB-Review
- b) Penn Tree Bank
- c) BioNLP
- d) SQuAD

**Câu hỏi 45** Số lượng sample của bộ dữ liệu sử dụng để huấn luyện mô hình POS Tagging trong phần thực nghiệm là?

- a) 3914
- b) 3915
- c) 3916
- d) 3917

**Câu hỏi 46** Số lượng nhãn từ loại được sử dụng trong xây dựng mô hình POS Tagging là?

- a) 45
- b) 46
- c) 47
- d) 48

**Câu hỏi 47** Mục đích của bài toán Named Entity Recognition là gì?

- a) Xác định các đoạn văn bản chứa các thực thể
- b) Xác định loại từ của các từ trong câu
- c) Cả 2 đáp án a và b đều đúng
- d) Cả 2 đáp án a và b đều sai

**Câu hỏi 48** Với N thực thể và sử dụng các đánh nhãn "BIO" thì số nhãn mới thu được sẽ là bao nhiêu?

- a)  $2^*N + 4$
- b)  $2^*N + 3$
- c)  $2^*N + 2$
- d)  $2^*N + 1$

**Câu hỏi 49** Với N thực thể và sử dụng các đánh nhãn "BIOE" thì số nhãn mới thu được sẽ là bao nhiêu?

- a)  $3^*N + 4$
- b)  $3^*N + 3$
- c)  $3^*N + 2$
- d)  $3^*N + 1$

**Câu hỏi 50** Bộ dữ liệu sử dụng cho nhận dạng thực thể trong y học được sử dụng trong phần thực nghiệm là?

- a) NCBI-Disease
- b) BioNLP13
- c) MACCROBAT2018
- d) BC5CDR

**Câu hỏi 51** Mô hình tiền huấn luyện được sử dụng cho bài toán Medical NER là?

- a) biomedical-base
- b) d4data/biomedical-ner-all
- c) bert-base-uncased
- d) bert-large

**Câu hỏi 52** Số lượng nhãn "BIO" được sử dụng cho bộ dữ liệu MACCROBAT2018 là?

- a) 81
- b) 82
- c) 83
- d) 84

## 43.5 Phụ lục

1. **Hint:** Dựa vào file tải về [tại đây](#) để hoàn thiện các đoạn code.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

- *Hết* -

# Chương 44

## Project 1: Aspect-Based Sentiment Analysis (Text Classification + NER)

### 44.1 Giới thiệu

The diagram illustrates the decomposition of a sentence into aspect terms and their associated sentiment pairs. The sentence is: "S<sub>i</sub>: The **price** was **too high**, but the **cab** was **amazing**". The word "price" is labeled with "sp<sup>1</sup> negative" and "a<sup>1</sup>". The word "cab" is labeled with "sp<sup>2</sup> positive" and "a<sup>2</sup>".

| Subtask                                             | Input                                                             | Output                                                                                                     |
|-----------------------------------------------------|-------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| Aspect Term Extraction (ATE)                        | S <sub>i</sub>                                                    | a <sup>1</sup> , a <sup>2</sup>                                                                            |
| Aspect Term Sentiment Classification (ATSC)         | S <sub>i</sub> + a <sup>1</sup> , S <sub>i</sub> + a <sup>2</sup> | sp <sup>1</sup> , sp <sup>2</sup>                                                                          |
| Aspect Sentiment Pair Extraction (ASPE)             | S <sub>i</sub>                                                    | (a <sup>1</sup> , sp <sup>1</sup> ), (a <sup>2</sup> , sp <sup>2</sup> )                                   |
| Aspect Oriented Opinion Extraction (AOOE)           | S <sub>i</sub> + a <sup>1</sup> , S <sub>i</sub> + a <sup>2</sup> | o <sup>1</sup> , o <sup>2</sup>                                                                            |
| Aspect Opinion Pair Extraction (AOPE)               | S <sub>i</sub>                                                    | (a <sup>1</sup> , o <sup>1</sup> ), (a <sup>2</sup> , o <sup>2</sup> )                                     |
| Aspect Opinion Sentiment Triplet Extraction (AOSTE) | S <sub>i</sub>                                                    | (a <sup>1</sup> , o <sup>1</sup> , sp <sup>1</sup> ), (a <sup>2</sup> , o <sup>2</sup> , sp <sup>2</sup> ) |

Hình 44.1: Phân tích cảm xúc mức khía cạnh (Aspect-Based Sentiment Analysis)

Phân tích cảm xúc mức khía cạnh (Aspect-Based Sentiment Analysis) là bài toán có nhiều ứng dụng hiện nay trong việc phân tích các khía cạnh khác nhau của các bình luận, đánh giá về các sản phẩm, dịch vụ,... Ví dụ mình hoạ được minh họa như hình 1, với câu đầu vào: "The price was too high,

but the cab was amazing.". Trong câu này, có hai khía cạnh được đánh giá là: "price" và "amazing". Với khía cạnh "price" được đánh giá là "too high" nên sẽ là "negative" và với khía cạnh "positive" được đánh giá là "amazing" nên sẽ là "positive".

Dựa vào việc xác định các giá trị đầu ra của mô hình, chúng ta có thể có một số bài toán nhỏ hơn như sau:

1. Aspect Term Extraction (ATE) hoặc Aspect-Based Term Extraction: trích xuất các khía cạnh được đánh giá trong bình luận
2. Aspect Term Sentiment Classification (ATSC): dựa vào câu đầu vào và các khía cạnh được đánh giá trong bình luận để dự đoán cảm xúc của bình luận
3. Aspect Sentiment Pair Extraction (ASPE): dựa vào câu đầu vào, trích xuất ra khía cạnh và dự đoán cảm xúc của các khía cạnh
4. Aspect Oriented Opinion Extraction (AOOE): dựa vào câu đầu vào và khía cạnh, dự đoán đoạn văn bản thể hiện cảm xúc của khía cạnh
5. Aspect Opinion Pair Extraction (AOPE): dựa vào câu đầu vào trích xuất thông tin về khía cạnh và đoạn văn bản thể hiện cảm xúc của khía cạnh
6. Aspect Opinion Sentiment Triplet Extraction (AOSTE): dựa vào câu đầu vào, trích xuất các thông tin về khía cạnh, đoạn văn bản thể hiện cảm xúc của khía cạnh và cảm xúc của khía cạnh trong bình luận

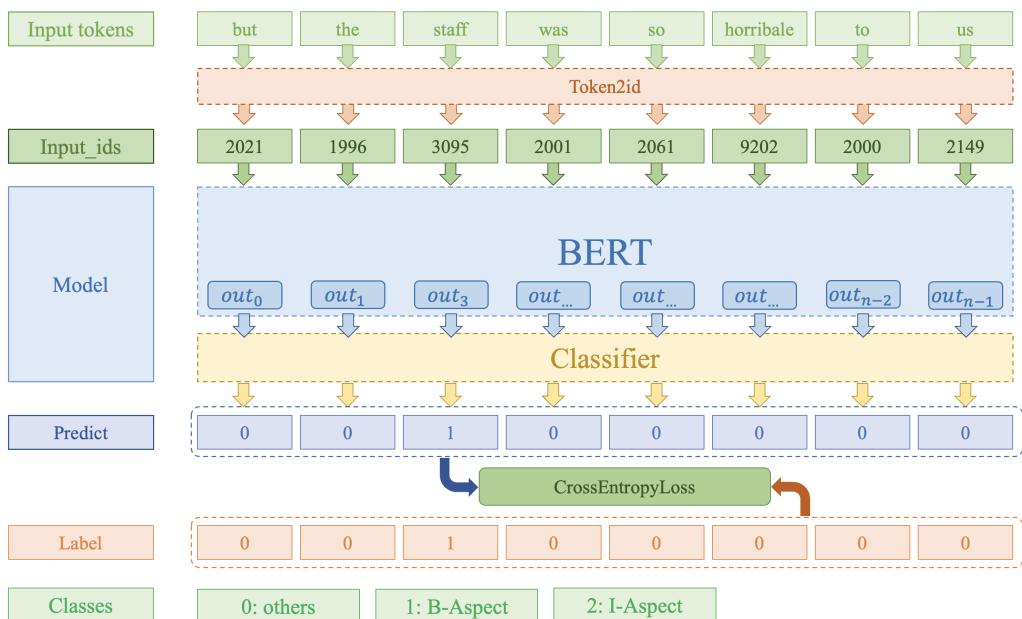
Trong phần này, chúng ta sẽ xây dựng mô hình giải quyết vấn đề cho bài toán ASPE. Dựa vào 2 bước chính:

1. Bước 1: Dự đoán từ trong văn bản thể hiện khía cạnh, hay chính là bài toán ATE
2. Bước 2: Dựa vào văn bản đầu vào và vị trí từ thể hiện khía cạnh dự đoán cảm xúc cho khía cạnh, hay là bài toán ATSC

Các thực nghiệm được xây dựng trên bộ dữ liệu [SemEval-2014 Task 4: Aspect Based Sentiment Analysis](#) và đã qua tiền xử lý cơ bản gồm: xoá bỏ dấu câu, chuẩn hoá và tách dựa vào khoảng trắng.

## 44.2 Aspect Term Extraction

Ở trong phần này, chúng ta xây dựng mô hình ATE dựa vào các phương pháp giải quyết bài toán sequence classification được sử dụng trong POS Tagging, NER,...



Hình 44.2: Aspect term extraction pipeline.

### 1. Build Dataset

```

1 # install lib
2 !pip install -q datasets==3.2.0
3
4 from datasets import load_dataset
5
6 ds = load_dataset("thainq107/abte-restaurants")

```

### 2. Tokenization

```

1 from transformers import AutoTokenizer
2

```

```

3 tokenizer = AutoTokenizer.from_pretrained("distilbert/
4 distilbert-base-uncased")
5
6 def tokenize_and_align_labels(examples):
7 tokenized_inputs = []
8 labels = []
9 for tokens, tags in zip(examples['Tokens'], examples['
10 Tags']):
11 tokens = tokens.replace('"', "").strip("]【").split(
12 ', ')
13 tags = tags.strip(']【').split(', ')
14
15 bert_tokens = []
16 bert_tags = []
17 for i in range(len(tokens)):
18 t = tokenizer.tokenize(tokens[i])
19 bert_tokens += t
20 bert_tags += [int(tags[i])] * len(t)
21
22 bert_ids = tokenizer.convert_tokens_to_ids(
23 bert_tokens)
24
25 tokenized_inputs.append(bert_ids)
26 labels.append(bert_tags)
27
28 return {
29 'input_ids': tokenized_inputs,
30 'labels': labels
31 }
32
33 preprocessed_ds = ds.map(tokenize_and_align_labels, batched=
34 True)

```

### 3. Data Collator

```

1 from transformers import DataCollatorForTokenClassification
2
3 data_collator = DataCollatorForTokenClassification(tokenizer=
4 tokenizer)

```

### 4. Evaluate

```

1 # Install lib
2 !pip install -q seqeval==1.2.2

```

```

3 import numpy as np
4 from seqeval.metrics import accuracy_score
5
6 def compute_metrics(p):
7 predictions, labels = p
8 predictions = np.argmax(predictions, axis=2)
9 true_predictions = [
10 [str(p) for (p, l) in zip(prediction, label) if l != -100]
11 for prediction, label in zip(predictions, labels)
12]
13 true_labels = [
14 [str(l) for (p, l) in zip(prediction, label) if l != -100]
15 for prediction, label in zip(predictions, labels)
16]
17 results = accuracy_score(true_predictions, true_labels)
18 return {"accuracy": results}

```

## 5. Model

```

1 from transformers import AutoModelForTokenClassification
2 id2label = {
3 0: "O",
4 1: "B-Term",
5 2: "I-Term"
6 }
7 label2id = {
8 "O": 0,
9 "B-Term": 1,
10 "I-Term": 2
11 }
12
13 model = AutoModelForTokenClassification.from_pretrained(
14 "distilbert/distilbert-base-uncased",
15 num_labels=3, id2label=id2label, label2id=label2id
16)

```

## 6. Training

```

1 import os
2 os.environ['WANDB_DISABLED'] = 'true'
3 from transformers import TrainingArguments, Trainer
4

```

```

5 training_args = TrainingArguments(
6 output_dir="abte-restaurants-distilbert-base-uncased",
7 learning_rate=2e-5,
8 per_device_train_batch_size=16,
9 per_device_eval_batch_size=16,
10 num_train_epochs=5,
11 weight_decay=0.01,
12 eval_strategy="epoch",
13 save_strategy="epoch",
14 load_best_model_at_end=True
15)
16
17 trainer = Trainer(
18 model=model,
19 args=training_args,
20 train_dataset=preprocessed_ds["train"],
21 eval_dataset=preprocessed_ds["test"],
22 processing_class=tokenizer,
23 data_collator=data_collator,
24 compute_metrics=compute_metrics,
25)
26
27 trainer.train()

```

Kết quả training và accuracy trên tập test là 81.73%

| <b>Epoch</b> | <b>Training Loss</b> | <b>Validation Loss</b> | <b>Accuracy</b> |
|--------------|----------------------|------------------------|-----------------|
| 1            | No log               | 0.476034               | 0.802080        |
| 2            | No log               | 0.473619               | 0.817293        |
| 3            | 0.339000             | 0.511703               | 0.814453        |
| 4            | 0.339000             | 0.557133               | 0.811468        |
| 5            | 0.167300             | 0.581737               | 0.810120        |

Hình 44.3: Quá trình huấn luyện mô hình ATE.

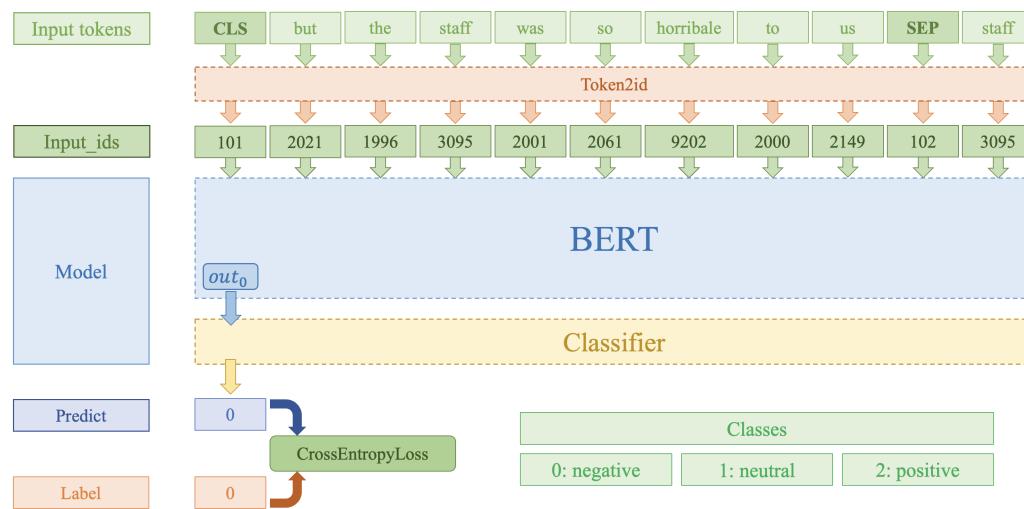
## 7. Prediction

Sau quá trình huấn luyện và mô hình có độ đo đánh giá tốt nhất được sử dụng để gán nhãn cho các câu đầu vào

```
1 from transformers import pipeline
2
3 token_classifier = pipeline(
4 model="thainq107/abte-restaurants-distilbert-base-uncased",
5 aggregation_strategy="simple"
6)
7
8 test_sentence = 'The bread is top notch as well'
9 results = token_classifier(test_sentence)
10 results => bread
```

### 44.3 Aspect Term Sentiment Classification

Trong phần 1, chúng ta đã tìm được từ là khía cạnh trong văn bản đầu vào, ở phần này, chúng ta xây dựng mô hình dựa vào câu đầu vào và khía cạnh được dự đoán xây dựng mô hình dự đoán cảm xúc.



Hình 44.4: Aspect term sentiment classification pipeline.

#### 1. Build Dataset

```

1 # Install lib
2 !pip install -q datasets==3.2.0
3
4 from datasets import load_dataset
5
6 ds = load_dataset("thainq107/abte-restaurants")

```

#### 2. Tokenization

```

1 from transformers import AutoTokenizer
2
3 tokenizer = AutoTokenizer.from_pretrained("distilbert/
 distilbert-base-uncased")
4
5 def tokenize_and_align_labels(examples):

```

```

6 sentences, sentence_tags = [], []
7 labels = []
8 for tokens, pols in zip(examples['Tokens'], examples['Polarities']):
9 tokens = tokens.replace(" ", "").strip("]【").split(',')
10 pols = pols.strip('】【').split(', ')
11
12 bert_tokens = []
13 bert_att = []
14 pols_label = 0
15 for i in range(len(tokens)):
16 t = tokenizer.tokenize(tokens[i])
17 bert_tokens += t
18 if int(pols[i]) != -1:
19 bert_att += t
20 pols_label = int(pols[i])
21
22 sentences.append(" ".join(bert_tokens))
23 sentence_tags.append(" ".join(bert_att))
24 labels.append(pols_label)
25
26 tokenized_inputs = tokenizer(sentences, sentence_tags,
27 padding=True, truncation=True, return_tensors="pt")
28 tokenized_inputs['labels'] = labels
29 return tokenized_inputs
30
31 preprocessed_ds = ds.map(tokenize_and_align_labels, batched=
32 True)

```

### 3. Evaluate

```

1 # Install lib
2 !pip install -q evaluate==0.4.3
3
4 import evaluate
5 import numpy as np
6
7 accuracy = evaluate.load("accuracy")
8
9 def compute_metrics(eval_pred):
10 predictions, labels = eval_pred
11 predictions = np.argmax(predictions, axis=1)
12 return accuracy.compute(predictions=predictions,
13 references=labels)

```

## 4. Model

```
1 from transformers import AutoModelForSequenceClassification
2
3 id2label = {0: 'Negative', 1: 'Neutral', 2: 'Positive'}
4 label2id = {'Negative': 0, 'Neutral': 1, 'Positive': 2}
5
6 model = AutoModelForSequenceClassification.from_pretrained(
7 "distilbert/distilbert-base-uncased", num_labels=3,
8 id2label=id2label, label2id=label2id
9)
```

## 5. Training

```
1 import os
2 from transformers import TrainingArguments, Trainer
3
4 os.environ['WANDB_DISABLED'] = 'true'
5
6 training_args = TrainingArguments(
7 output_dir="absa-restaurants-distilbert-base-uncased",
8 learning_rate=2e-5,
9 per_device_train_batch_size=16,
10 per_device_eval_batch_size=16,
11 num_train_epochs=5,
12 weight_decay=0.01,
13 eval_strategy="epoch",
14 save_strategy="epoch",
15 load_best_model_at_end=True
16)
17
18 trainer = Trainer(
19 model=model,
20 args=training_args,
21 train_dataset=preprocessed_ds["train"],
22 eval_dataset=preprocessed_ds["test"],
23 processing_class=tokenizer,
24 compute_metrics=compute_metrics,
25)
26 trainer.train()
```

Kết quả training và accuracy trên tập test là 79.18

## 6. Prediction

| Epoch | Training Loss | Validation Loss | Accuracy |
|-------|---------------|-----------------|----------|
| 1     | No log        | 0.711144        | 0.733691 |
| 2     | No log        | 0.651741        | 0.744415 |
| 3     | 0.694500      | 0.629071        | 0.760500 |
| 4     | 0.694500      | 0.598540        | 0.783735 |
| 5     | 0.425200      | 0.584476        | 0.791778 |

Hình 44.5: Quá trình huấn luyện ATSC.

```

1 from transformers import pipeline
2
3 token_classifier = pipeline(
4 model="thainq107/abte-restaurants-distilbert-base-uncased",
5 aggregation_strategy="simple"
6)
7
8 classifier = pipeline(
9 model="thainq107/absa-restaurants-distilbert-base-uncased",
10 ""
11)
12 test_sentence = 'The bread is top notch as well'
13 results = token_classifier(test_sentence)
14 sentence_tags = " ".join([result['word'] for result in
15 results])
15 pred_label = classifier(f'{test_sentence} [SEP] {
16 sentence_tags}')
16 sentence_tags, pred_label
17 # ('bread', [{'label': 'Positive', 'score':
18 0.9864555597305298}])

```

### 7. Deployment

Triển khai ứng dụng sử dụng streamlit tham khảo mã nguồn [tại đây](#), thử nghiệm ứng dụng [tại đây](#). Giao diện và kết quả:

## Aspect-based Sentiment Analysis

### Model: DistilBERT. Dataset: SemEval4 Restaurants

Sentence:

The bread is top notch as well

Sentence: The bread is top notch as well === Term: bread === Sentiment: Positive

Hình 44.6: Triển khai ứng dụng.

#### 44.4 Câu hỏi trắc nghiệm

Cho câu bình luận như sau: "The food is very fresh."

**Câu hỏi 53** Dựa vào câu bình luận trên xác định giá trị dự đoán mô hình giải quyết bài toán ATE?

- a) food
- b) food, positive
- c) food, very, fresh
- d) food, very fresh, positive

**Câu hỏi 54** Dựa vào câu bình luận trên xác định giá trị dự đoán mô hình giải quyết bài toán ASPE?

- a) food
- b) food, positive
- c) food, very, fresh
- d) food, very fresh, positive

**Câu hỏi 55** Dựa vào câu bình luận trên xác định giá trị dự đoán mô hình giải quyết bài toán AOPE?

- a) food
- b) food, positive
- c) food, very, fresh
- d) food, very fresh, positive

**Câu hỏi 56** Dựa vào câu bình luận trên xác định giá trị dự đoán mô hình giải quyết bài toán AOSTE?

- a) food
- b) food, positive
- c) food, very, fresh
- d) food, very fresh, positive

**Câu hỏi 57** Mô hình giải quyết bài toán ATE ở phần 2 trên sử dụng bộ dữ

liệu nào?

- a) IMDB-Review
- b) Penn Tree Bank
- c) SemEval-2014 Task 4
- d) SQuAD

**Câu hỏi 58** Số lượng samples được sử dụng cho tập train trong phần thực nghiệm là

- a) 3600
- b) 3601
- c) 3602
- d) 3603

**Câu hỏi 59** Mô hình giải quyết bài toán ATE ở phần 2 trên dựa vào bài toán tổng quát nào sau đây?

- a) Sequence Labeling
- b) Machine Translation
- c) Topic Modeling
- d) Summarization

**Câu hỏi 60** Mô hình tiền huấn luyện sử dụng cho bài toán ATE ở phần 2 trên là?

- a) BERT
- b) RoBERTA
- c) DEBERTA
- d) DistilBERT

**Câu hỏi 61** Mô hình tiền huấn luyện sử dụng cho bài toán ASTC ở phần 3 trên là?

- a) BERT
- b) RoBERTA
- c) DEBERTA
- d) DistilBERT

**Câu hỏi 62** Hàm mục tiêu để giải quyết bài toán ASTC ở phần 3 là

- a) Masked Language Model
- b) Next Sentence Prediction
- c) Cả 2 đáp án trên đều đúng
- d) Cả 2 đáp án trên đều sai

## 44.5 Phụ lục

1. **Hint:** Dựa vào file tải về [Term Extraction](#) và [Sentiment Analysis](#) để hoàn thiện các đoạn code.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

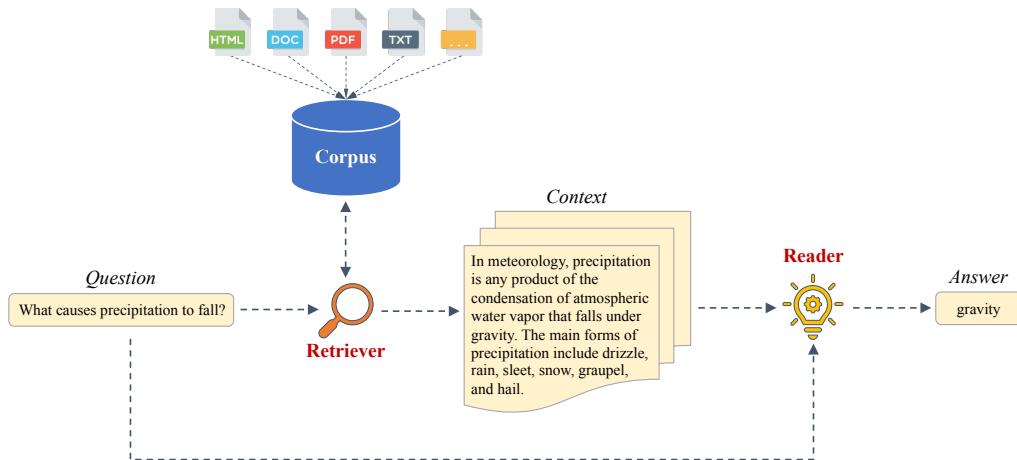
- *Hết* -

# Chương 45

## Project 2: Xây dựng hệ thống end-to-end Question Answering

### 45.1 Giới thiệu

**End-to-end Question Answering** là một bài toán thuộc lĩnh vực Xử lý ngôn ngữ tự nhiên liên quan đến việc xây dựng một hệ thống hỏi đáp sử dụng công cụ truy vấn để tìm kiếm các tài liệu có liên quan đến câu hỏi, từ đó lựa chọn câu trả lời phù hợp nhất đến người dùng. End-to-end Question Answering cũng là một ý tưởng tiền đề khởi nguồn cho một trong những khái niệm phổ biến trong khía cạnh cải thiện các mô hình ngôn ngữ lớn (LLMs), còn được gọi là Retrieval Augmented Generation (RAG).



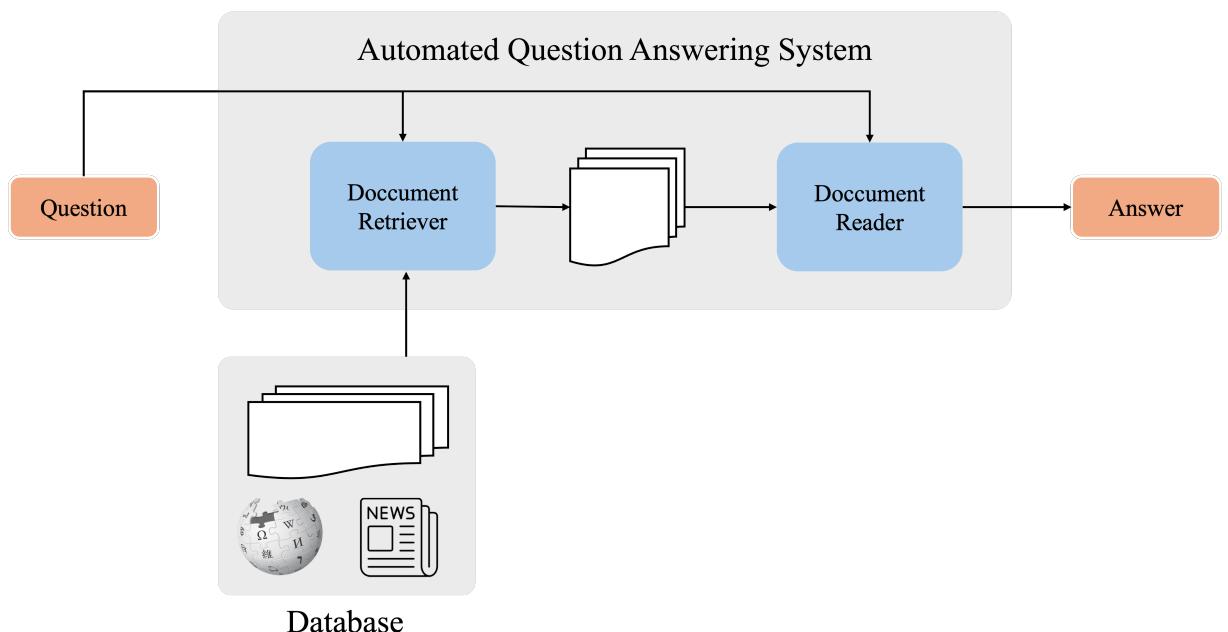
Hình 45.1: Ảnh minh họa tổng quát về một hệ thống End-to-end QA.

Trong project này, chúng ta sẽ cùng tìm hiểu việc phát triển một hệ thống end-to-end hỏi đáp tự động, với khả năng trả lời một câu hỏi với nội dung bất kì. Theo đó, Input và output của chương trình như sau:

- **Input:** Một câu hỏi.

- **Output:** Câu trả lời tương ứng.

Cụ thể, ta sẽ xây dựng chương trình dựa vào dataset SQuAD2.0, một bộ dữ liệu về đọc hiểu, vector database là FAISS và mô hình BERT để thực hiện các nhiệm vụ cụ thể trong chương trình. Để xây dựng một chương trình End-to-end Question Answering, chúng ta cần hoàn thiện hai module chính bao gồm Retriever và Reader.



Hình 45.2: Minh họa pipeline của bài toán End-to-end Question Answering.

- **Retriever:** Thành phần truy vấn các tài liệu có liên quan đến câu hỏi có trong cơ sở dữ liệu.
- **Reader:** Dựa trên các tài liệu truy vấn được và câu hỏi đầu vào, trích xuất vùng văn bản chứa câu trả lời chính xác nhất và gửi tới người dùng.

## 45.2 Cài đặt chương trình

Trong phần này, quá trình cài đặt toàn bộ chương trình End-to-end Question Answering được mô tả như sau:

### 45.2.1 Dataset SQuAD2.0

Stanford Question Answering Dataset (SQuAD) là bộ data theo hướng đọc hiểu, bao gồm các đoạn văn (passage) khác nhau về nhiều chủ đề, ứng với mỗi đoạn văn sẽ có một các câu hỏi ngắn kèm theo. Bảng 45.1 miêu tả cấu trúc chi tiết về dataset SQuAD2.0:

**Context:** The English name "Normans" comes from the French words Normans/Norman, plural of Normant, modern French normand, which is itself borrowed from (Old Low Franconian Nortmann "Northman" or directly from Old Norse Nordmaðr, Latinized variously as Nortmannus, Normannus, or Nordmannus (recorded in Medieval Latin, **9th century**) to mean "Norseman, **Viking**".

**Q1:** What causes precipitation to fall?  
**A1:** Viking

**Q2:** When was the Latin version of the word Norman first recorded?  
**A2:** 9th century

**Q3:** When was the French version of the word Norman first recorded?  
**A3:** No Answers

**Context:** In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

**Q1:** What causes precipitation to fall?  
**A1:** Gravity

**Q2:** What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?  
**A2:** Graupel

**Q3:** Where do water droplets collide with ice crystals to form precipitation?  
**A3:** Within a cloud

Hình 45.3: Ví dụ minh họa về dataset SQuAD2.0.

Câu trả lời cho các câu hỏi ngắn là những từ/cụm từ có sẵn trong đoạn văn cho trước (không yêu cầu suy luận phức tạp), hoặc các câu hỏi không trả lời được dựa vào đoạn văn (answer là no answer). Bảng 45.2 thống kê về dataset SQuAD2.0:

### 45.2.2 Reader: DistilBERT

Đầu tiên ta sẽ xây dựng model Reader hay chính là model QA trong project này.

| <b>SQuAD 2.0</b>        |         |
|-------------------------|---------|
| <b>Train</b>            |         |
| Total examples          | 130,319 |
| Negative examples       | 43,498  |
| Total articles          | 442     |
| Articles with negatives | 285     |
| <b>Development</b>      |         |
| Total examples          | 11,873  |
| Negative examples       | 5,945   |
| Total articles          | 35      |
| Articles with negatives | 35      |
| <b>Test</b>             |         |
| Total examples          | 8,862   |
| Negative examples       | 4,332   |
| Total articles          | 28      |
| Articles with negatives | 28      |

Bảng 45.1: Thống kê số lượng sample của dataset SQuAD2.0.

| <b>Answer type</b> | <b>Percentage</b> | <b>Example</b>          |
|--------------------|-------------------|-------------------------|
| Date               | 8.9%              | 19 October 2023         |
| Other Numeric      | 10.9%             | 12                      |
| Person             | 12.9%             | Thomas Coke             |
| Location           | 4.4%              | Germany                 |
| Other Entity       | 15.3%             | ABC Sports              |
| Common Noun Phrase | 31.8%             | property damage         |
| Adjective Phrase   | 3.9%              | second-largest          |
| Verb Phrase        | 5.5%              | returned to Earth       |
| Clause             | 3.7%              | to avoid trivialization |
| Other              | 2.7%              | quietly                 |

Bảng 45.2: Thống kê các loại câu trả lời khác nhau của dataset SQuAD2.0.

### Install and import libraries

Đầu tiên ta sẽ install một số thư viện cần thiết mà Colab chưa hỗ trợ.

```
1 !pip install -qq datasets==2.16.1 evaluate==0.4.1 transformers[
 sentencepiece]==4.35.2
```

```

2 !pip install -qq accelerate==0.26.1
3 !apt install git-lfs

```

Sau đó ta sẽ tiến hành login vào HuggingFace để download dataset và model có sẵn. Khi chạy block code này thì HuggingFace sẽ đưa ra một đường dẫn đến trang [HuggingFace](#) để lấy mã token.

```

1 from huggingface_hub import notebook_login
2
3 notebook_login()

```

Cuối cùng ta sẽ import các thư viện chính được sử dụng trong phần này:

```

1 import numpy as np
2 from tqdm.auto import tqdm
3 import collections
4
5 import torch
6
7 from datasets import load_dataset
8 from transformers import AutoTokenizer
9 from transformers import AutoModelForQuestionAnswering
10 from transformers import TrainingArguments
11 from transformers import Trainer
12 import evaluate
13
14 device = torch.device("cuda") if torch.cuda.is_available() else \
15 torch.device("cpu")

```

## Setup config

Tiếp theo ta sẽ setup một số config cơ bản:

```

1 # Sử dụng mô hình "distilbert-base-uncased" làm mô hình checkpoint
2 MODEL_NAME = "distilbert-base-uncased"
3
4 # Độ dài tối đa cho mỗi đoạn văn bản sau khi được xử lý
5 MAX_LENGTH = 384
6
7 # Khoảng cách giữa các điểm bắt đầu của các đoạn văn bản liên tiếp
8 STRIDE = 128

```

## Setup Dataset

- Download dataset:

```

1 # Download squad dataset từ HuggingFace
2 DATASET_NAME = "squad_v2"
3 raw_datasets = load_dataset(DATASET_NAME)

```

- Load tokenizer and run some examples:

```

1 # Load tokenizer để run một số example
2 tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

```

## Tokenize Dataset

Trong phần này ta sẽ tiến hành tokenize dataset cho tập train và tập val.

- **Tokenize train set:** Hàm preprocess\_training\_examples nhận dữ liệu đào tạo làm đầu vào và tiền xử lý để chuẩn bị cho việc huấn luyện mô hình hỏi đáp. Trong quá trình này, hàm trích xuất danh sách câu hỏi, mã hóa thông tin đầu vào bằng tokenizer, và trích xuất offset\_mapping và sample\_map để ánh xạ vị trí từ mã hóa về văn bản gốc. Hàm cũng xác định vị trí bắt đầu và kết thúc của câu trả lời trong ngữ cảnh và thêm thông tin về vị trí này vào biến inputs.

```

1 # Định nghĩa hàm preprocess_training_examples và nhận tham số
 # là dữ liệu training
2 def preprocess_training_examples(examples):
3 # Trích xuất danh sách câu hỏi từ examples và
4 # loại bỏ các khoảng trắng dư thừa
5 questions = [q.strip() for q in examples["question"]]
6
7 # Tiến hành mã hóa thông tin đầu vào sử dụng tokenizer
8 inputs = tokenizer(
9 questions,
10 examples["context"],
11 max_length=MAX_LENGTH,
12 truncation="only_second",
13 stride=STRIDE,
14 return_overflowing_tokens=True,
15 return_offsets_mapping=True,
16 padding="max_length",
17)

```

```
18)
19
20 # Trích xuất offset_mapping từ inputs và loại bỏ nó ra khỏi
21 # inputs
22 offset_mapping = inputs.pop("offset_mapping")
23
24 # Trích xuất sample_map từ inputs và loại bỏ nó ra khỏi inputs
25 sample_map = inputs.pop("overflow_to_sample_mapping")
26
27 # Trích xuất thông tin về câu trả lời (answers) từ examples
28 answers = examples["answers"]
29
30 # Khởi tạo danh sách các vị trí bắt đầu và kết thúc câu trả lời
31 start_positions = []
32 end_positions = []
33
34 # Duyệt qua danh sách offset_mapping
35 for i, offset in enumerate(offset_mapping):
36 # Xác định index của mẫu (sample) liên quan đến offset hiện
37 # tại
38 sample_idx = sample_map[i]
39
40 # Trích xuất sequence_ids từ inputs
41 sequence_ids = inputs.sequence_ids(i)
42
43 # Xác định vị trí bắt đầu và kết thúc của ngữ cảnh
44 idx = 0
45 while sequence_ids[idx] != 1:
46 idx += 1
47 context_start = idx
48 while sequence_ids[idx] == 1:
49 idx += 1
50 context_end = idx - 1
51
52 # Trích xuất thông tin về câu trả lời cho mẫu này
53 answer = answers[sample_idx]
54
55 if len(answer["text"]) == 0:
56 start_positions.append(0)
57 end_positions.append(0)
58 else:
59 # Xác định vị trí ký tự bắt đầu và kết thúc của câu trả
60 # lời
61 # trong ngữ cảnh
62 start_char = answer["answer_start"][0]
```

```

60 end_char = answer["answer_start"][0] + len(answer["text"]
61][0])
62
63 # Nếu câu trả lời không nằm hoàn toàn trong ngữ cảnh,
64 # gán nhãn là (0, 0)
65 if offset[context_start][0] > start_char
66 or offset[context_end][1] < end_char:
67 start_positions.append(0)
68 end_positions.append(0)
69 else:
70 # Nếu không, gán vị trí bắt đầu và kết thúc dựa trên
71 # vị trí của các mã thông tin
72 idx = context_start
73 while idx <= context_end and offset[idx][0] <=
74 start_char:
75 idx += 1
76 start_positions.append(idx - 1)
77
78 idx = context_end
79 while idx >= context_start and offset[idx][1] >=
80 end_char:
81 idx -= 1
82 end_positions.append(idx + 1)
83
84 # Thêm thông tin vị trí bắt đầu và kết thúc vào inputs
85 inputs["start_positions"] = start_positions
86 inputs["end_positions"] = end_positions
87
88 return inputs

```

Sau đó ta sẽ chạy đoạn hàm trên với từng dòng trong raw\_dataset của tập train:

```

1 # Tạo một biến train_dataset và gán cho nó giá trị sau khi áp dụng
2 # hàm preprocess_training_examples lên tập dữ liệu "train"
3 #
4 # Bật chế độ xử lý theo từng batch bằng cách đặt batched=True
5 #
6 # Loại bỏ các cột không cần thiết trong
7 # tập dữ liệu "train" bằng cách sử dụng remove_columns
8
9 train_dataset = raw_datasets["train"].map(
10 preprocess_training_examples,
11 batched=True,
12 remove_columns=raw_datasets["train"].column_names,

```

```

13)
14
15 # In ra độ dài của tập dữ liệu "train" ban đầu và
16 # độ dài của tập dữ liệu đã được xử lý (train_dataset)
17 len(raw_datasets["train"]), len(train_dataset)

```

- Tokenize val set:** Ta sẽ làm tương tự với tập val, hàm preprocess\_validation\_examples thực hiện việc tiền xử lý dữ liệu cho quá trình đánh giá mô hình. Hàm chuẩn bị danh sách câu hỏi, mã hóa các câu hỏi và văn bản liên quan bằng cách sử dụng tokenizer. Sau đó xác định ví dụ tham chiếu cho từng dòng đầu vào và điều chỉnh ánh xạ offset để loại bỏ các offset không phù hợp với sequence\_ids. Cuối cùng là thêm thông tin về ví dụ tham chiếu vào đầu vào.

```

1 def preprocess_validation_examples(examples):
2 # Chuẩn bị danh sách câu hỏi bằng cách
3 # loại bỏ khoảng trắng ở đầu và cuối mỗi câu hỏi
4 questions = [q.strip() for q in examples["question"]]
5
6 # Sử dụng tokenizer để mã hóa các câu hỏi và văn bản liên quan
7 inputs = tokenizer(
8 questions,
9 examples["context"],
10 max_length=MAX_LENGTH,
11 truncation="only_second",
12 stride=STRIDE,
13 return_overflowing_tokens=True,
14 return_offsets_mapping=True,
15 padding="max_length",
16)
17
18 # Lấy ánh xạ để ánh xạ lại ví dụ tham chiếu cho từng dòng trong
19 # inputs
20 sample_map = inputs.pop("overflow_to_sample_mapping")
21 example_ids = []
22
23 # Xác định ví dụ tham chiếu cho mỗi dòng đầu vào và
24 # điều chỉnh ánh xạ offset
25 for i in range(len(inputs["input_ids"])):
26 sample_idx = sample_map[i]
27 example_ids.append(examples["id"][sample_idx])
28
29 sequence_ids = inputs.sequence_ids(i)
30 offset = inputs["offset_mapping"][i]

```

```

30
31 # Loại bỏ các offset không phù hợp với sequence_ids
32 inputs["offset_mapping"] [i] = [
33 o if sequence_ids[k] == 1 else None \
34 for k, o in enumerate(offset)
35]
36
37 # Thêm thông tin ví dụ tham chiếu vào đầu vào
38 inputs["example_id"] = example_ids
39
40 return inputs

```

Ta sẽ chạy đoạn hàm trên với từng dòng trong raw\_dataset của tập validation:

```

1 # Tạo một biến validation_dataset và gán giá trị bằng việc sử dụng dữ liệu
2 # từ raw_datasets["validation"] sau khi áp dụng một hàm xử lý trước.
3
4 validation_dataset = raw_datasets["validation"].map(
5 preprocess_validation_examples,
6 batched=True,
7 remove_columns=raw_datasets["validation"].column_names,
8)
9
10 # In ra độ dài của raw_datasets["validation"]
11 # và validation_dataset để so sánh.
12 len(raw_datasets["validation"]), len(validation_dataset)

```

## Train model

Sau khi đã chuẩn bị xong dataset, ta sẽ tiến hành load model từ HuggingFace để chuẩn bị cho quá trình training:

```

1 # Load model
2 model = AutoModelForQuestionAnswering.from_pretrained(MODEL_NAME)

```

Tiếp theo ta sẽ định nghĩa một số parameter mà ta sẽ sử dụng để training model:

```

1 # Tạo đối tượng args là các tham số cho quá trình huấn luyện
2 args = TrainingArguments(
3 output_dir="distilbert-finetuned-squadv2", # Thư mục lưu output

```

```
4 evaluation_strategy="no", # Chế độ đánh giá không tự động sau mỗi
 epoch
5 save_strategy="epoch", # Lưu checkpoint sau mỗi epoch
6 learning_rate=2e-5, # Tốc độ học
7 num_train_epochs=3, # Số epoch huấn luyện
8 weight_decay=0.01, # Giảm trọng lượng mô hình để tránh overfitting
9 fp16=True, # Sử dụng kiểu dữ liệu half-precision để tối ưu tài nguyên
10 push_to_hub=True, # Đẩy kết quả huấn luyện lên HuggingFace Hub
11)
```

Cuối cùng ta sẽ khởi tạo class Trainer, đây là class chính để training model, ta sẽ không cần phải định nghĩa hàm train, đưa input vào mode, tính toán loss, update gradient nữa, hàm class này sẽ tự động làm giúp chúng ta. Sau khi đã khởi tạo thì chỉ cần gọi `trainer.train()` thì quá trình training model sẽ được tiến hành:

```
1 # Khởi tạo một đối tượng Trainer để huấn luyện mô hình
2 trainer = Trainer(
3 model=model, # Sử dụng mô hình đã tạo trước đó
4 args=args, # Các tham số và cấu hình huấn luyện
5 train_dataset=train_dataset, # Sử dụng tập dữ liệu huấn luyện
6 eval_dataset=validation_dataset, # Sử dụng tập dữ liệu đánh giá
7 tokenizer=tokenizer, # Sử dụng tokenizer để xử lý văn bản
8)
9
10 # Bắt đầu quá trình huấn luyện
11 trainer.train()
```

| [49354/49410 1:39:37 < 00:06, 8.26 it/s, Epoch 3.00/3] |               |       |          |
|--------------------------------------------------------|---------------|-------|----------|
| Step                                                   | Training Loss |       |          |
| 500                                                    | 3.102900      | 45500 | 0.686000 |
| 1000                                                   | 2.268800      | 46000 | 0.696700 |
| 1500                                                   | 1.971300      | 46500 | 0.674500 |
| 2000                                                   | 1.810200      | 47000 | 0.651100 |
| 2500                                                   | 1.702800      | 47500 | 0.676700 |
| 3000                                                   | 1.625000      | 48000 | 0.651300 |
| 3500                                                   | 1.574800      | 48500 | 0.673300 |
| 4000                                                   | 1.566400      | 49000 | 0.697000 |
| 4500                                                   | 1.495000      |       |          |
| 5000                                                   | 1.480800      |       |          |
| 5500                                                   | 1.438300      |       |          |
| 6000                                                   | 1.402200      |       |          |
| 6500                                                   | 1.418000      |       |          |
| 7000                                                   | 1.426900      |       |          |

Hình 45.4: Kết quả training loss tại mỗi 500 step.

Sau khi quá trình training hoàn tất, ta sẽ đưa weight, config của model lên HuggingFace Hub để lưu lại:

```

1 # Gửi dữ liệu đào tạo lên Hub
2 trainer.push_to_hub(commit_message="Training complete")

```

### Evaluate model

Để đánh giá performance của model ta sẽ sử dụng metric squad từ thư viện evaluate:

```

1 # Tải metric "squad" từ thư viện evaluate
2 metric = evaluate.load("squad_v2")

```

Hàm `compute_metrics()` nhận các đầu vào như `start_logits`, `end_logits`, `features`, và `examples`, và thực hiện các bước sau để tính toán các độ đo và kết quả đánh giá mô hình hỏi đáp. Trong quá trình tính toán, hàm này tạo

một danh sách các câu trả lời dự đoán dựa trên các logits được dự đoán bởi mô hình. Điều này bao gồm việc xác định vị trí bắt đầu và kết thúc tốt nhất cho các câu trả lời và đánh giá xem chúng có hợp lệ hay không dựa trên độ dài tối đa cho câu trả lời. Cuối cùng, hàm tính toán các độ đo và trả về kết quả đánh giá mô hình hỏi đáp dựa trên các câu trả lời dự đoán và câu trả lời lý thuyết từ ví dụ.

```
1 N_BEST = 20 # Số lượng kết quả tốt nhất được lựa chọn sau khi dự đoán
2 MAX_ANS_LENGTH = 30 # Độ dài tối đa cho câu trả lời dự đoán
3
4 def compute_metrics(start_logits, end_logits, features, examples):
5 # Tạo một từ điển mặc định để ánh xạ mỗi ví dụ
6 # với danh sách các đặc trưng tương ứng
7 example_to_features = collections.defaultdict(list)
8 for idx, feature in enumerate(features):
9 example_to_features[feature["example_id"]].append(idx)
10
11 predicted_answers = []
12 for example in tqdm(examples):
13 example_id = example["id"]
14 context = example["context"]
15 answers = []
16
17 # Lặp qua tất cả các đặc trưng liên quan đến ví dụ đó
18 for feature_index in example_to_features[example_id]:
19 start_logit = start_logits[feature_index]
20 end_logit = end_logits[feature_index]
21 offsets = features[feature_index]["offset_mapping"]
22
23 # Lấy các chỉ số có giá trị lớn nhất cho start và end logits
24 start_indexes = np.argsort(start_logit)[-1:-N_BEST-1:-1].
25 tolist()
26 end_indexes = np.argsort(end_logit)[-1:-N_BEST-1:-1].tolist()
27 for start_index in start_indexes:
28 for end_index in end_indexes:
29 # Bỏ qua các câu trả lời
30 # không hoàn toàn nằm trong ngữ cảnh
31 if offsets[start_index] is None or \
32 offsets[end_index] is None:
33 continue
34 # Bỏ qua các câu trả lời có độ dài > max_answer_length
35 if end_index - start_index + 1 > MAX_ANS_LENGTH:
36 continue
37
38 # Tạo một câu trả lời mới
```

```

38 text = context[
39 offsets[start_index][0]:offsets[end_index][1]
40]
41 logit_score = start_logit[start_index] + \
42 end_logit[end_index]
43 answer = {
44 "text": text,
45 "logit_score": logit_score,
46 }
47 answers.append(answer)
48
49 # Chọn câu trả lời có điểm số tốt nhất
50 if len(answers) > 0:
51 best_answer = max(answers, key=lambda x: x["logit_score"])
52 answer_dict = {
53 "id": example_id,
54 "prediction_text": best_answer["text"],
55 "no_answer_probability": 1 - best_answer["logit_score"]
56 }
57 else:
58 answer_dict = {
59 "id": example_id,
60 "prediction_text": "",
61 "no_answer_probability": 1.0
62 }
63 predicted_answers.append(answer_dict)
64
65 # Tạo danh sách câu trả lời lý thuyết từ các ví dụ
66 theoretical_answers = [
67 {"id": ex["id"], "answers": ex["answers"]} for ex in examples
68]
69 # Sử dụng metric.compute để tính toán các độ đo và trả về kết quả
70 return metric.compute(
71 predictions=predicted_answers,
72 references=theoretical_answers
73)

```

Sau khi đã định nghĩa hàm evaluation, ta sẽ tiến hành predict model trên tập validation rồi đưa vào hàm compute\_metrics:

```

1 # Thực hiện dự đoán trên tập dữ liệu validation
2 predictions, _, _ = trainer.predict(validation_dataset)
3
4 # Lấy ra thông tin về các điểm bắt đầu và
5 # điểm kết thúc của câu trả lời dự đoán

```

```

6 start_logits, end_logits = predictions
7
8 # Tính toán các chỉ số đánh giá sử dụng hàm compute_metrics
9 results = compute_metrics(
10 start_logits,
11 end_logits,
12 validation_dataset,
13 raw_datasets["validation"]
14)
15 results

```

100%  11873/11873 [00:19<00:00, 686.51it/s]

```

{'exact': 47.452202476206516,
 'f1': 51.28265600122848,
 'total': 11873,
 'HasAns_exact': 74.78070175438596,
 'HasAns_f1': 82.45259357331055,
 'HasAns_total': 5928,
 'NoAns_exact': 20.201850294365013,
 'NoAns_f1': 20.201850294365013,
 'NoAns_total': 5945,
 'best_exact': 64.46559420533984,
 'best_exact_thresh': -11.35546875,
 'best_f1': 66.16170764530801,
 'best_f1_thresh': -9.61328125}

```

## Load model from hub

Ở phần trước, xong khi training model xong thì ta đã đưa model lên HuggingFace, nếu muốn sử dụng thì ta chỉ cần dùng class pipeline có sẵn của HuggingFace là đã có thể load model và tiến hành inference:

```

1 # Use a pipeline as a high-level helper
2 from transformers import pipeline
3
4 PIPELINE_NAME = "question-answering"
5 MODEL_NAME = "thangduong0509/distilbert-finetuned-squadv2"
6 pipe = pipeline(PIPELINE_NAME, model=MODEL_NAME)

```

Sau đây ta sẽ chạy thử một example để test model:

```

1 INPUT_QUESTION = "What is my name?"
2 INPUT_CONTEXT = "My name is AI Vietnam and I live in Vietnam."
3 pipe(question=INPUT_QUESTION, context=INPUT_CONTEXT)

```

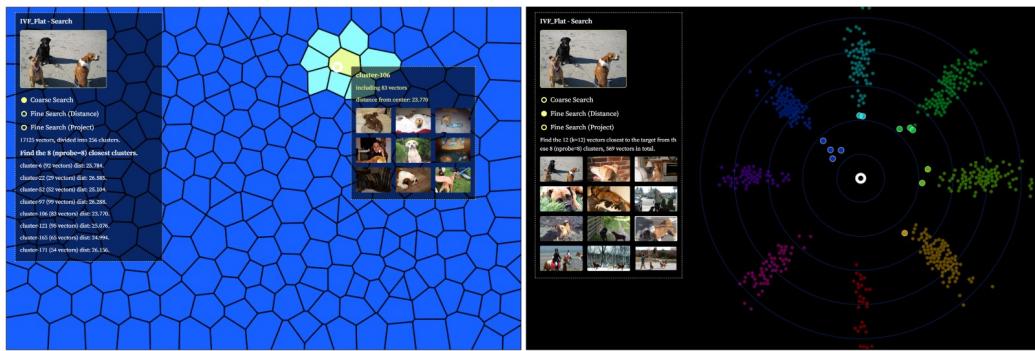
```

4
5 ## >> Output: {"score": 0.97179114818573, "start": 11, "end": 21, "answer"
 ": "AI Vietnam"}

```

### 45.2.3 Retriever: Faiss

Faiss (Facebook AI Similarity Search) là một thư viện được phát triển bởi Facebook AI Research Team, hỗ trợ trong việc tìm kiếm tương đồng và phân cụm (clustering) các vector với tốc độ và độ chính xác cao. Các bạn có thể đọc thêm về Faiss tại [đây](#).

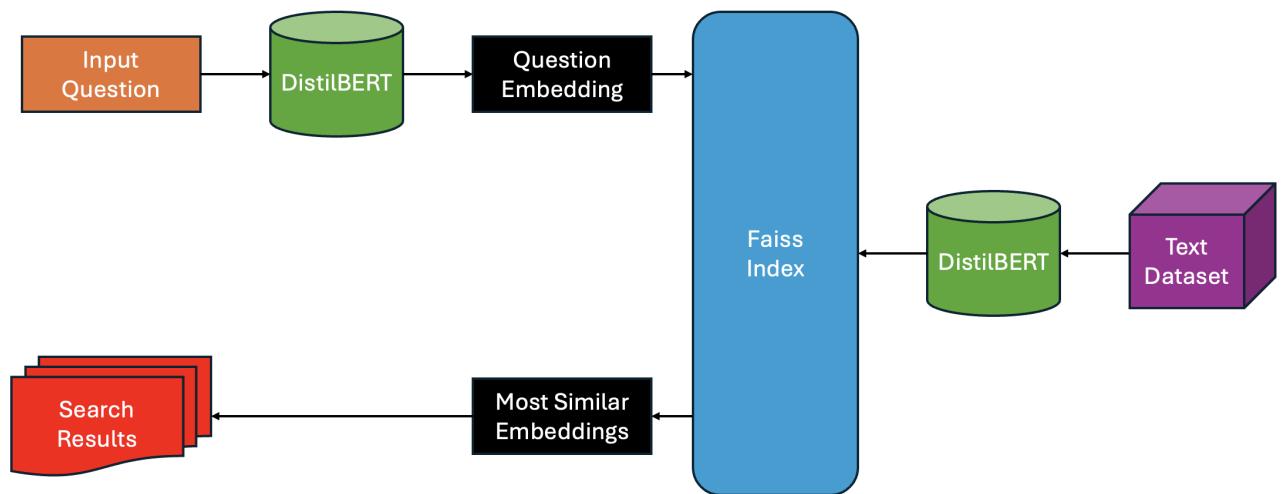


Hình 45.5: Minh họa phương pháp lập chỉ mục (indexing) IVF\_Flat trong thư viện Faiss. [Source](#).

Tại đây, chúng ta sẽ ứng dụng Faiss để làm module Retriever cho hệ thống E2E QA của chúng ta. Với nhiệm vụ tìm kiếm các context phù hợp nhất cho câu hỏi đầu vào, ta sẽ cài đặt Faiss theo cách thức như sau:

1. Với bộ dữ liệu SQuAD2.0, ta sẽ xây dựng một database chứa thêm cột đại diện cho vector embedding của câu hỏi.
2. Thực hiện embedding các câu hỏi sử dụng DistilBERT.
3. Thực hiện tìm kiếm tương đồng giữa các vector trong cột mới và vector câu hỏi đầu vào, từ đó tìm ra nội dung context có liên quan nhất.

Quy trình xử lý của Faiss trong bài có thể được tóm gọn qua ảnh sau:



Hình 45.6: Minh họa các bước xây dựng một vector database với Faiss.

Để cài đặt Faiss phục vụ cho việc tìm kiếm các văn bản context của các câu hỏi có nội dung giống với câu hỏi đầu vào, ta thực hiện như sau:

### Cài đặt và import các thư viện cần thiết

```

1 !pip install -qq transformers[sentencepiece]==4.35.2 datasets==2.16.1
 evaluate==0.4.1
2 !sudo apt-get install libomp-dev
3 !pip install -qq faiss-gpu

```

```

1 import numpy as np
2 import collections
3 import torch
4 import faiss
5 import evaluate
6
7 from datasets import load_dataset
8 from transformers import AutoTokenizer, AutoModel
9 from transformers import AutoModelForQuestionAnswering
10 from transformers import TrainingArguments
11 from transformers import Trainer
12 from tqdm.auto import tqdm

```

```

13 device = torch.device("cuda") if torch.cuda.is_available() else torch.
14 device("cpu")

```

## Tải bộ dữ liệu

Ta tải bộ dữ liệu SQuAD2.0:

```

1 DATASET_NAME = "squad_v2"
2 raw_datasets = load_dataset(DATASET_NAME, split="train+validation")
3 raw_datasets

```

Để tận dụng toàn bộ ngữ liệu, chúng ta sẽ gom hai bộ dữ liệu train và validation trong bước tạo vector database này.

## Loại bỏ các mẫu không có đáp án

Các mẫu dữ liệu không có đáp án thường chứa các câu hỏi không liên quan đến đoạn văn ngữ cảnh. Vì vậy, ta sẽ loại bỏ các trường hợp này ra khỏi bộ dữ liệu:

```

1 raw_datasets = raw_datasets.filter(
2 lambda x: len(x["answers"]["text"]) > 0
3)

```

## Khởi tạo mô hình

Để tạo vector embedding cho các câu hỏi, ta sẽ sử dụng mô hình pre-trained DistilBERT:

```

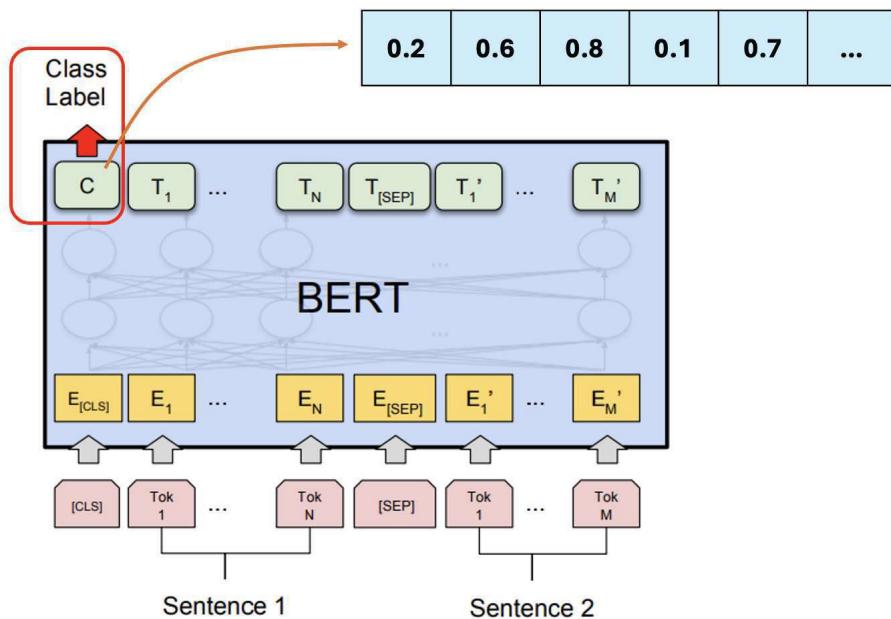
1 MODEL_NAME = "distilbert-base-uncased"
2 tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
3 model = AutoModel.from_pretrained(MODEL_NAME).to(device)

```

## Xây dựng hàm lấy vector embedding:

Ở đây, ta sẽ xây dựng một hàm trả về vector embedding cho một đoạn text đầu vào, cụ thể ở đây sẽ là câu hỏi. Để tạo vector embedding đại diện cho câu

hỏi, ta sẽ sử dụng vector hidden state từ token [CLS] trong kết quả output của mô hình DistilBERT:



Hình 45.7: Ảnh minh họa vị trí của final hidden state của token CLS.

Đầu tiên, ta xây dựng hàm lấy final hidden state của token CLS:

```
1 def cls_pooling(model_output):
2 return model_output.last_hidden_state[:, 0]
```

Sau đó, xây dựng hàm đưa một văn bản vào model để từ đó có thể gọi hàm `cls_pooling()`:

```
1 def get_embeddings(text_list):
2 encoded_input = tokenizer(
3 text_list,
4 padding=True,
5 truncation=True,
6 return_tensors="pt"
7)
8 encoded_input = {k: v.to(device) for k, v in encoded_input.items()}
9 model_output = model(**encoded_input)
10
```

```
11 | return cls_pooling(model_output)
```

## Xây dựng vector database

Với hàm tạo vector embedding đã triển khai, ta sẽ sử dụng nó để tạo một cột trong bảng dữ liệu dùng để chứa kết quả lời gọi hàm `get_embeddings()` với input là các câu hỏi của từng mẫu dữ liệu. Cụ thể, ta tạo cột mới tên là `question_embedding` và lưu vector embedding của câu hỏi như sau:

```
1 EMBEDDING_COLUMN = "question_embedding"
2 embeddings_dataset = raw_datasets.map(
3 lambda x: {
4 EMBEDDING_COLUMN: get_embeddings(
5 x["question"]
6).detach().cpu().numpy()[0]
7 }
8)
```

Sau đó, để có thể tìm kiếm các vector tương đồng, ta sẽ tạo một cấu trúc dữ liệu đặc biệt là Faiss Index như sau:

```
1 embeddings_dataset.add_faiss_index(column=EMBEDDING_COLUMN)
```

Cuối cùng, chúng ta đã có một vector database lưu trữ vector embedding của các câu hỏi trong bộ dữ liệu. Từ đây, ta sẽ tiến hành thử thực hiện truy vấn với một câu hỏi đầu vào bất kì như sau:

```
1 input_question = "When did Beyonce start becoming popular?"
2
3 input_quest_embedding = get_embeddings([input_question])
4 input_quest_embedding = input_quest_embedding.cpu().detach().numpy()
5
6 TOP_K = 5
7 scores, samples = embeddings_dataset.get_nearest_examples(
8 EMBEDDING_COLUMN, input_quest_embedding, k=TOP_K
9)
10
11 for idx, score in enumerate(scores):
12 print(f"Top {idx + 1}\tScore: {score}")
13 print(f"Question: {samples['question'][idx]}")
14 print(f"Context: {samples['context'][idx]}")
15 print()
```

Khi chạy xong đoạn lệnh trên, ta được kết quả trả về như sau:

```

Top 1 Score: 0.0
Question: When did Beyonce start becoming popular?
Context: Beyoncé Giselle Knowles-Carter (/bi:'jɒnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter,
Top 2 Score: 2.6135313510894775
Question: When did Beyoncé rise to fame?
Context: Beyoncé Giselle Knowles-Carter (/bi:'jɒnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter,
Top 3 Score: 4.859482288360596
Question: When did Beyoncé release Formation?
Context: On February 6, 2016, one day before her performance at the Super Bowl, Beyoncé released a new single exclusively on m
Top 4 Score: 5.054221153259277
Question: In which decade did Beyonce become famous?
Context: Beyoncé Giselle Knowles-Carter (/bi:'jɒnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter,
Top 5 Score: 5.170375347137451
Question: When did Beyonce begin her deals with name brands?
Context: The release of a video-game Starpower: Beyoncé was cancelled after Beyoncé pulled out of a $100 million with GateFive

```

Hình 45.8: Kết quả truy vấn được in ra màn hình.

Có thể thấy, vì câu hỏi đầu vào có tồn tại trong vector database của chúng ta nên mẫu dữ liệu tương đồng nhất cũng chính là mẫu có câu hỏi tương tự.

#### 45.2.4 Áp dụng mô hình hỏi-đáp để trả lời câu hỏi

Như vậy, chúng ta đã có hai thành phần Retriever và Reader. Chúng ta sẽ viết một đoạn code ngắn để thực hiện chương trình End-to-End QA. Đầu tiên, khởi tạo mô hình hỏi-đáp đã huấn luyện:

```

1 from transformers import pipeline
2
3 PIPELINE_NAME = "question-answering"
4 MODEL_NAME = "thangduong0509/distilbert-finetuned-squadv2"
5 pipe = pipeline(PIPELINE_NAME, model=MODEL_NAME)

```

Tận dụng kết quả truy vấn vừa rồi (nằm ở biến `scores` và `samples`), chúng ta sẽ truyền vào mô hình QA hai thông tin gồm question và context:

```

1 print(f"Input question: {input_question}")
2 for idx, score in enumerate(scores):
3 question = samples["question"][idx]
4 context = samples["context"][idx]
5 answer = pipe(
6 question=question,

```

```

7 context=context
8)
9 print(f"Top {idx + 1}\tScore: {score}")
10 print(f"Context: {context}")
11 print(f"Answer: {answer}")
12 print()

```

```

Input question: When did Beyonce start becoming popular?
Top 1 Score: 0.0
Context: Beyoncé Giselle Knowles-Carter (/bi:'jnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter,
Answer: {'score': 0.6091989278793335, 'start': 276, 'end': 286, 'answer': 'late 1990s'}

Top 2 Score: 2.6135313510894775
Context: Beyoncé Giselle Knowles-Carter (/bi:'jnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter,
Answer: {'score': 0.6103343963623047, 'start': 276, 'end': 286, 'answer': 'late 1990s'}

Top 3 Score: 4.859482288360596
Context: On February 6, 2016, one day before her performance at the Super Bowl, Beyoncé released a new single exclusively on m
Answer: {'score': 0.9715896248817444, 'start': 3, 'end': 19, 'answer': 'February 6, 2016'}

Top 4 Score: 5.054221153259277
Context: Beyoncé Giselle Knowles-Carter (/bi:'jnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter,
Answer: {'score': 0.7218101620674133, 'start': 281, 'end': 286, 'answer': '1990s'}

Top 5 Score: 5.170375347137451
Context: The release of a video-game Starpower: Beyoncé was cancelled after Beyoncé pulled out of a $100 million with GateFive
Answer: {'score': 0.6330415606498718, 'start': 433, 'end': 452, 'answer': 'since the age of 18'}

```

Hình 45.9: Kết quả E2E QA được in ra màn hình.

Với mô hình xây dựng được, ta có thể triển khai thành một ứng dụng web demo chương trình cho phép trả lời những câu hỏi xoay quanh một tài liệu PDF mà chúng ta đưa vào. Thông tin về web demo có thể được tìm thấy tại phần 60.4



## Retrieval-Augmented Generation (RAG) Demo

Enter your Hugging Face API Key

.....

Don't have an API key? Get one [here](#) ([Read Token](#) is enough)

Upload documents



Drag and drop files here

Limit 200MB per file

DeepSeek\_V3.pdf 1.6MB

Uploaded 1/1 files

Indexing 1 documents

Querying with 70 nodes

Enter your query for RAG

How many tokens was DeepSeek-V3 pre-trained on, and how many GPU hours did it cost?

Hình 45.10: Demo Retrieval-augmented Generation (RAG) (1).

Enter your query for RAG

How many tokens was DeepSeek-V3 pre-trained on, and how many GPU hours did it cost?

### Retrieved Documents

Document 1 (Score: 0.8498)

Training Costs Pre-Training Context Extension Post-Training Total in H800 GPU Hours 2664K 119K 5K 2788K in USD 5.328M 0.238M 0.01M 5.576M Table 1 | Training costs of DeepSeek-V3, assuming the rental price of H800 is 2 per GPU hour, and generation length. We evaluate DeepSeek – V3 on a comprehensive array of benchmarks. Despite its economical training costs, comprehensive evaluations reveal that DeepSeek – V3 – Base has emerged as the strongest open – source base model currently available, especially in code and math. Its chat version also outperforms other open – source models and achieves comparable to leading closed – source models, including GPT – 4 and Claude – 3.5 – Sonnet, on a series of standard and open – ended benchmarks. Lastly, we emphasize again the economical training costs of DeepSeek – V3, summarized in Table 1, achieved through hour optimized design of algorithms, frameworks, and hardware. During the pre – training stage, training DeepSeek – V3 on each trillion tokens requires only 180K H800 GPU hours, i.e., 3.7 days on our cluster with 2048 H800 GPUs. Consequently, our pre – training stage is completed in less than two months and costs 2664K GPU hours. Combined with 119K GPU hours for the context length extension and 5K GPU hours for post – training, DeepSeek – V3 costs only 2.788M GPU hours for its full training. Assuming the rental price of the H800 GPU is \$2 per GPU hour, our total training costs amount to only 5.576M. Note that the aforementioned costs include only the official training of DeepSeek-V3, excluding the costs associated with prior research and ablation experiments on architectures, algorithms, or data. Our main contribution includes: Architecture: Innovative Load Balancing Strategy and Training Objective • On top of the efficient architecture of DeepSeek-V2, we pioneer an auxiliary-loss-free strategy for load balancing, which minimizes the performance degradation that arises from encouraging load balancing.

Document 2 (Score: 0.7705)

Document 3 (Score: 0.7564)

### RAG Response:

DeepSeek-V3 was pre-trained on 14.8 trillion tokens, and it cost 2.664M H800 GPU hours. The Multi-Token Prediction (MTP) objective used in DeepSeek-V3 extends the prediction scope to multiple future tokens at each position, which densifies the training signals and may improve data efficiency. The MTP implementation consists of D sequential modules to predict D additional tokens, with each module having a shared embedding layer, a shared output head, a Transformer block, and a projection matrix.

Hình 45.11: Demo Retrieval-augmented Generation (RAG) (2).

### 45.3 Câu hỏi trắc nghiệm

1. So với QA, chương trình End-to-end QA có điểm gì khác biệt?
  - (a) Mô hình trích xuất câu hỏi tốt hơn.
  - (b) Sử dụng kiến trúc transformer.
  - (c) Có sử dụng mô hình tìm kiếm context.
  - (d) Tốc độ xử lý nhanh hơn.
2. Tại sao mô hình Transformer được sử dụng phổ biến trong bài toán Question Answering (QA)?
  - (a) Do Transformer không có khả năng tự học đặc trưng từ văn bản tự nhiên.
  - (b) Do Transformer chứa nhiều kiến thức về dữ liệu.
  - (c) Có sử dụng mô hình tìm kiếm context.
  - (d) Do Transformer có khả năng xử lý dữ liệu dạng sequence.
3. Trong QA, tại sao phải sử dụng Transfer learning?
  - (a) Transfer learning giúp mô hình học được kiến thức từ dữ liệu lớn.
  - (b) Mô hình QA không cần sử dụng transfer learning.
  - (c) Transfer learning làm tăng khả năng xử lý của CPU.
  - (d) Transfer learning giúp mô hình bị overfitting.
4. Mô hình End-to-end QA khác biệt với QA truyền thống ở điểm nào?
  - (a) Mô hình End-to-end QA sử dụng mô hình tìm kiếm context.
  - (b) Mô hình End-to-end QA có khả năng tự động xây dựng câu hỏi.
  - (c) Mô hình End-to-end QA sử dụng RNN để trích xuất câu trả lời.
  - (d) Mô hình End-to-end QA chỉ hoạt động trên dữ liệu có cấu trúc.
5. Tham số **stride** có nghĩa là gì trong đoạn code sau:

```

1 inputs = tokenizer(
2 question, # Danh sách các câu hỏi
3 context, # Nội dung liên quan đến câu hỏi
4 max_length=MAX_LENGTH, # Độ dài tối đa cho đầu ra mã hóa
5 truncation="only_second", # Cắt bớt dữ liệu chỉ cho phần thứ
6 # hai (context)
7 stride=STRIDE,
8 return_overflowing_tokens=True, # Trả về các tokens vượt quá độ
9 # dài tối đa
10 return_offsets_mapping=True, # Trả về bản đồ vị trí của các
11 # tokens trong văn bản gốc
12 padding="max_length" # Điện vào dữ liệu để có cùng độ dài
13 # max_length
14)

```

- (a) Độ dài bước nhảy trong trường hợp dữ liệu dài hơn max\_length.
- (b) Độ dài bước nhảy trong trường hợp dữ liệu ngắn hơn max\_length.
- (c) Độ dài bước nhảy trong trường hợp dữ liệu bằng max\_length.
- (d) Độ dài bước nhảy trong trường hợp bất kỳ.
6. Tham số fp16=True có nghĩa là gì trong đoạn code sau:

```

1 # Tạo đối tượng args là các tham số cho quá trình huấn luyện
2 args = TrainingArguments(
3 output_dir="distilbert-finetuned-squadv2", # Thư mục lưu trữ kết
4 # quả huấn luyện
5 evaluation_strategy="no", # Chế độ đánh giá không tự động sau mỗi
6 # epoch
7 save_strategy="epoch", # Lưu checkpoint sau mỗi epoch
8 learning_rate=2e-5, # Tốc độ học
9 num_train_epochs=3, # Số epoch huấn luyện
10 weight_decay=0.01, # Giảm trọng lượng mô hình để tránh
11 # overfitting
12 fp16=True,
13 push_to_hub=True, # Đẩy kết quả huấn luyện lên một nơi chia sẻ
14 # trực tuyến (Hub)
15)

```

- (a) Sử dụng kiểu dữ liệu 32-bit để tối ưu hóa tài nguyên.
- (b) Sử dụng kiểu dữ liệu double để tối ưu hóa tài nguyên.

- (c) Sử dụng kiểu dữ liệu float để tối ưu hóa tài nguyên.  
(d) Sử dụng kiểu dữ liệu half-precision để tối ưu hóa tài nguyên.

7. Trong đoạn code sau đây, biến PIPELINE\_NAME dùng để làm gì?

```
1 # Use a pipeline as a high-level helper
2 from transformers import pipeline
3
4 PIPELINE_NAME = "question-answering"
5 MODEL_NAME = "thangduong0509/distilbert-finetuned-squadv2"
6 pipe = pipeline(PIPELINE_NAME, model=MODEL_NAME)
```

- (a) Xác định tên của task hiện tại, người dùng có thể đặt tên bất kỳ.  
(b) Xác định tên của task hiện tại, người dùng phải đặt đúng tên quy định của HuggingFace.  
(c) Tên của model, người dùng phải đặt tên đúng yêu cầu.  
(d) Tên của model, người dùng có thể đặt bất kỳ.
8. Ưu điểm của vector database khi xử lý các loại dữ liệu phức tạp như hình ảnh, âm thanh và văn bản so với cơ sở dữ liệu quan hệ truyền thống là gì?
- (a) Nó cung cấp khả năng chuẩn hóa dữ liệu và ràng buộc tính toàn vẹn tốt hơn.  
(b) Nó cho phép tìm kiếm và truy vấn dữ liệu dựa trên nội dung một cách hiệu quả.  
(c) Nó cung cấp các cơ chế kiểm soát giao dịch mạnh mẽ hơn.  
(d) Nó tăng cường khả năng truy vấn SQL cho phân tích dữ liệu có cấu trúc.
9. Để tính sự tương đồng giữa hai vector, độ đo nào sau đây không thể áp dụng?
- (a) Cosine Similarity.  
(b) Euclidean Distance.  
(c) Pearson Correlation Coefficient.  
(d) Maximum Likelihood Estimation.

10. Khi ứng dụng BERT để tạo vector embedding trong vector database, final hidden state của token nào thường được sử dụng trong output của BERT?
- (a) [CLS] token.
  - (b) [SEP] token.
  - (c) [EOS] token.
  - (d) Final token.
11. Trong các vector database như Faiss, kỹ thuật nào thường được sử dụng để tối ưu hiệu quả tìm kiếm trên dữ liệu đa chiều?
- (a) Linear Search.
  - (b) Indexing.
  - (c) Quantization.
  - (d) Encryption.
12. Trong đoạn code dưới đây:

```
1 TOP_K = 5
2 scores, samples = embeddings_dataset.get_nearest_examples
3 (
4 EMBEDDING_COLUMN, input_quest_embedding, k=TOP_K
5)
```

Biến TOP\_K còn được hiểu là?

- (a) Số lượng cluster.
- (b) Số lượng kết quả trả về.
- (c) Số epochs.
- (d) Số chiều trong không gian embedding.

## 45.4 Phụ lục

1. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Demo:** Web demo và mã nguồn có thể được truy cập tại [đây](#).
4. **Rubric:**

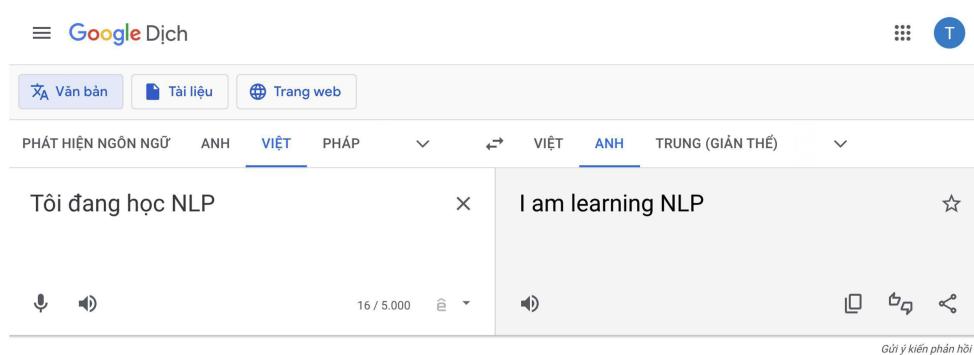
| Mục | Kiến Thức                                                                                                                                                                                                                                                                                                                                                                                         | Đánh Giá                                                                                                                                                                                                                                                                                                                                               |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| II. | <ul style="list-style-type: none"> <li>- Kiến thức về bài toán Question Answering.</li> <li>- Kiến thức về bài toán Text Retrieval.</li> <li>- Kiến thức về Open-domain Question Answering.</li> <li>- Kiến thức về cách xây dựng một hệ thống End-to-end Question Answering.</li> <li>- Cách cài đặt, xây dựng và huấn luyện mô hình QA.</li> <li>- Cách cài đặt một vector database.</li> </ul> | <ul style="list-style-type: none"> <li>- Nắm được các khái niệm về bài toán Question Answering, Text Retrieval và End-to-end Question Answering.</li> <li>- Biết cách cài đặt code xây dựng, huấn luyện mô hình QA sử dụng thư viện HuggingFace.</li> <li>- Biết cách cài đặt một vector database để thực hiện tìm kiếm văn bản tương đồng.</li> </ul> |

- *Hết* -

# Chương 46

## Text Generation và Machine Translation

### 46.1 Giới thiệu



Hình 46.1: Hệ thống dịch máy Google.

**Dịch Máy (Machine Translation)** với mục đích tự động dịch văn bản hoặc lời nói từ ngôn ngữ tự nhiên này sang ngôn ngữ tự nhiên khác. Dịch máy sử dụng kết hợp nhiều ý tưởng và các kỹ thuật với nhau từ ngôn ngữ học, khoa học máy tính, xác suất thống kê và trí tuệ nhân tạo. Với mục tiêu của dịch máy là phát triển một hệ thống cho phép tạo ra bản dịch chính xác giữa các ngôn ngữ tự nhiên của con người.

Các hệ thống dịch máy điển hình hiện nay như: Google Translate, Bing Translator,... đã đạt được chất lượng bản dịch tốt và được tích hợp trong nhiều nền tảng ứng dụng khác nhau và có thể dịch tốt giữa hơn 100 các ngôn ngữ tự nhiên.

**Như vậy, Input/Output của bài toán là:**

- **Input:** Văn bản đầu vào của ngôn ngữ nguồn.  
VD: Câu đầu vào tiếng việt: "Tôi đang học NLP"
- **Output:** Văn bản được dịch sang ngôn ngữ đích.  
VD: Câu được dịch sang tiếng anh: "I am learning NLP"

Mô hình hoá bài toán dịch máy, với mục tiêu là huấn luyện và tối ưu các tham số mô hình  $\theta$  với đầu vào văn bản từ ngôn ngữ nguồn  $w^{(s)}$  và đầu ra văn bản từ ngôn ngữ đích tương ứng  $w^{(t)}$ :

$$\hat{w}^{(t)} = \text{argmax}_{(w^{(t)})} \theta(w^{(s)}, w^{(t)})$$

Vì vậy, để giải quyết tốt bài toán dịch máy, chúng ta cần quan tâm tối ưu:

- **Phần 1:** Thuật toán học tối ưu bộ tham số  $\theta$
- **Phần 2:** Thuật toán giải mã (decoding) để sinh ra bản dịch tốt nhất cho văn bản đầu vào

Hiện nay, có ba hướng tiếp cận chính cho bài toán này:

- **Hướng 1:** Dịch máy dựa vào luật (Rule-based Machine Translation - RBMT)
- **Hướng 2:** Dịch máy dựa vào thống kê (Statistical Machine Translation - SMT)
- **Hướng 3:** Dịch máy dựa vào mạng nơ-ron (Neural Machine Translation - NMT)

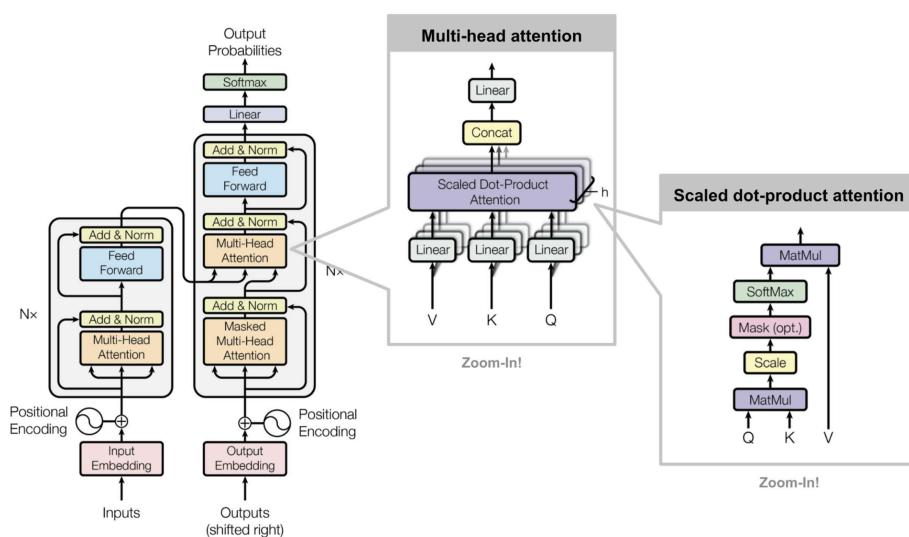
Trong các hướng tiếp cận này NMT đang ngày càng phát triển và cho chất lượng bản dịch tốt vượt trội. Vì vậy, phạm vi project tập trung vào các phương pháp dựa trên mạng nơ-ron gồm 2 nội dung chính:

- **Phương pháp 1:** Xây dựng mô hình dịch máy sử dụng kiến trúc Transformer
- **Phương pháp 2:** Xây dựng mô hình dịch máy sử dụng Pre-trained Language Model như BERT và GPT

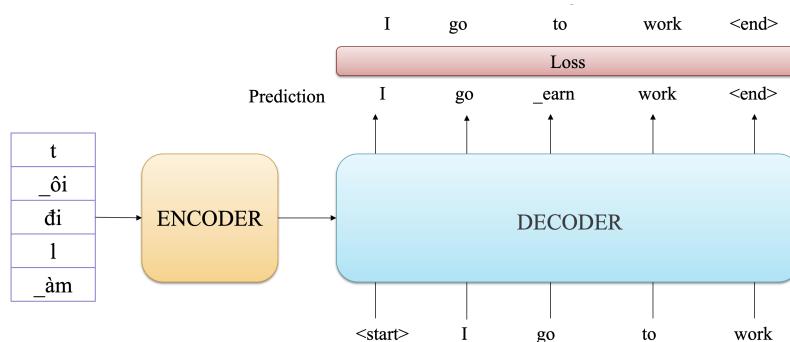
## 46.2 Transformer-based Translation

Để xây dựng, đánh giá hiệu suất các mô hình chúng ta sử dụng bộ dữ liệu dịch IWSLT'15 English - Vietnamese với số lượng mẫu cho training: 133,317 cặp câu song ngữ, tập validation: 1,553 cặp câu song ngữ và tập test: 1,269 cặp câu song ngữ. Chiều dịch sẽ từ tiếng anh sang tiếng việt.

Metric để đánh giá chúng ta sử dụng: ScacreBleu (BLEU)



Hình 46.2: Mô hình Transformer.



Hình 46.3: Dịch máy sử dụng mô hình Transformer.

### 1. Build Dataset

Trong phần này chúng ta sẽ tải về bộ dữ liệu dịch máy en-vi và xây dựng bộ từ điển thông qua tokenizers.

```

1 !pip install -q datasets
2
3 import os
4 from tokenizers import Tokenizer, pre_tokenizers, trainers,
5 models
6 from datasets import load_dataset
7
8 ds = load_dataset("thainq107/iwslt2015-en-vi")
9
10 # word-based
11 tokenizer_en = Tokenizer(models.WordLevel(unk_token=""))
12 tokenizer_vi = Tokenizer(models.WordLevel(unk_token=""))
13
14 tokenizer_en.pre_tokenizer = pre_tokenizers.Whitespace()
15 tokenizer_vi.pre_tokenizer = pre_tokenizers.Whitespace()
16
17 trainer = trainers.WordLevelTrainer(
18 vocab_size=15000,
19 min_frequency=2,
20 special_tokens=["<pad>", "<unk>", "<bos>", "<eos>"]
21)
22
23 # train tokenizer
24 tokenizer_en.train_from_iterator(ds["train"]["en"], trainer)
25 tokenizer_vi.train_from_iterator(ds["train"]["vi"], trainer)
26
27 # tokenizer
28 tokenizer_en.save("tokenizer_en.json")
29 tokenizer_vi.save("tokenizer_vi.json")

```

## 2. Encoding

```

1 from transformers import PreTrainedTokenizerFast
2
3 MAX_LEN = 75
4
5 # Load tokenizer
6 tokenizer_en = PreTrainedTokenizerFast(
7 tokenizer_file="tokenizer_en.json",
8 unk_token="", pad_token="", bos_token="",
9 eos_token=""
9)

```

```

10 tokenizer_vi = PreTrainedTokenizerFast(
11 tokenizer_file="tokenizer_vi.json",
12 unk_token="", pad_token="", bos_token="",
13 eos_token=""
14)
15 def preprocess_function(examples):
16 src_texts = examples["en"]
17 tgt_texts = ["<bos> " + sent + "<eos>" for sent in
18 examples["vi"]]
19
20 src_encodings = tokenizer_en(
21 src_texts, padding="max_length", truncation=True,
22 max_length=MAX_LEN
23)
24 tgt_encodings = tokenizer_vi(
25 tgt_texts, padding="max_length", truncation=True,
26 max_length=MAX_LEN
27)
28
29 return {
30 "input_ids": src_encodings["input_ids"],
31 "labels": tgt_encodings["input_ids"],
32 }
33
34 preprocessed_ds = ds.map(preprocess_function, batched=True)

```

### 3. Modeling

Trong phần này chúng ta xây dựng mô hình cơ bản là Sequence-to-Sequence có sử dụng mạng RNNs (như GRU) và mô hình Transformer

#### a. RNNs

```

1 import torch
2 import torch.nn as nn
3 from transformers import PreTrainedModel, PretrainedConfig
4
5 class Seq2SeqRNNConfig(PretrainedConfig):
6 def __init__(self,
7 vocab_size_src=10000, vocab_size_tgt=10000,
8 embedding_dim=128, hidden_size=128, dropout
9 =0.1, **kwargs):
10 super().__init__(**kwargs)
11 self.vocab_size_src = vocab_size_src
12 self.vocab_size_tgt = vocab_size_tgt

```

```
12 self.embedding_dim = embedding_dim
13 self.hidden_size = hidden_size
14 self.dropout = dropout
15
16 class EncoderRNN(nn.Module):
17 def __init__(self, input_size, embedding_dim, hidden_size,
18 dropout_p=0.1):
19 super(EncoderRNN, self).__init__()
20 self.hidden_size = hidden_size
21 self.embedding = nn.Embedding(input_size,
22 embedding_dim)
23 self.gru = nn.GRU(embedding_dim, hidden_size,
24 batch_first=True)
25 self.dropout = nn.Dropout(dropout_p)
26
27 def forward(self, input):
28 embedded = self.dropout(self.embedding(input)) # B x
29 S x H
30 output, hidden = self.gru(embedded) # B x S x H, B x
31 H
32 return output, hidden
33
34 class DecoderRNN(nn.Module):
35 def __init__(self, hidden_size, embedding_dim,
36 output_size):
37 super(DecoderRNN, self).__init__()
38 self.embedding = nn.Embedding(output_size,
39 embedding_dim)
40 self.gru = nn.GRU(embedding_dim, hidden_size,
41 batch_first=True)
42 self.out = nn.Linear(hidden_size, output_size) # LM
43 Head
44
45 def forward(self, input, hidden):
46 output = self.embedding(input)
47 output, hidden = self.gru(output, hidden)
48 output = self.out(output) # B x 1 x Vocab
49 return output, hidden
50
51 class Seq2SeqRNNModel(PreTrainedModel):
52 config_class = Seq2SeqRNNConfig
53
54 def __init__(self, config, tokenizer_en):
55 super().__init__(config)
56 self.encoder = EncoderRNN(
```

```

48 config.vocab_size_src, config.embedding_dim,
49 config.hidden_size, config.dropout)
50 self.decoder = DecoderRNN(
51 config.hidden_size, config.embedding_dim, config.
vocab_size_tgt)
52 self.BOS_IDX = tokenizer_en.bos_token_id
53 self.loss_fn = nn.CrossEntropyLoss(ignore_index=0) #
Ignore PAD Token
54
55 def forward(self, input_ids, labels):
56 batch_size, seq_len = labels.shape
57 decoder_input = torch.full((batch_size, 1), self.
BOS_IDX, dtype=torch.long).to(input_ids.device) # Sửa lỗi
58 encoder_output, decoder_hidden = self.encoder(
input_ids)
59 decoder_outputs = []
60
61 for i in range(seq_len):
62 decoder_output, decoder_hidden = self.decoder(
decoder_input, decoder_hidden)
63 decoder_outputs.append(decoder_output)
64 decoder_input = labels[:, i].unsqueeze(1) #
Teacher forcing
65
66 logits = torch.cat(decoder_outputs, dim=1) # B x S x
Vocab
67 loss = self.loss_fn(logits.view(-1, logits.shape[-1]),
, labels.view(-1))
68 return {"loss": loss, "logits": logits}
69
70 config = Seq2SeqRNNConfig(
71 vocab_size_src=len(tokenizer_en), vocab_size_tgt=len(
tokenizer_vi)
72)
73 model = Seq2SeqRNNModel(config, tokenizer_en)

```

## b. Transformer

```

1 import torch
2 import torch.nn as nn
3 from transformers import PreTrainedModel, PretrainedConfig
4
5 def generate_square_subsequent_mask(sz, device):
6 mask = (torch.triu(torch.ones((sz, sz), device=device))
== 1).transpose(0, 1)

```

```
7 mask = mask.float().masked_fill(mask == 0, float('-inf'))
8 .masked_fill(mask == 1, float(0.0))
9 return mask
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
```

```
mask = mask.float().masked_fill(mask == 0, float('-inf'))
 .masked_fill(mask == 1, float(0.0))
return mask

def create_mask(src, tgt):
 src_seq_len = src.shape[1]
 tgt_seq_len = tgt.shape[1]
 device = src.device

 tgt_mask = generate_square_subsequent_mask(tgt_seq_len,
 device).to(torch.bool)
 src_mask = torch.zeros((src_seq_len, src_seq_len), device=device)
 .type(torch.bool)
 src_padding_mask = (src == 0)
 tgt_padding_mask = (tgt == 0)
 return src_mask, tgt_mask, src_padding_mask,
 tgt_padding_mask

class Seq2SeqTransformerConfig(PretrainedConfig):
 def __init__(self, vocab_size_src=10000, vocab_size_tgt=10000,
 max_seq_length=50,
 d_model=256, num_heads=8, num_layers=6, dropout=0.1, **kwargs):
 super().__init__(**kwargs)
 self.vocab_size_src = vocab_size_src
 self.vocab_size_tgt = vocab_size_tgt
 self.max_seq_length = max_seq_length
 self.d_model = d_model
 self.num_heads = num_heads
 self.num_layers = num_layers
 self.dropout = dropout

class Seq2SeqTransformerModel(PreTrainedModel):
 config_class = Seq2SeqTransformerConfig

 def __init__(self, config):
 super().__init__(config)

 self.embedding_src = nn.Embedding(
 config.vocab_size_src, config.d_model)
 self.embedding_tgt = nn.Embedding(
 config.vocab_size_tgt, config.d_model)

 self.position_embedding_src = nn.Embedding(
```

```
46 config.max_seq_length, config.d_model)
47 self.position_embedding_tgt = nn.Embedding(
48 config.max_seq_length, config.d_model)
49
50 self.transformer = nn.Transformer(
51 d_model=config.d_model,
52 nhead=config.num_heads,
53 num_encoder_layers=config.num_layers,
54 num_decoder_layers=config.num_layers,
55 dropout=config.dropout,
56 batch_first=True
57)
58
59 self.generator = nn.Linear(
60 config.d_model, config.vocab_size_tgt
61)
62 self.loss_fn = nn.CrossEntropyLoss(ignore_index=0) #
Ignore PAD token
63
64 def forward(self, input_ids, labels):
65 tgt_input = labels[:, :-1]
66 tgt_output = labels[:, 1:]
67 batch_size, seq_len_src = input_ids.shape
68 _, seq_len_tgt = tgt_input.shape
69
70 src_positions = torch.arange(seq_len_src, device=
71 input_ids.device).unsqueeze(0)
72 tgt_positions = torch.arange(seq_len_tgt, device=
73 labels.device).unsqueeze(0)
74
75 src_embedded = self.embedding_src(input_ids) + self.
76 position_embedding_src(src_positions)
77 tgt_embedded = self.embedding_tgt(tgt_input) + self.
78 position_embedding_tgt(tgt_positions)
79
80 src_mask, tgt_mask, src_key_padding_mask,
81 tgt_key_padding_mask = create_mask(input_ids, tgt_input)
82
83 outs = self.transformer(
84 src_embedded, tgt_embedded, src_mask, tgt_mask,
85 src_key_padding_mask=src_key_padding_mask,
86 tgt_key_padding_mask=tgt_key_padding_mask
87)
88
89 logits = self.generator(outs)
```

```

85 loss = self.loss_fn(logits.permute(0, 2, 1),
86 tgt_output)
87
88 return {"loss": loss, "logits": logits}
89
90 def encode(self, src, src_mask):
91 _, seq_len_src = src.shape
92 src_positions = torch.arange(
93 seq_len_src, device=src.device).unsqueeze(0)
94 src_embedded = self.embedding_src(src) + self.
95 position_embedding_src(
96 src_positions)
97 return self.transformer.encoder(src_embedded,
98 src_mask)
99
100 def decode(self, tgt, encoder_output, tgt_mask):
101 _, seq_len_tgt = tgt.shape
102 tgt_positions = torch.arange(
103 seq_len_tgt, device=tgt.device).unsqueeze(0)
104 tgt_embedded = self.embedding_tgt(tgt) + self.
105 position_embedding_tgt(
106 tgt_positions)
107 return self.transformer.decoder(
108 tgt_embedded, encoder_output, tgt_mask
109)
110
111 config = Seq2SeqTransformerConfig(
112 vocab_size_src=len(tokenizer_en), vocab_size_tgt=len(
113 tokenizer_vi), max_seq_length=75
114)
115
116 model = Seq2SeqTransformerModel(config)

```

### c. Test the model

Sau khi khởi tạo mô hình, thực hiện đoạn code sau để kiểm tra quá trình xây dựng mô hình đã hoàn tất hay chưa

```

1 input_ids = torch.tensor([preprocessed_ds['train'][0][
2 'input_ids']])
3
4 pred = model(input_ids, labels)
5 # loss => 9.
6 # logits => shape: 1x75x13684

```

#### 4. Trainer

```

1 # Disable wandb
2 import os
3 os.environ['WANDB_DISABLED'] = 'true'
4
5 from transformers import Trainer, TrainingArguments
6
7 # Training
8 training_args = TrainingArguments(
9 output_dir='./en-vi-machine-translation',
10 logging_dir="logs",
11 eval_strategy="epoch",
12 save_strategy="epoch",
13 logging_strategy="epoch",
14 per_device_train_batch_size=512,
15 per_device_eval_batch_size=512,
16 num_train_epochs=25,
17 learning_rate=2e-5,
18 save_total_limit=1,
19 % report_to="wandb",
20)
21
22 trainer = Trainer(
23 model=model,
24 args=training_args,
25 train_dataset=preprocessed_ds["train"],
26 eval_dataset=preprocessed_ds["validation"]
27)
28
29 trainer.train()

```

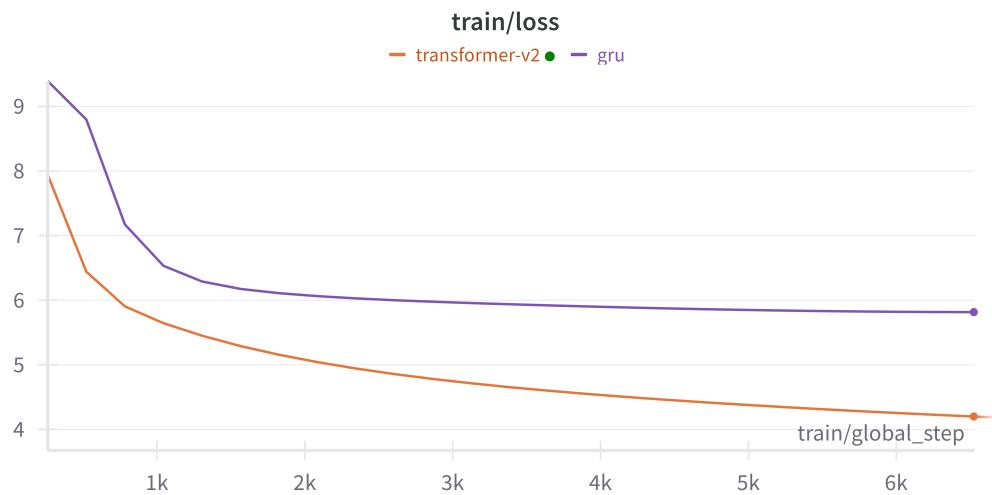
Kết quả training model:

#### 5. Evaluation

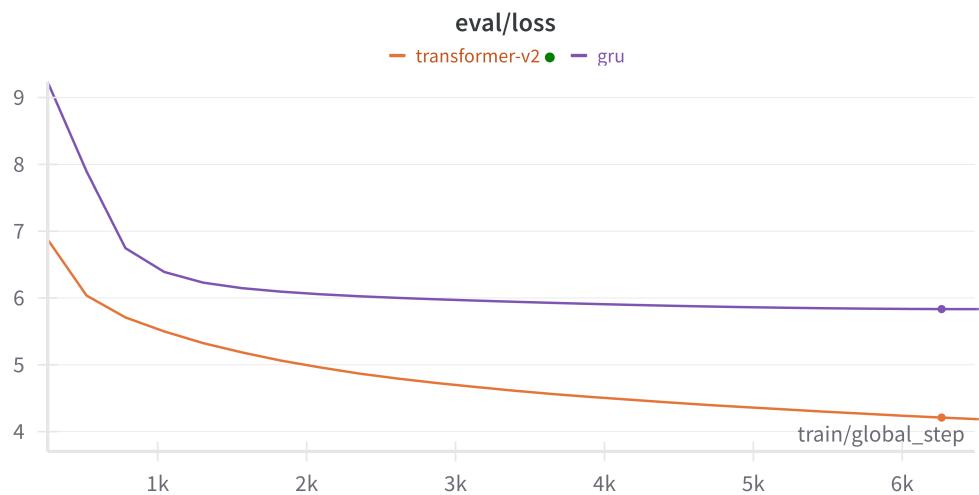
```

1 def greedy_decode(model, src, src_mask, max_len, start_symbol
, device="cpu"):
2 src = src.to(device)
3 src_mask = src_mask.to(device)
4
5 memory = model.encode(src, src_mask)
6 ys = torch.ones(1, 1).fill_(start_symbol).type(torch.long
).to(device)
7 for i in range(max_len-1):

```



Hình 46.4: Training loss.



Hình 46.5: Validation loss.

```
8 memory = memory.to(device)
9 tgt_mask = (generate_square_subsequent_mask(ys.size
10 (1), device)
11 .type(torch.bool)).to(device)
```

```
11 out = model.decode(ys, memory, tgt_mask)
12 prob = model.generator(out[:, -1, :]) # LM Head
13 _, next_word = torch.max(prob, dim=1)
14 next_word = next_word[-1].item() # index
15
16 ys = torch.cat([ys, torch.ones(1, 1).type_as(
17 src.data).fill_(next_word)], dim=1)
18 if next_word == 3: #EOS : 3
19 break
20 return ys
21
22 def translate(model, src_sentence, device):
23 model.eval()
24 input_ids = tokenizer_en([src_sentence], return_tensors='
25 pt')[‘input_ids’].to(device)
26 num_tokens = input_ids.shape[1]
27 src_mask = (torch.zeros(num_tokens, num_tokens)).type(
28 torch.bool).to(device)
29 tgt_tokens = greedy_decode(
30 model, input_ids, src_mask, max_len=num_tokens + 5,
31 start_symbol=2, device=device)
32 return tokenizer_vi.decode(a.detach().cpu()[0])
33 translate(model, "i go to school", model.device) # => toi den
34 truong
35
36 pred_sentences, tgt_sentences = [], []
37 for sample in tqdm(ds['test']):
38 src_sentence = sample['en']
39 tgt_sentence = sample['vi']
40
41 pred_sentence = translate(model, src_sentence)
42 pred_sentences.append(pred_sentence)
43
44 tgt_sentences.append(tgt_sentence)
45
46 bleu_score = sacrebleu.corpus_bleu(pred_sentences, [
47 tgt_sentences], force=True)
48 bleu_score => 46.8/16.8/6.4/2.4
```

### 46.3 Câu hỏi trắc nghiệm

**Câu hỏi 63** Mục tiêu của bài toán dịch máy là gì?

- a) Xây dựng mô hình dịch từ ngôn ngữ nguồn sang ngôn ngữ đích
- b) Xây dựng mô hình phân loại văn bản
- c) Xây dựng mô hình tóm tắt văn bản
- d) Xây dựng mô hình phân tích cảm xúc

**Câu hỏi 64** Số lượng sample tập test của bộ dữ liệu sử dụng trong thực nghiệm là?

- a) 1267
- b) 1268
- c) 1269
- d) 1270

**Câu hỏi 65** Hệ thống nào sau đây không phải là hệ thống dịch?

- a) Bing Translator
- b) Google Translate
- c) ChatGPT
- d) Dall-E

**Câu hỏi 66** Thư viện sentencepiece dùng để làm gì?

- a) Xây dựng Transformer-Encoder
- b) Xây dựng Transformer-Decoder
- c) Xây dựng Subword-Based Tokenization
- d) Huấn luyện mô hình Transformer

**Câu hỏi 67** BERT sử dụng kiến trúc nào sau đây?

- a) Bi-LSTM
- b) RNN
- c) Transformer-Encoder

d) Transformer-Decoder

**Câu hỏi 68** BERT-Large sử dụng bao nhiêu khối encoder?

- a) 6
- b) 12
- c) 18
- d) 24

**Câu hỏi 69** Trong số các mask tokens trong quá trình huấn luyện BERT. Tỷ lệ mask nào là tối ưu nhất cho BERT?

- a) 80% thay bởi token [MASK] : 20% thay bởi token bất kỳ : 0% giữ nguyên không đổi
- b) 80% thay bởi token [MASK] : 0% thay bởi token bất kỳ : 20% giữ nguyên không đổi
- c) 80% thay bởi token [MASK] : 10% thay bởi token bất kỳ : 10% giữ nguyên không đổi
- d) 80% thay bởi token [MASK] : 15% thay bởi token bất kỳ : 5% giữ nguyên không đổi

**Câu hỏi 70** GPT sử dụng kiến trúc nào sau đây?

- a) Bi-LSTM
- b) RNN
- c) Transformer-Encoder
- d) Transformer-Decoder

**Câu hỏi 71** GPT-Small sử dụng bao nhiêu khối decoder?

- a) 6
- b) 12
- c) 18
- d) 24

**Câu hỏi 72** Độ đo thường sử dụng cho mô hình dịch máy là?

- a) F1
- b) BLEU
- c) Accuracy
- d) ROUGE

## 46.4 Phụ lục

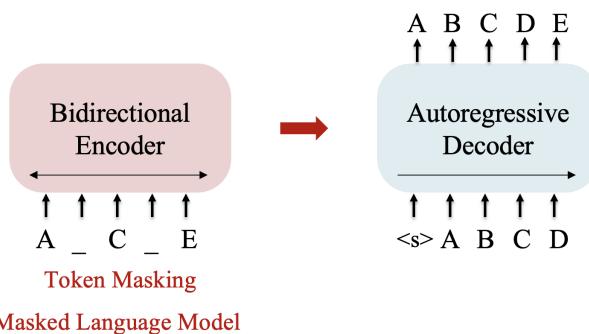
1. **Hint:** Dựa vào file tải về [Neural Machine Translation](#) để hoàn thiện các đoạn code.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, admin mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

- *Hết* -

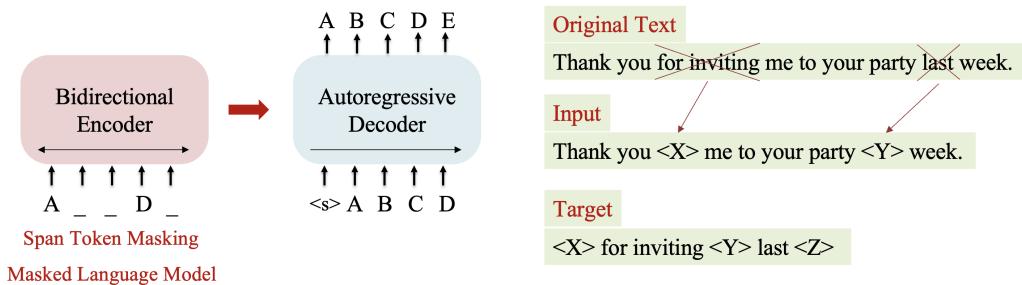
# Chương 47

## Project 1: Dịch ngôn ngữ Anh-Việt với điều kiện tài nguyên ít

### 47.1 Giới thiệu



Hình 47.1: Mô hình BART.



Hình 47.2: Mô hình T5.

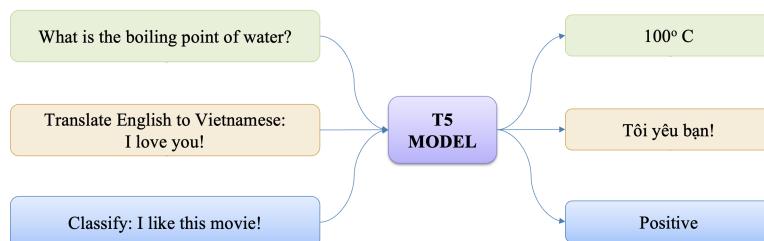
**Dịch Máy (Machine Translation)** với mục đích tự động dịch văn bản từ ngôn ngữ tự nhiên này sang ngôn ngữ tự nhiên khác. Một trong những thách thức lớn hiện nay của các hệ thống dịch là có ít tài nguyên để huấn luyện mô hình. Vì vậy, trong phần này, chúng ta sẽ tập trung vào cải thiện các mô hình dịch với ít tài nguyên (Low-resource machine translation), số

lượng các cặp dữ liệu song ngữ hạn chế, ví dụ mô hình dịch huấn luyện với chỉ 20000 cặp câu tiếng Anh và tiếng Việt. Vì vậy, để cải tiến mô hình dịch trong trường hợp có ít tài nguyên, chúng ta tập trung vào 2 hướng tiếp cận như sau:

1. Hướng 1: Sử dụng mô hình pre-trained mBART50, T5
2. Hướng 2: Sử dụng kỹ thuật để tăng cường dữ liệu như thu thập thêm dữ liệu, kỹ thuật học bán giám sát,...

Trong đó sử dụng các pre-trained models như mBART50 hoặc T5 sẽ đạt hiệu quả tốt hơn. Cả 2 mô hình đều có kiến trúc Transformer với nhiều lớp xếp chồng lên nhau.

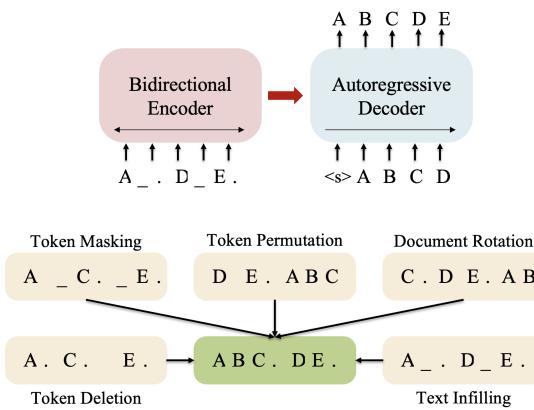
1. Mô hình T5: T5 là mô hình sử dụng cùng dạng biểu diễn chuỗi-chuỗi cho tất cả các bài toán, được mô tả như sau:



Hình 47.3: Biểu diễn đầu vào / đầu ra mô hình T5.

Kỹ thuật huấn luyện mô hình T5 được biểu diễn trong Hình 2.

2. Mô hình BART: Mô hình BART xây dựng để tối ưu cho các bài toán sinh chuỗi, vì vậy BART có một số hàm mục tiêu như sau:



Hình 47.4: Các hàm mục tiêu của mô hình BART.

## 47.2 Fine-tuning mBART50

Để xây dựng, đánh giá hiệu suất các mô hình chúng ta sử dụng bộ dữ liệu dịch **IWSLT'15 English - Vietnamese** với số lượng mẫu cho training: 133,317 cặp câu song ngữ, tập validation: 1,553 cặp câu song ngữ và tập test: 1,269 cặp câu song ngữ. Chiều dịch sẽ từ tiếng anh sang tiếng việt.

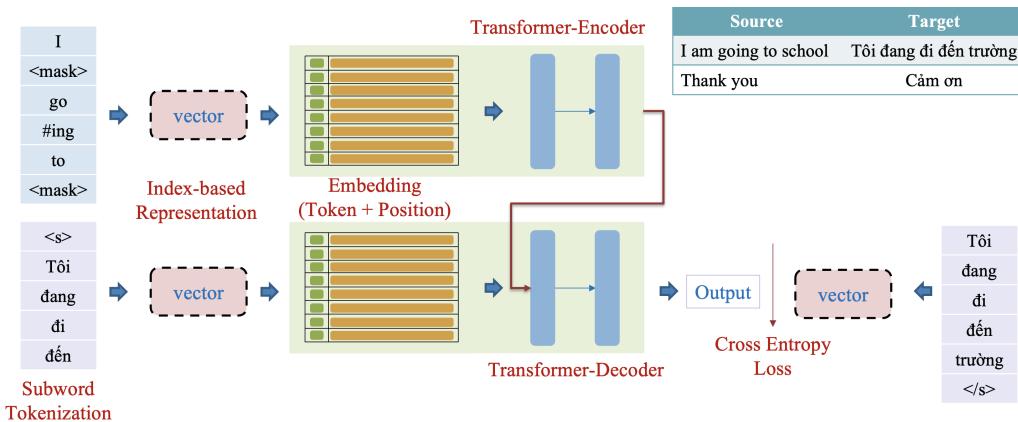
Metric để đánh giá chúng ta sử dụng: ScacreBleu (BLEU)

Trong phần này, chúng ta sẽ sử dụng kỹ thuật fine-tuning để huấn luyện mô hình dịch theo chiều EN-VI.

Mô hình được sử dụng là **mBART50** được huấn luyện trên 50 ngôn ngữ khác nhau và phù hợp cho bài toán dịch EN-VI.

Các bước thực hiện như sau:

- Dataset: tải về bộ dữ liệu
- Tokenizer: tải về bộ mã hoá
- Encoding: chuyển văn bản thành vector
- Model: tải về mô hình mBART50
- Evaluate: định nghĩa độ đo đánh giá mô hình
- Trainer: huấn luyện mô hình



Hình 47.5: Kỹ thuật fine-tuning mô hình BART.

- Inference: kiểm thử kết quả
- Deployment: triển khai trên streamlit

### 1. Dataset

Chạy đoạn code sau đây để tải về bộ dữ liệu.

```

1 # install libs
2 !pip install -q transformers sentencepiece datasets
 accelerate evaluate sacrebleu
3
4 # import libs
5 from datasets import load_dataset
6
7 ds = load_dataset("thainq107/iwslt2015-en-vi")

```

### 2. Tokenizer

Chạy đoạn code dưới đây để tải về bộ tokenizer của mô hình mBART50.

```

1 from transformers import AutoTokenizer
2 model_name = "facebook/mbart-large-50-many-to-many-mmt"
3 tokenizer = AutoTokenizer.from_pretrained(model_name)

```

### 3. Encoding

Sử dụng bộ tách từ để biểu diễn văn bản thành vector.

```
1 import torch
2
3 MAX_LEN = 75
4
5 def preprocess_function(examples):
6 input_ids = tokenizer(
7 examples["en"], padding="max_length", truncation=
8 True, max_length=MAX_LEN
9)["input_ids"]
10
11 labels = tokenizer(
12 examples["vi"], padding="max_length", truncation=
13 True, max_length=MAX_LEN
14)["input_ids"]
15 labels = [
16 [-100 if item == tokenizer.pad_token_id else item
17 for item in label]
18 for label in labels]
19
20 return {
21 "input_ids": torch.tensor(input_ids),
22 "labels": torch.tensor(labels)
23 }
24
25 preprocessed_ds = ds.map(preprocess_function, batched=
26 True)
```

### 4. Model

Tải về mô hình mBART50.

```
1 from transformers import AutoModelForSeq2SeqLM
2
3 model_name = "facebook/mbart-large-50-many-to-many-mmt"
4 model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
```

## 5. Evaluation

Sử dụng độ đo BLEU để đánh giá.

```
1 import numpy as np
2 import evaluate
3 metric = evaluate.load("sacrebleu")
4
5 def postprocess_text(preds, labels):
6 preds = [pred.strip() for pred in preds]
7 labels = [[label.strip()] for label in labels]
8
9 return preds, labels
10
11 def compute_metrics(eval_preds):
12 preds, labels = eval_preds
13 if isinstance(preds, tuple):
14 preds = preds[0]
15
16 preds= np.where(preds != -100, preds, tokenizer.
17 pad_token_id)
17 decoded_preds = tokenizer.batch_decode(
18 preds, skip_special_tokens=True,
19 clean_up_tokenization_spaces=True
20)
21
21 labels= np.where(labels != -100, labels, tokenizer.
22 pad_token_id)
22 decoded_labels = tokenizer.batch_decode(
23 labels, skip_special_tokens=True,
24 clean_up_tokenization_spaces=True
25)
26
26 decoded_preds, decoded_labels = postprocess_text(
27 decoded_preds, decoded_labels
28)
29
30 result = metric.compute(predictions=decoded_preds,
31 references=decoded_labels)
31 result = {"bleu": result["score"]}
32
33 return result
```

## 6. Trainer

Định nghĩa các tham số trong quá trình huấn luyện để tối ưu mô hình và giúp mô hình học được nhiều đặc trưng hơn. Sau khi huấn luyện có thể đẩy lên huggingface thông qua tài khoản cá nhân để sử dụng.

```
1 # Disable wandb
2 import os
3 os.environ["WANDB_DISABLED"] = "true"
4
5 from transformers import Seq2SeqTrainingArguments,
6 DataCollatorForSeq2Seq, Seq2SeqTrainer
7 training_args = Seq2SeqTrainingArguments(
8 output_dir=". ./en-vi-mbart50",
9 logging_dir="logs",
10 logging_steps=1000,
11 predict_with_generate=True,
12 eval_strategy="steps",
13 eval_steps=1000,
14 save_strategy="steps",
15 save_steps=1000,
16 per_device_train_batch_size=32,
17 per_device_eval_batch_size=32,
18 save_total_limit=1,
19 num_train_epochs=3,
20 load_best_model_at_end=True,
21)
22 data_collator = DataCollatorForSeq2Seq(tokenizer, model=
23 model)
24 trainer = Seq2SeqTrainer(
25 model,
26 training_args,
27 train_dataset=preprocessed_ds["train"],
28 eval_dataset=preprocessed_ds["validation"],
29 data_collator=data_collator,
30 processing_class=tokenizer,
31 compute_metrics=compute_metrics
32)
```

Kết quả hàm loss so sánh với mô hình transformer và seq2seq-gru hình sau:



Hình 47.6: Kết quả train loss.



Hình 47.7: Kết quả eval loss.

Kết quả đánh giá trên tập validation thu được như sau:

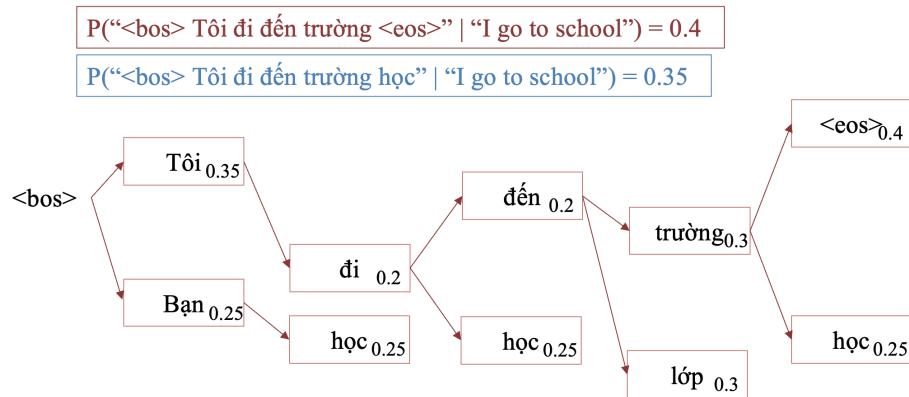
| Training Loss | Epoch  | Step  | Bleu    | Validation Loss |
|---------------|--------|-------|---------|-----------------|
| 1.228         | 0.2400 | 1000  | 33.4010 | 1.3419          |
| 1.2022        | 0.4800 | 2000  | 34.0752 | 1.3063          |
| 1.1771        | 0.7199 | 3000  | 34.1612 | 1.2806          |
| 1.1607        | 0.9599 | 4000  | 34.3856 | 1.2582          |
| 0.9698        | 1.1999 | 5000  | 34.3075 | 1.2860          |
| 0.9298        | 1.4399 | 6000  | 34.4419 | 1.2671          |
| 0.9282        | 1.6799 | 7000  | 34.7962 | 1.2552          |
| 0.9174        | 1.9198 | 8000  | 34.7904 | 1.2516          |
| 0.8538        | 2.1598 | 9000  | 1.3000  | 34.3462         |
| 0.822         | 2.3998 | 10000 | 1.2953  | 34.3928         |
| 0.8206        | 2.6398 | 11000 | 1.2834  | 34.4372         |
| 0.8177        | 2.8798 | 12000 | 1.2825  | 34.6635         |

Hình 47.8: Kết quả cho từng bước và kết quả bleu.

## 7. Inference

Sau khi train model có thể sử dụng model từ huggingface để dự đoán và đánh giá trên tập test. Sử dụng greedy search (luôn chọn token dự đoán với giá trị xác suất lớn nhất) và beam search (chọn top k các token với giá trị xác suất lớn nhất) để đánh giá.

Thuật toán của beam search được mô tả hình sau với beam size = 2:



Hình 47.9: Thuật toán beam search.

```

1 # download model
2 from transformers import pipeline
3 translator = pipeline(model="thainq107/en-vi-mbart50")
4
5 # test a sample with beam search
6 translated_text = translator("I go to school", num_beams=2)
7 translated_text
8
9 # greedy search for test set
10 pred_sentences = translator(ds["test"]["en"], batch_size=32, num_beams=1, do_sample=False)
11
12 # beam search for test set
13 pred_sentences = translator(ds["test"]["en"], batch_size=32, num_beams=5)
14
15 # evaluate
16 import sacrebleu
17
18 bleu_score = sacrebleu.corpus_bleu(pred_sentences, [ds["test"]["vi"]], force=True)
19 bleu_score

```

Kết quả đánh giá trên tập test set của các mô hình như sau:

| Experiment | Model                                | ScoreBLEU |
|------------|--------------------------------------|-----------|
| #1         | Standard Transformer (Greedy Search) | 24.66     |
| #2         | mBART50 (Greedy Search)              | 33.50     |
| #3         | mBART50 (Beam size = 5)              | 34.17     |

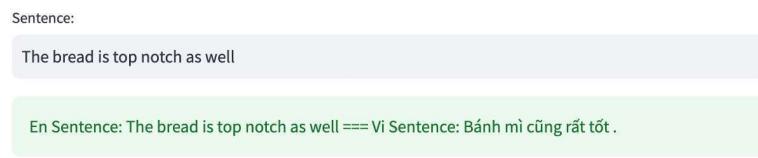
Hình 47.10: Kết quả BLEU trên tập test.

## 7. Deployment

Trong phần này, chúng ta triển khai mô hình trên streamlit. Tham khảo về [code](#) và [demo](#).

## En-Vi Machine Translation

**Model: mBART50. Dataset: IWSLT15-En-Vi**



Hình 47.11: Triển khai ứng dụng trên Streamlit.

### 47.3 Câu hỏi trắc nghiệm

**Câu hỏi 73** Mô hình BART có kiến trúc là gì?

- a) Transformer-Encoder
- b) Transformer-Decoder
- c) Transformer-Encoder-Decoder
- d) BiLSTM

**Câu hỏi 74** Hàm mục tiêu nào sau đây không phải của BART?

- a) Token Masking
- b) Token Deletion
- c) Sentence Permutation
- d) Next Sentence Prediction

**Câu hỏi 75** Mô hình BART-Base có bao nhiêu lớp encoder?

- a) 6
- b) 12
- c) 24
- d) 48

**Câu hỏi 76** Mô hình BART-Large có bao nhiêu lớp encoder?

- a) 6
- b) 12
- c) 24
- d) 48

**Câu hỏi 77** mBART50 được huấn luyện trên bao nhiêu ngôn ngữ?

- a) 5
- b) 25
- c) 50
- d) 100

**Câu hỏi 78** mBART50 là mô hình tiền huấn luyện đa ngữ cho bài toán nào?

- a) Text Summarization
- b) Text Classification
- c) Question-Answering
- d) Machine Translарion

**Câu hỏi 79** Tập dữ liệu dịch máy tiếng việt tiếng anh là?

- a) PhoMT
- b) IMDB-Review
- c) C4
- d) ROOT

**Câu hỏi 80** Mô hình nào sau đây có kiến trúc tương tự mô hình BART?

- a) BERT
- b) GPT
- c) RoBERTa
- d) T5

**Câu hỏi 81** Mô hình T5 sử dụng tiền tố nào cho bài toán dịch?

- a) translate English to Vietnamese
- b) translate to Vietnamese from English
- c) Cả 2 đáp án đều đúng
- d) Cả 2 đáp án đều sai

**Câu hỏi 82** Độ đo đánh giá được sử dụng trong phần thực nghiệm là?

- a) F1
- b) SacreBLEU
- c) Accuracy
- d) ROUGE

## 47.4 Phụ lục

1. **Hint:** Dựa vào file tải về [Fine-tuning mBART50 for En-Vi Machine Translation](#) để hoàn thiện các đoạn code.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, admin mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

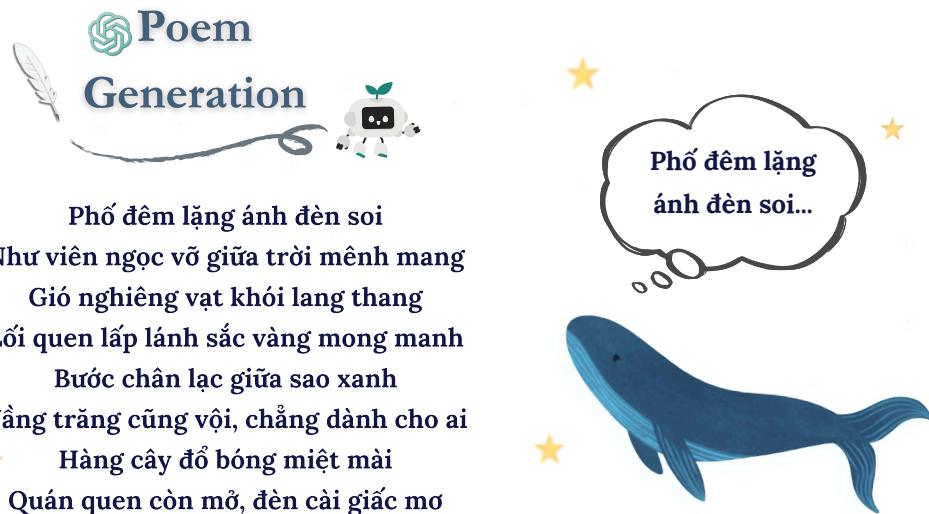
- *Hết* -

# Chương 48

## Project 2: Xây dựng chương trình sinh thơ

### 48.1 Giới thiệu

Ngôn từ vốn mang vẻ đẹp của tri thức, và khi đi vào thơ ca, chúng mở ra một thế giới đầy tính nghệ thuật. Từ lâu, việc cảm thụ nghệ thuật, tư duy và sáng tạo được xem là đặc quyền của con người. Nhưng sẽ như thế nào nếu các mô hình học máy có thể nắm bắt được vẻ đẹp ấy, tự mình dệt nên những vần thơ mang nhịp điệu và ý nghĩa? Bài toán **Text Generation** đã mở ra khả năng cho máy tính không chỉ hiểu mà còn tạo ra ngôn ngữ một cách có ý thức ngữ cảnh. Đặc biệt, các mô hình như [ChatGPT](#) đã chứng minh khả năng sinh văn bản với độ trôi chảy và sáng tạo cao. Nhưng khi áp dụng vào sinh thơ, thách thức không chỉ nằm ở việc tạo ra câu từ hợp lý, mà còn phải đảm bảo vần điệu, nhịp điệu và sắc thái ngữ cảnh.



Hình 48.1: Bài toán sinh thơ tiếng Việt (thơ trong hình được tạo từ mô hình sinh thơ).

Thơ ca Việt Nam có nhiều thể loại phong phú, mỗi thể thơ mang đặc trưng về nhịp điệu, số chữ trong câu, cách gieo vần và ý nghĩa biểu đạt. Trong số đó, **thơ ngũ ngôn** hay thơ năm chữ là một thể thơ đặc biệt phổ biến, đơn giản nhưng vẫn giữ được sự hài hòa về âm điệu. Mỗi dòng gồm 5 âm tiết, thường được chia nhịp 2/3 hoặc 3/2 để tạo sự uyển chuyển khi đọc. Cách gieo vần linh hoạt, thường là vần liền (AABB) hoặc vần cách (ABAB), giúp bài thơ có tính nhạc cao, phù hợp với việc diễn đạt tình cảm, thiên nhiên, hoặc triết lý nhân sinh.

Ví dụ về một đoạn thơ năm chữ sử dụng vần cách (ABAB):

*Gió lồng từng khóm trúc  
Lá rì rào reo ca  
Dòng sông trôi ánh bạc  
Lững lờ con thuyền qua.*

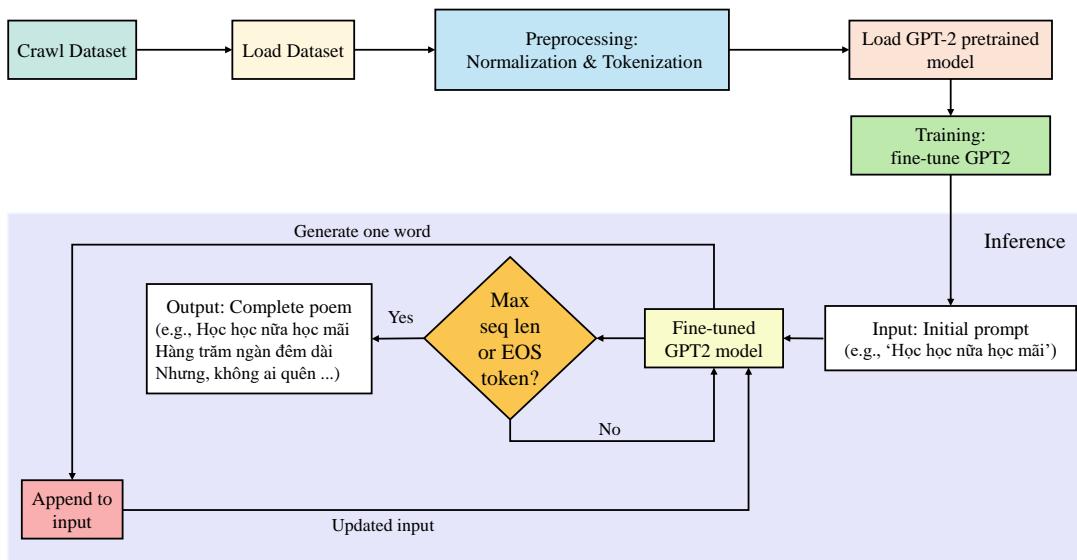
— Tố Hữu, "Tiếng ru" (1956)

Trong project này, chúng ta sẽ cùng triển khai một chương trình sử dụng mô hình Text Generation với chủ đề sinh thơ Tiếng Việt, chủ yếu là thể

thơ năm chữ dựa vào một từ tiếng Việt đầu vào từ người dùng. **Như vậy, Input/Output của chương trình là:**

- **Input:** Một chuỗi gồm các kí tự mở đầu cho bài thơ.
- **Output:** Bài thơ hoàn chỉnh.

Dựa theo nội dung được mô tả ở trên, ta sẽ có một pipeline với các bước thực hiện trong chương trình tổng thể như sau:



Hình 48.2: Pipeline mô tả bài toán sinh thơ tiếng Việt bằng GPT-2.

1. **Crawl Dataset (Thu thập dữ liệu):** Dữ liệu đầu vào là các bài thơ Tiếng Việt, được thu thập từ nhiều nguồn khác nhau. Quá trình này bao gồm:
  - Crawl dữ liệu từ các trang web hoặc tập dữ liệu có sẵn.
  - Lưu trữ và kiểm tra dữ liệu để đảm bảo chất lượng.
2. **Load and Preprocessing Data (Tải và tiền xử lý dữ liệu):** Sau khi thu thập dữ liệu, cần làm sạch và chuyển đổi dữ liệu về định dạng phù hợp. Các bước chính gồm:

- Chuẩn hóa văn bản: loại bỏ ký tự đặc biệt, xử lý khoảng trắng, chuẩn hóa dấu câu.
- Tokenization: chuyển văn bản thành các token để mô hình GPT-2 có thể xử lý.
- Tạo tập huấn luyện từ dữ liệu đã xử lý.

3. **Training (Huấn luyện mô hình):** Chúng ta tải mô hình GPT-2 đã được tiền huấn luyện và thực hiện fine-tuning với tập dữ liệu thơ tiếng Việt:

- Nạp mô hình GPT-2 pre-trained.
- Đào tạo mô hình trên tập dữ liệu thơ.
- Lưu mô hình đã fine-tune để sử dụng cho quá trình suy luận.

4. **Inference (Thực hiện dự đoán):** Sau khi mô hình được huấn luyện, quá trình sinh thơ được thực hiện như sau:

- Người dùng nhập vào một đoạn prompt ban đầu (ví dụ: “Học học nữa học mãi”).
- Prompt được token hóa và đưa vào mô hình GPT-2 đã fine-tune.
- Mô hình sinh ra từng token một theo cơ chế dự đoán từ tiếp theo.
- Token mới sinh ra được nối vào đầu vào và tiếp tục đưa vào mô hình.
- Quá trình lặp lại cho đến khi đạt độ dài tối đa hoặc gặp token kết thúc (EOS token).
- Kết quả cuối cùng là bài thơ hoàn chỉnh.

Pipeline này đảm bảo quá trình sinh thơ diễn ra tự động, từ thu thập dữ liệu đến tạo ra bài thơ chất lượng cao.

## 48.2 Cài đặt chương trình

Trong phần này, chúng ta sẽ thực hiện hai giai đoạn chính của project để hoàn thiện được mô hình yêu cầu, bao gồm: Thu thập dữ liệu và Xây dựng mô hình. **Nội dung cụ thể như sau:**

### 48.2.1 Thu thập dữ liệu

Để huấn luyện được mô hình với Input/Output theo đúng như yêu cầu đã đề ra ở phần trước, chúng ta cần thu thập vào xây dựng một bộ dữ liệu theo đúng mô tả. Đối với dữ liệu thơ, có rất nhiều trang web tổng hợp các bài thơ của Việt Nam cũng như thế giới. Tuy nhiên ở project này, ta sẽ thu thập các văn thơ ngũ ngôn trên trang web [thivien.net](http://thivien.net), một trang web lớn chuyên tổng hợp các văn thơ gồm đủ các thể loại của Việt Nam.

**\*Note:** Các bạn có thể tải bộ dữ liệu đã được thu thập sẵn tại phần 60.4 và bỏ qua bước này.

The screenshot shows the homepage of thivien.net. At the top, there's a navigation bar with links for TÁC GIÀ, THƠ, THAM GIA, KHÁC, and Đăng nhập. Below the navigation, there's a search bar with the placeholder "Q. 93,054 bài thơ, 4.760 tác giả [Alt+3]" and a "Loan" button. The main content area has two sections: "Nội dung chính" on the left and "Trích diễm" on the right. The "Nội dung chính" section lists statistics: "Thư viện thơ: 93.054 tác phẩm của 4.760 tác giả từ 104 nước, trong đó - 63.269 bài tiếng Việt; - 17.207 bài chữ Hán; - 12.578 bài ngôn ngữ khác". It also lists categories: "Thơ thành viên: 38.048 bài của 1.520 tác giả", "Diễn đàn: 3.558 chủ đề", and "Nội dung chi tiết theo các chuyên mục...". The "Trích diễm" section displays three poems in Vietnamese: "Từ đạo lối về tôi khuất néo", "Em còn nghiêng nón thận thùng che?", and "Đường xa có thấy lòng hiu quanh". Below each poem is a short description: "Dù nắng ngoài kia rực rỡ hè...", followed by "... Vật nắng sân trường tôi bỏ lại (Nhược Thu)". At the bottom, there's a "Tác giả mới" section with links to profiles: Marius Chelaru (Rumani), Eric Patrick Clapton (Anh), and Alain Delorme (Pháp).

Hình 48.3: Trang chủ [thivien.net](http://thivien.net).

Có rất nhiều thư viện Python giúp ta có thể tương tác và trích xuất thông

tin từ trang web một cách dễ dàng. Song ở project này, ta sẽ dùng thư viện [Selenium](#) để thực hiện việc thu thập dữ liệu trên Google Colab. **Các bước thực hiện như sau:**

### Tải thư viện Selenium

Với môi trường máy tính cá nhân, ta đơn giản cài đặt bằng dòng lệnh `pip install selenium webdriver_manager`. Tuy nhiên với môi trường Google Colab, ta sẽ có cách cài đặt phức tạp hơn (chi tiết các bạn coi tại [đây](#)), các bạn hãy copy và chạy đoạn code bên dưới trong Colab:

```
1 %%shell
2 # Ubuntu no longer distributes chromium-browser outside of snap
3 #
4 # Proposed solution: https://askubuntu.com/questions/1204571/how-to-install-chromium-without-snap
5
6 # Add debian buster
7 cat > /etc/apt/sources.list.d/debian.list << "EOF"
8 deb [arch=amd64 signed-by=/usr/share/keyrings/debian-buster.gpg] http://
9 deb [arch=amd64 signed-by=/usr/share/keyrings/debian-buster-updates.gpg]
10 deb [arch=amd64 signed-by=/usr/share/keyrings/debian-security-buster.gpg]
11 EOF
12
13 # Add keys
14 apt-key adv --keyserver keyserver.ubuntu.com --recv-keys DCC9EFCB77E11517
15 apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 648ACFD622F3D138
16 apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 112695A0E562B32A
17
18 apt-key export 77E11517 | gpg --dearmour -o /usr/share/keyrings/debian-
19 apt-key export 22F3D138 | gpg --dearmour -o /usr/share/keyrings/debian-
20 apt-key export E562B32A | gpg --dearmour -o /usr/share/keyrings/debian-
21
22 # Prefer debian repo for chromium* packages only
23 # Note the double-blank lines between entries
24 cat > /etc/apt/preferences.d/chromium.pref << "EOF"
```

```
25 Package: *
26 Pin: release a=eoan
27 Pin-Priority: 500
28
29
30 Package: *
31 Pin: origin "deb.debian.org"
32 Pin-Priority: 300
33
34
35 Package: chromium*
36 Pin: origin "deb.debian.org"
37 Pin-Priority: 700
38 EOF
39
40 # Install chromium and chromium-driver
41 apt-get update
42 apt-get install chromium chromium-driver
43
44 # Install selenium
45 pip install selenium
```

### Import các thư viện cần thiết

```
1 import pandas as pd
2 import os
3 import requests
4 import time
5 import random
6
7 from tqdm import tqdm
8 from selenium import webdriver
9 from selenium.webdriver.chrome.service import Service
10 from selenium.webdriver.common.by import By
11 from selenium.webdriver.support.ui import WebDriverWait
12 from selenium.webdriver.support import expected_conditions as EC
```

### Khởi tạo Selenium driver

Với selenium, ta có thể hiểu đơn giản rằng việc truy cập vào một trang web sẽ được thực hiện như chính chúng ta sử dụng trình duyệt web hằng ngày. Đầu tiên, ta khởi tạo một driver sử dụng đoạn code sau:

```
1 WEBDRIVER_DELAY_TIME_INT = 10
2 TIMEOUT_INT = 10
3 service = Service(executable_path=r"/usr/bin/chromedriver")
4 chrome_options = webdriver.ChromeOptions()
5 chrome_options.add_argument("--headless")
6 chrome_options.add_argument("--no-sandbox")
7 chrome_options.add_argument("--disable-dev-shm-usage")
8 chrome_options.add_argument("window-size=1920x1080")
9 chrome_options.headless = True
10 driver = webdriver.Chrome(service=service, options=chrome_options)
11 driver.implicitly_wait(TIMEOUT_INT)
12 wait = WebDriverWait(driver, WEBDRIVER_DELAY_TIME_INT)
```

Driver trong Selenium đóng vai trò như trình duyệt web, giúp ta thực hiện các thao tác như truy cập vào trang web dựa vào đường dẫn, thao tác chuyển trang...

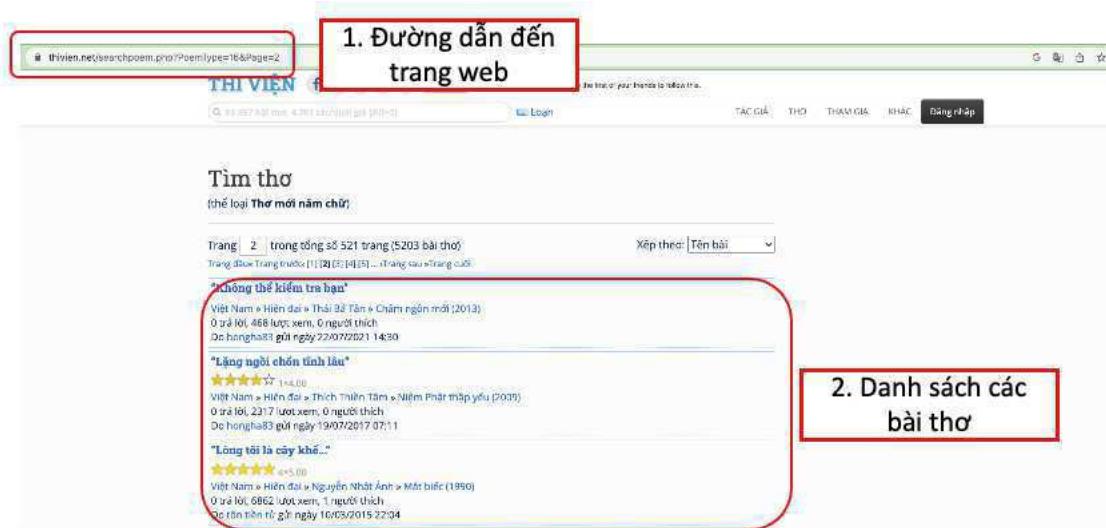
### Phân tích nội dung cần trích xuất

Để việc triển khai code được thuận lợi, ta cần xác định rõ kiến trúc file html của trang web cũng như các thành phần, nội dung mà ta mong muốn trích xuất. Một cách tìm nhanh chóng nhất đó là ta nên tìm đến trang tìm kiếm, từ đó sử dụng selenium để duyệt qua toàn bộ các bài viết được liệt kê trong trang tìm kiếm đó. Trong [thivien.net](#), ta chọn mục **Tìm thơ...** để đến trang này.



Hình 48.4: Minh họa vị trí tìm thơ.

Sau đó, các bạn hãy điền một số thông tin trong bảng **TÌM BÀI THƠ**; ở đây ta chỉ cần quan tâm đến trường thông tin **Thể thơ**: được chọn vào "**Thơ mới năm chữ**". Sau khi bấm tìm kiếm, một trang web với các bài thơ với thể thơ ngũ ngôn xuất hiện.



Hình 48.5: Minh họa trang chứa danh sách các bài thơ năm chữ.

Tại đây, ta đã có được thông tin đường dẫn của trang web (mainpage\_url). Chúng ta sẽ sử dụng driver đã định nghĩa ở trên để truy cập vào trang này bằng lệnh **driver.get()**:

```

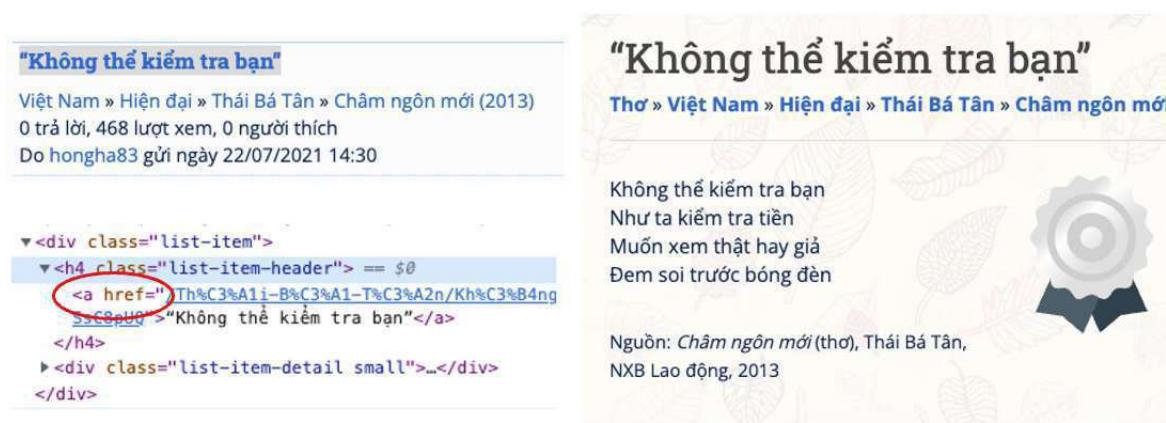
1 for page_idx in tqdm(range(1, 11)):
2 main_url = f"https://www.thivien.net/searchpoem.php?PoemType=16&
3 viewType=1&Country=2&Age []=3&Page=[page_idx]"
4 driver.get(main_url)

```

Nhận thấy đường dẫn trang web có chứa các trường thông tin để ta có thể di chuyển qua lại tại các trang tiếp theo của bảng tìm kiếm ('**Page=2**' tức đang ở trang 2 của bảng). Vì vậy ta có thể tận dụng điều này để tạo một vòng lặp qua từng trang một cách tự động (với đoạn code trên ta sẽ duyệt từ trang thứ 1 đến trang thứ 10).

Ta tiếp tục phân tích các thành phần ta cần trích xuất đối với một trang thơ thông qua việc đọc cấu trúc html của trang web (cấu trúc html của trang web có thể được tìm thấy thông qua tính năng Inspect trên trình duyệt). Nhận thấy, thông tin duy nhất ta quan tâm đó chính là nội dung bài thơ.

Song, để tiện cho các tác vụ sau này nếu có, ta sẽ trích xuất thêm các thông tin khác của bài thơ như Tựa đề, Nguồn...



**"Không thể kiểm tra bạn"**

Việt Nam » Hiện đại » Thái Bá Tân » Châm ngôn mới (2013)  
0 trả lời, 468 lượt xem, 0 người thích  
Do hongha83 gửi ngày 22/07/2021 14:30

```

▼<div class="list-item">
 ▼<h4 class="list-item-header"> == $0
 <a href="Th%C3%A1i-B%C3%A1-T%C3%A2n/Kh%C3%84ng
 %C8%90">"Không thể kiểm tra bạn"
 </h4>
 ▶<div class="list-item-detail small">...</div>
</div>
```

**"Không thể kiểm tra bạn"**

Thơ » Việt Nam » Hiện đại » Thái Bá Tân » Châm ngôn mới

Không thể kiểm tra bạn  
Như ta kiểm tra tiền  
Muốn xem thật hay giả  
Đem soi trước bóng đèn

Nguồn: Châm ngôn mới (thơ), Thái Bá Tân,  
NXB Lao động, 2013

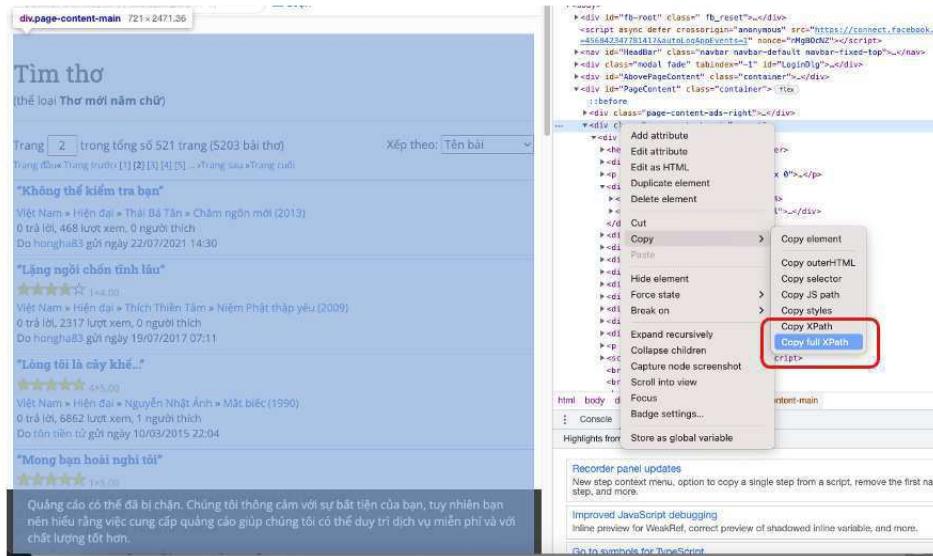
Hình 48.6: Ví dụ về một bài thơ được liệt kê trong bảng tìm kiếm và thẻ HTML của nó.

Như vậy, mỗi bài thơ được liệt kê tại trang tìm kiếm là một thẻ `<div>`, bên trong có chứa đường dẫn đến trang chứa bài thơ chính khi ta click chuột vào. Từ đây, ta dễ dàng truy cập vào mỗi bài thơ bằng cách đọc đường dẫn của thuộc tính ‘`href`’ chứa tại thẻ `<a>` (như hình). Để thực hiện được điều này, đầu tiên ta cần đọc được thẻ html chứa bảng dữ liệu, thực hiện như sau:

```

1 content_tags_xpath = '//*[@@class="page-content container"]//div[@class="page-content-main"]//div[@class="list-item"]'
2 content_tags = driver.find_elements(By.XPATH, content_tags_xpath)
```

Trong Selenium, có nhiều cách để xác định và đọc thẻ html từ trang web thông qua hai phương thức `driver.find_element()` (tìm một thẻ khớp) và `driver.find_elements()` (tìm nhiều thẻ khớp) (chi tiết tại [đây](#)) song tìm kiếm bằng XPATH là một cách nhanh chóng nhất. Ở đây, ta quan tâm đến các thẻ div chứa thông tin bài thơ nên ta sẽ dùng `find_elements()` để tìm toàn bộ các thẻ này. Và với danh sách các thẻ của từng bài thơ (`content_tags`), ta đã có thể truy cập vào nội dung chi tiết của từng bài thơ.



Hình 48.7: Tìm XPATH của một thẻ HTML trên trình duyệt.

Khi gom hết tất cả các dữ kiện trên, ta có thể tạo một hàm dùng để bóc tách các trang web thơ với input là số thứ tự của trang danh sách thơ:

```

1 def extract_poem_links(driver, page_idx):
2 main_url = f"https://www.thivien.net/searchpoem.php?PoemType=16&
3 viewType=1&Country=2&Age []=3&Page=[&
4 page_idx]"
5
6 driver.get(main_url)
7 time.sleep(random.uniform(3, 5))
8
9 content_tags_xpath = '//*[@@class="page-content container"]//div[@class=&
10 "page-content-main"]//div[@class="list-item"]'
11
12 content_tags = driver.find_elements(By.XPATH, content_tags_xpath)
13 poem_links = []
14 for tag in content_tags:
15 try:
16 link_element = tag.find_element(By.XPATH, './h4[@class="list-
17 item-header"]/a')
18 poem_title = link_element.text
19 poem_url = link_element.get_attribute("href")
20 poem_links.append({"title": poem_title, "url": poem_url})
21 except Exception as e:
22 print(f"Error extracting link: {e}")

```

```

17 continue
18 return poem_links

```

## Thực hiện trích xuất nội dung thơ

Cuối cùng, với từng thẻ html của trang thơ, ta sẽ thực hiện trích xuất các thông tin mà ta đã xác định (gồm Nội dung bài thơ, Tựa đề, Nguồn) của bài thơ tương ứng. Các thông tin này sẽ được lưu thành một dictionary và đẩy vào một list lưu trữ chung (**datasets**). Trước khi duyệt qua từng trang danh sách thơ, ta xây dựng sẵn một hàm chuyên dùng để tiền xử lý các nội dung thơ sau khi thu thập được. Chức năng của hai hàm dưới đây chỉ dùng để loại bỏ một số nội dung không liên quan đến thơ như các thẻ HTML xen lẫn thơ:

```

1 def clean_poem_html(html):
2 html = re.sub(r"<img.*?>", "", html, flags=re.IGNORECASE)
3 html = re.sub(r"<i>.*?</i>", "", html, flags=re.IGNORECASE | re.DOTALL
4)
5 html = re.sub(r"(.*)(?!
\s*(?:
\s*){2,})", r"\1", html,
6 flags=re.IGNORECASE)
7 html = re.sub(r"
\s*/?>", "\n", html, flags=re.IGNORECASE)
8 html = re.sub(r"</?p>", "", html, flags=re.IGNORECASE)
9
10 return html.strip()
11
12
13 def process_poem_content(html, poem_src, poem_url, default_title=""):
14 cleaned = clean_poem_html(html)
15
16 pattern = re.compile(r"(.*)\s*\n{2,}", flags=re.IGNORECASE)
17 matches = list(pattern.finditer(cleaned))
18
19 poems = []
20 if matches:
21 for i, match in enumerate(matches):
22 title = match.group(1).strip()
23 start = match.end()
24 end = matches[i+1].start() if i + 1 < len(matches) else len(
25 cleaned)
26 content = cleaned[start:end].strip("\n")
27 poems.append({
28 "title": title,
29 "content": content,
30 "source": poem_src,
31 })
32
33 return poems

```

```

27 "url": poem_url
28 })
29 else:
30 poems.append({
31 "title": default_title,
32 "content": cleaned,
33 "source": poem_src,
34 "url": poem_url
35 })
36 return poems

```

Cuối cùng, ta định nghĩa hai hàm dùng để trích xuất nội dung của một đường dẫn chứa thơ năm chữ cũng như xây dựng hàm duyệt qua từng trang chứa thơ như sau:

```

1 def scrape_poem(driver, poem_url):
2 driver.get(poem_url)
3 time.sleep(random.uniform(3, 5))
4
5 poem_content_tag = WebDriverWait(driver, 10).until(
6 EC.visibility_of_element_located((By.CSS_SELECTOR, "div.poem-
7 content")))
8
9 html_content = poem_content_tag.get_attribute("innerHTML")
10
11 try:
12 poem_src_tag = WebDriverWait(driver, 10).until(
13 EC.presence_of_element_located((By.XPATH, '//div[@class="small
14 "]')))
15 poem_src = poem_src_tag.text
16 except Exception:
17 poem_src = ""
18
19 return process_poem_content(html_content, poem_src, poem_url)
20
21 def scrape_poems(driver, num_pages=10):
22 datasets = []
23 for page_idx in tqdm(range(1, num_pages + 1)):
24 poem_links = extract_poem_links(driver, page_idx)
25 for poem in poem_links:
26 poem_url = poem["url"]
27 try:
28 poems = scrape_poem(driver, poem_url)

```

```

29 datasets.extend(poems)
30 except Exception as e:
31 print(f"Error processing {poem_url}: {e}")
32 continue
33 return datasets

```

Khi mọi hàm đã sẵn sàng, việc còn lại của chúng ta là thực thi hàm `scrape_poems()` để bắt đầu quá trình thu thập thơ:

```

1 datasets = scrape_poems(driver, num_pages=10)
2 driver.quit()

```

Các kỹ thuật tại bước này đều xoay quanh việc tìm kiếm bằng XPATH kèm theo một số logic python phát sinh trong quá trình thử nghiệm code, các bạn nên đọc qua từng dòng cũng như kiểm nghiệm lại giá trị các biến dữ liệu để có thể hiểu tường tận các bước trích xuất trên. Thông qua đó, các bạn cũng có thể áp dụng để thu thập dữ liệu từ các trang web khác theo nhu cầu của các bạn (Ví dụ: thu thập các bài thơ lục bát tại [thivien.net...](https://www.thivien.net/))

### Lưu bộ dữ liệu thành file .csv

Sau khi hoàn tất giai đoạn thu thập trên, ta sẽ có một danh sách các dictionary (record) chứa thông tin của một bài thơ. Lúc này, để thuận tiện trong việc lưu trữ, ta sẽ lưu danh sách này thành một file .csv.

	<b>title</b>	<b>content</b>	<b>source</b>	<b>url</b>
0	"Bạn xấu như chiếc bóng"	Bạn xấu như chiếc bóng\nCứ bám riết theo anh\n...	[Thông tin 1 nguồn tham khảo đã được ấn]	<a href="https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%A1i-%C3%8D...">https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%A1i-%C3%8D...</a>
1	"Cái làm ta hạnh phúc"	Cái làm ta hạnh phúc\nThực ra cũng chẳng nhiều...	[Thông tin 1 nguồn tham khảo đã được ấn]	<a href="https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%A1i-%C3%8D...">https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%A1i-%C3%8D...</a>
2	"Chiều vừa xốp trên tay"	Chiều vừa xốp trên tay\nChợt nghe thoáng ong b...	[Thông tin 1 nguồn tham khảo đã được ấn]	<a href="https://www.thivien.net/L%C3%A2m-Huy-Nhu%E1%BA...">https://www.thivien.net/L%C3%A2m-Huy-Nhu%E1%BA...</a>
3	"Chơi thân không có nghĩa"	Chơi thân không có nghĩa\nKhông cãi nhau bao g...	[Thông tin 1 nguồn tham khảo đã được ấn]	<a href="https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%A1i-%C3%8D...">https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%A1i-%C3%8D...</a>
4	"Có thể buồn chút ít"	Có thể buồn chút ít\nMột mình, không người yêu...	[Thông tin 1 nguồn tham khảo đã được ấn]	<a href="https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%A1i-%C3%8D...">https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%A1i-%C3%8D...</a>
...	...	...	...	...

Hình 48.8: Dữ liệu thu thập được trong bảng dữ liệu.

```

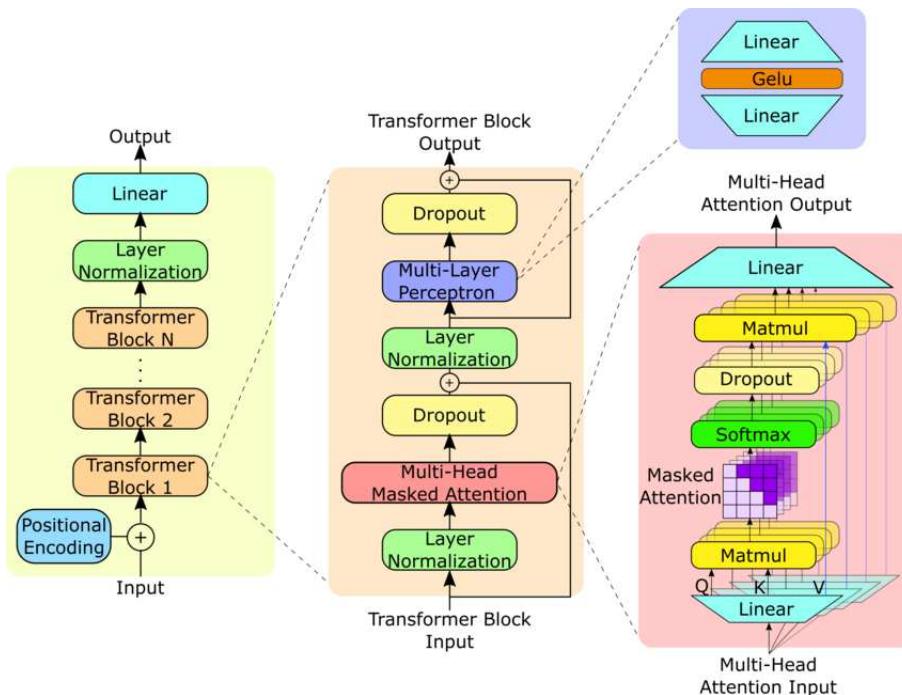
1 df = pd.DataFrame(datasets)
2 df.to_csv("poem_dataset.csv", index=True)

```

Trong phần code ví dụ trên, ta chỉ cài đặt để thu thập 10 trang đầu tiên trong trang tìm kiếm (vì trang web giới hạn số trang hiển thị).

### 48.2.2 Xây dựng mô hình sinh thơ

Sau khi hoàn tất quá trình thu thập dữ liệu, chúng ta sẽ thực hiện huấn luyện (fine-tuning) mô hình GPT-2. GPT-2 (Generative Pre-trained Transformer 2) là một mô hình ngôn ngữ lớn phiên bản thứ 2 trong chuỗi các mô hình họ GPT được phát triển bởi OpenAI. GPT-2 được xây dựng dựa trên kiến trúc Transformer Decoder-only, các bạn có thể coi ảnh minh họa kiến trúc của GPT-2 ở hình dưới đây:



Hình 48.9: Kiến trúc mô hình của GPT-2. Nguồn: [link](#).

Ở phần này, chúng ta sẽ sử dụng mô hình GPT2 để thực hiện fine-tuning cho mục đích sinh thơ tiếng Việt. Theo đó, các bước thực hiện như sau:

#### Tải các thư viện cần thiết

```
1 !pip install -qq datasets evaluate transformers[sentencepiece]
2 !pip install -qq accelerate
3 !apt install git-lfs
```

## Import các thư viện cần thiết

Chúng ta sẽ sử dụng thư viện HuggingFace với 2 module quan trọng là GPT2Tokenizer và GPT2LMHeadModel.

```
1 import os
2 import math
3 import torch
4 import pandas as pd
5
6 from transformers import GPT2Tokenizer, GPT2LMHeadModel
7 from transformers import DataCollatorForLanguageModeling
8 from transformers import TrainingArguments, Trainer
9 from huggingface_hub import notebook_login
10 from datasets import Dataset
```

## Load bộ dữ liệu

Với bộ dữ liệu đã thu thập được, chúng ta sẽ tiến hành đọc file .csv lên như sau:

```
1 DATASET_PATH = "poem_final.csv"
2 df = pd.read_csv(DATASET_PATH)
3 df
```

## Chuẩn bị bộ dữ liệu

Hiện tại, đoạn thơ ta tách được có cấu trúc như sau:

Áo chiều nào còn vương  
 Người ấy đến rất nhớ  
 Từ đất đen đứng dậy  
 Vào cuộc đời đáng yêu

Một khổ thơ

Người đi trong nắng mới  
 Gió hát khúc yêu đời  
 Mùa xuân xanh lá thắm  
 Hy vọng mãi không rời.

Hình 48.10: Cấu trúc của một mẫu dữ liệu (một bài thơ năm chữ).

Một bài thơ có thể có nhiều khổ thơ (part), một khổ thơ sẽ gồm 4 dòng thơ, mỗi dòng gồm 5 chữ. Để giảm độ phức tạp của bài toán, chúng ta sẽ coi mỗi khổ thơ là một data sample, và dùng chúng cho việc huấn luyện mô hình. Đầu tiên, ta xây dựng hàm tách nội dung thơ của một mẫu dữ liệu thành các danh sách chứa 4 dòng thơ:

```

1 def split_content(content):
2 samples = []
3
4 poem_parts = content.split("\n\n")
5 for poem_part in poem_parts:
6 poem_in_lines = poem_part.split("\n")
7 if len(poem_in_lines) == 4:
8 samples.append(poem_in_lines)
9
10 return samples
11
12 df["content"] = df["content"].apply(lambda x: split_content(x))
13 df

```

Nhận thấy nội dung cột content của mỗi hàng dữ liệu là một list chứa các sublist. Ta sẽ thực hiện tách các sublist này thành một hàng trong bảng dữ liệu mới, coi như là một sample mới trong bộ dữ liệu. Cách thực hiện như sau:

```

1 df_exploded = df.explode("content")
2 df_exploded.reset_index(drop=True, inplace=True)
3 df_exploded = df_exploded.dropna(subset=["content"])
4 df_exploded

```

Hàm `df.explode()` sẽ giúp ta tách các phần tử trong một list thành các hàng mới. Khi thực thi xong đoạn code trên, ta có bảng dữ liệu mới như sau:

	Unnamed: 0	title	content	source	url
0	0	"Cái làm ta hạnh phúc"	[Cái làm ta hạnh phúc, Thực ra cũng chẳng n...]	Nguồn: Châm ngôn mới (thơ), Thái Bá Tân, NXB L...	<a href="https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%ADn-NXB-L...">https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%ADn-NXB-L...</a>
1	0	"Cái làm ta hạnh phúc"	[Rồi thêm chút công việc, Cho ta làm hàng ngày...]	Nguồn: Châm ngôn mới (thơ), Thái Bá Tân, NXB L...	<a href="https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%ADn-NXB-L...">https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%ADn-NXB-L...</a>
2	1	"Chiều vừa xốp trên tay"	[Chiều vừa xốp trên tay, Chợt nghe thoáng ong ...]	Nguồn: Lâm Huy Nhuận, Chiều có thật (thơ), NXB...	<a href="https://www.thivien.net/L%C3%A2m-Huy-Nhu%E1%BA...">https://www.thivien.net/L%C3%A2m-Huy-Nhu%E1%BA...</a>
3	1	"Chiều vừa xốp trên tay"	[Ớt đỏ sao cù đỗ, Táo chín cho thật vàng, Em đ...	Nguồn: Lâm Huy Nhuận, Chiều có thật (thơ), NXB...	<a href="https://www.thivien.net/L%C3%A2m-Huy-Nhu%E1%BA...">https://www.thivien.net/L%C3%A2m-Huy-Nhu%E1%BA...</a>
4	2	"Dưới giàn hoa thiên lý..."	[Dưới giàn hoa thiên lý, Một mình anh đang ngồi...]	Nguồn: Nguyễn Nhật Ánh, Mắt biếc, NXB Trẻ, 2004	<a href="https://www.thivien.net/Nguy%E1%BB%85n-Nh%E1%BA...">https://www.thivien.net/Nguy%E1%BB%85n-Nh%E1%BA...</a>
...	...	...	...	...	...

Hình 48.11: Bảng dữ liệu sau khi sử dụng hàm `explode` của pandas để tách các khổ thơ thành các hàng dữ liệu mới. Có thể nhận thấy số hàng trong bảng dữ liệu đã tăng lên.

Ta cần nội dung thơ (giá trị của cột `content`) phải ở dạng string. Vì vậy, ta sẽ thực hiện convert nội dung `content` sang string như sau:

```

1 df_exploded["content"] = df_exploded["content"].apply(lambda x: "\n".join(
 x))
2 df_exploded

```

	Unnamed: 0	title	content	source	url
0	0	"Cái làm ta hạnh phúc"	Cái làm ta hạnh phúc\nThực ra cũng chẳng nhiều...	Nguồn: Châm ngôn mới (thơ), Thái Bá Tân, NXB L...	<a href="https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%83n-NXB-L...">https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%83n-NXB-L...</a>
1	0	"Cái làm ta hạnh phúc"	Rồi thêm chút công việc\nCho ta làm hàng ngày...	Nguồn: Châm ngôn mới (thơ), Thái Bá Tân, NXB L...	<a href="https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%83n-NXB-L...">https://www.thivien.net/Th%C3%A1i-B%C3%A1-T%C3%83n-NXB-L...</a>
2	1	"Chiều vừa xốp trên tay"	Chiều vừa xốp trên tay\nChợt nghe thoáng ơng b...	Nguồn: Lâm Huy Nhuận, Chiều có thật (thơ), NXB...	<a href="https://www.thivien.net/L%C3%A2m-Huy-Nhu%E1%BA...">https://www.thivien.net/L%C3%A2m-Huy-Nhu%E1%BA...</a>
3	1	"Chiều vừa xốp trên tay"	Ớt đó sao cứ đòn\nTáo chín cho thật vàng\nEm đẹ...	Nguồn: Lâm Huy Nhuận, Chiều có thật (thơ), NXB...	<a href="https://www.thivien.net/L%C3%A2m-Huy-Nhu%E1%BA...">https://www.thivien.net/L%C3%A2m-Huy-Nhu%E1%BA...</a>
4	2	"Dưới giàn hoa thiên lý"	Dưới giàn hoa thiên lý\nMột mình anh đang ngồi...	Nguồn: Nguyễn Nhật Ánh, Mắt biếc, NXB Trẻ, 2004	<a href="https://www.thivien.net/Nguy%E1%BB%85n-Nh%E1%B...">https://www.thivien.net/Nguy%E1%BB%85n-Nh%E1%B...</a>
...	...	...	...	...	...

Hình 48.12: Bảng dữ liệu sau khi chuyển đổi nội dung cột content sang dạng string.

Với DataFrame đã chuẩn bị xong, chúng ta sẽ đổi dạng dữ liệu pandas này sang HuggingFace dataset để thuận tiện trong việc sử dụng thư viện:

```
1 TEST_SIZE = 0.1
2 poem_dataset = Dataset.from_pandas(df_exploded)
3 poem_dataset = poem_dataset.train_test_split(test_size=TEST_SIZE)
```

```
1 poem_dataset = Dataset.from_pandas(df_exploded)
2 poem_dataset

Dataset({
 features: ['Unnamed: 0', 'title', 'content', 'source', 'url', '__index_level_0__'],
 num_rows: 441
})

1 TEST_SIZE = 0.1
2 poem_dataset = poem_dataset.train_test_split(test_size=TEST_SIZE)
3 poem_dataset

DatasetDict({
 train: Dataset({
 features: ['Unnamed: 0', 'title', 'content', 'source', 'url', '__index_level_0__'],
 num_rows: 396
 })
 test: Dataset({
 features: ['Unnamed: 0', 'title', 'content', 'source', 'url', '__index_level_0__'],
 num_rows: 45
 })
})
```

Hình 48.13: Bảng dữ liệu sau khi được đổi sang HuggingFace dataset.

## Tiền xử lý dữ liệu

Với bộ dữ liệu thơ đã sẵn sàng, chúng ta bắt đầu quy trình tiền xử lý bộ dữ liệu để chuẩn bị cho việc huấn luyện mô hình. Đầu tiên, ta khai báo tokenizer:

```
1 MODEL_NAME = "danghuy1999/gpt2-viwiki"
2
3 tokenizer = GPT2Tokenizer.from_pretrained(MODEL_NAME)
```

Sau đó, xây dựng hàm chạy tokenization cho mỗi sample và thực thi chúng lên bộ dữ liệu:

```
1 tokenizer.pad_token = tokenizer.eos_token
2 MAX_SEQ_LEN = 100
3
4 def preprocess_function(row):
5 return tokenizer(
6 row["content"],
7 max_length=MAX_SEQ_LEN,
8 padding="max_length",
9 truncation=True
10)
11
12 tokenized_poem_dataset = poem_dataset.map(
13 preprocess_function,
14 batched=True,
15 num_proc=4,
16 remove_columns=poem_dataset["train"].column_names,
17)
```

Khi huấn luyện mô hình ngôn ngữ trong HuggingFace, chúng ta sẽ khai báo một instance từ class DataCollatorForLanguageModeling. Việc này nhằm giúp HuggingFace hỗ trợ chúng ta việc batching dữ liệu để việc huấn luyện mô hình ngôn ngữ trở nên hiệu quả hơn.

```
1 data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer,
2 mlm=False)
```



Hình 48.14: Ảnh minh họa việc đoán từ tiếp theo dựa trên các từ trước đó.  
Nguồn: [link](#).

## Huấn luyện mô hình

Với bộ dữ liệu đã được tiền xử lý, ta sẽ bắt đầu việc huấn luyện mô hình.  
Đầu tiên, ta load pre-trained model GPT-2:

```
1 model = GPT2LMHeadModel.from_pretrained(MODEL_NAME)
```

Sau đó, khai báo một vài config trong việc huấn luyện mô hình:

```
1 training_args = TrainingArguments(
2 output_dir="gpt2_viet_poem_generation",
3 save_strategy="epoch",
4 learning_rate=2e-5,
5 num_train_epochs=10,
6 weight_decay=0.01,
7 fp16=True
8)
```

Cuối cùng, ta thực thi trainer để tiến hành training:

```
1 trainer = Trainer(
2 model=model,
3 args=training_args,
4 train_dataset=tokenized_poem_dataset["train"],
5 eval_dataset=tokenized_poem_dataset["test"],
6 data_collator=data_collator,
7 tokenizer=tokenizer
8)
```

```

9
10 trainer.train()

```

Như vậy, sau khi hoàn thành các bước trên, ta đã hoàn tất quá trình huấn luyện một mô hình sinh thơ tiếng Việt.

## Inference

Để sử dụng mô hình này, chúng ta có thể đưa lên HuggingFace hub và gọi mô hình xuống để sử dụng hoặc chúng ta có thể làm theo cách sau:

```

1 prompt = "Học học nữa học mãi\n"
2 device = "cuda" if torch.cuda.is_available() else "cpu"
3 inputs = tokenizer(prompt, return_tensors="pt").input_ids.to(device)
4 outputs = model.generate(
5 inputs,
6 max_new_tokens=50,
7 do_sample=True,
8 top_k=50,
9 top_p=0.95,
10 temperature=0.8,
11 repetition_penalty=1.2
12)
13 results = tokenizer.batch_decode(outputs, skip_special_tokens=True)
14 results = results[0]
15 print()
16 for line in results.split("\n"):
17 print(line)

```

Demo về một đoạn thơ năm chữ được tạo sinh từ mô hình GPT-2 sau khi thực hiện fine-tuning của chúng ta:

*Học học nữa học mãi  
Hàng trăm ngàn đêm dài  
Nhưng, không ai quên nghĩ  
Quanh ta thấy chuyện gì?*

— Fine-tuned GPT-2 on a tiny Vietnamese Five-Word Poetry Dataset

Với mô hình xây dựng được, ta có thể triển khai thành một ứng dụng web demo chương trình cho phép tạo sinh một bài thơ bắt đầu từ một vài từ cho

trước. Web demo còn được cài đặt để ta có thể điều chỉnh các thông số kỹ thuật có thể làm thay đổi kết quả tạo sinh của mô hình GPT-2, giúp người dùng có thể thấy được đa dạng các kết quả hơn từ mô hình. Thông tin về web demo có thể được tìm thấy tại phần 60.4

The screenshot shows the AI VIET NAM Vietnamese Poem Generation web application. At the top left is the AI VIET NAM logo, featuring a green circle with the letters 'AI' in white. To its right is the text 'AI VIET NAM'. Below the logo is a large, bold title 'Vietnamese Poem Generation'. A subtext below the title states: 'The used model is GPT-2 trained on Vietnamese poems: thangduong0509/gpt2\_viet\_poem\_generation'. There is a dropdown menu labeled 'Instructions' with a downward arrow icon. Below it is a text input field containing the starting text 'Con song que toi dep'. To the right of the input field is a 'Generate Poem' button. On the left side, there are three sliders for generating parameters: 'Max Output Tokens' set to 75, 'Temperature' set to 0.80, and 'Top-k' set to 50. To the right of these sliders is a section titled 'Generated Poem:' which displays the generated poem: 'Con song que toi dep', 'Tiếng mẹ người đâu xưa', 'nguyễn hoa biển lạnh?', and 'Đôi mắt dài vè!'. The entire interface is contained within a light gray frame.

Hình 48.15: Ảnh minh họa web demo cho chương trình tạo sinh thơ tiếng Việt.

### 48.3 Câu hỏi trắc nghiệm

1. Mô hình Text Generation là?
  - a) Mô hình sinh chữ từ ảnh.
  - b) Mô hình sinh chữ từ video.
  - c) Mô hình sinh chữ từ một input nào đó.
  - d) Mô hình sinh chữ từ bản ghi âm thanh.
2. Ứng dụng nào sau đây thuộc về Text Generation?
  - a) Image Captioning.
  - b) Text Summarization.
  - c) Automatic Speech Recognition.
  - d) Tất cả các phương án trên.
3. Mục tiêu của bài toán Text Generation là?
  - a) Copy văn bản đầu vào.
  - b) Tạo văn bản mới dựa trên dữ liệu đầu vào.
  - c) Sửa văn bản đầu vào.
  - d) Không phương án nào đúng.
4. Các thách thức trong bài Text Generation?
  - a) Văn bản đầu ra có ngữ pháp đúng.
  - b) Văn bản đầu ra có nghĩa.
  - c) Văn bản đầu ra phải mạch lạc.
  - d) Tất cả các phương án trên.
5. Khi mô hình được thiết kế để dự đoán từ tiếp theo dựa trên một chuỗi các từ trước đó, ta gọi bài toán này là gì?
  - a) Causal Language Modeling.
  - b) Masked Language Modeling.
  - c) Sequence to Sequence.

- d) Denoising.
6. Mô hình GPT2 được xây dựng dựa theo kiến trúc nào?
- a) Transformer Encoder-Decoder.
  - b) Transformer Encoder-only.
  - c) Transformer Decoder-only.
  - d) Long Short-Term Memory.
7. Selenium là?
- a) Ngôn ngữ lập trình.
  - b) Trình duyệt web.
  - c) Một thư viện trong Python.
  - d) Thiết kế mô hình học sâu.
8. Selenium thường được sử dụng trong việc?
- a) Thu thập dữ liệu trên web.
  - b) Thiết kế giao diện web.
  - c) Tối ưu siêu tham số mô hình học sâu.
  - d) Thiết kế mô hình học sâu.
9. Dòng lệnh nào sau đây dùng để truy cập vào một trang web trong Selenium với đường dẫn cho trước:
- a) `driver.get()`
  - b) `driver.switch_to_window()`
  - c) `driver.execute_script()`
  - d) `driver.close()`
10. Dòng lệnh nào sau đây dùng để tìm một thẻ HTML trong Selenium:
- a) `driver.get()`
  - b) `driver.find_element()`
  - c) `driver.back()`
  - d) `driver.execute_async_script()`

## 48.4 Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
2. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
3. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
4. **Demo:** Web demo và mã nguồn của ứng dụng có thể được truy cập tại [đây](#).
5. **Rubric:**

Mục	Kiến Thức	Đánh Giá
II.	<ul style="list-style-type: none"> <li>- Kiến thức về crawl data.</li> <li>- Kiến thức về bài toán Text Generation.</li> <li>- Kiến thức về thư viện Selenium để crawl dữ liệu và thư viện HuggingFace để fine-tune mô hình ngôn ngữ.</li> </ul>	<ul style="list-style-type: none"> <li>- Nắm được cách sử dụng thư viện Selenium để lấy dữ liệu từ một trang web.</li> <li>- Nắm được cách sử dụng thư viện HuggingFace để fine-tune mô hình ngôn ngữ cho bài toán sinh tho.</li> </ul>
III.	<ul style="list-style-type: none"> <li>- Các kiến thức cơ bản về bài toán Text Generation.</li> <li>- Các kiến thức cơ bản về thư viện Selenium trong Python.</li> <li>- Một số phương thức, hàm cơ bản trong Selenium.</li> </ul>	<ul style="list-style-type: none"> <li>- Hiểu được các khái niệm cơ bản về Selenium.</li> <li>- Cách sử dụng một số chức năng cơ bản về Selenium.</li> <li>- Hiểu được các khái niệm cơ bản của bài toán Text Generation.</li> </ul>

- *Hết* -

## **Phần XI**

# **Module 11: Những mô hình tạo sinh**

# Chương 49

## Style Transfer

### 49.1 Style Transfer

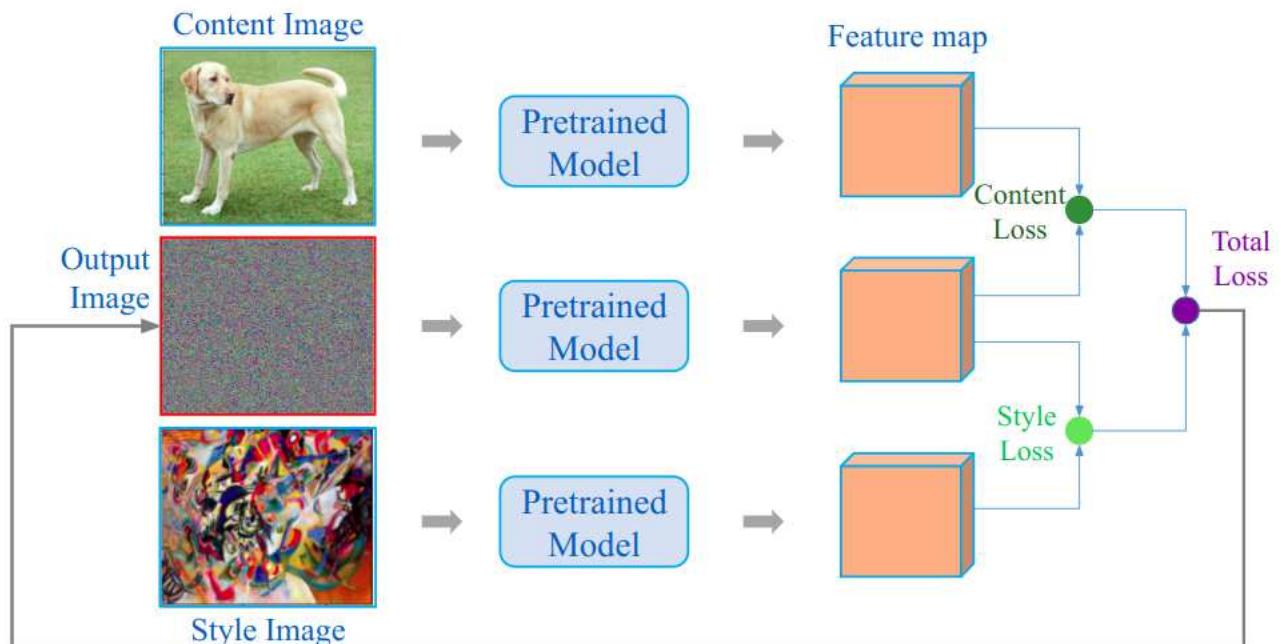
#### 1. Style Transfer:

Style transfer là sự kết hợp giữa trí tuệ nhân tạo và nghệ thuật (art), đây là lĩnh vực thú vị nơi mà thẩm mỹ của nghệ thuật hình ảnh được kết hợp với nội dung số hóa. Quá trình này bao gồm việc áp dụng style (phong cách) nghệ thuật từ một hình ảnh và áp dụng nó lên nội dung (content) của một hình ảnh khác, từ đó tạo ra một hình ảnh mới thể hiện bản chất của cả hai.



Hình 49.1: Style Transfer

## 2. Neural Representations of Content and Style:



Hình 49.2: Style Transfer Pipeline

Bài toán này dựa trên việc sử dụng Mạng Nơ-ron Tích chập (CNNs), đặc biệt là các mạng được huấn luyện trước (pretrain) như VGGNet, có khả năng nắm bắt các đặc trưng của hình ảnh theo từng cấp bậc - từ các cạnh và kết cấu cơ bản đến các biểu diễn nội dung phức tạp hơn.

**Content Representation:** Nội dung (content) của một hình ảnh được học bởi feature map của một số layer trong mạng, thường là các lớp sâu hơn nơi mạng đã trừu tượng hóa khỏi dữ liệu pixel thô và thay vào đó biểu diễn các đặc điểm cấp cao hơn như hình dạng và đối tượng.

**Style Representation:** Phong cách (style) của một hình ảnh được học bởi các tương quan giữa các đặc điểm trong các layer khác nhau của mạng. Điều này được biểu diễn toán học bởi ma trận Gram, nắm bắt được phân phối của các đặc điểm

### 3. Loss Functions:

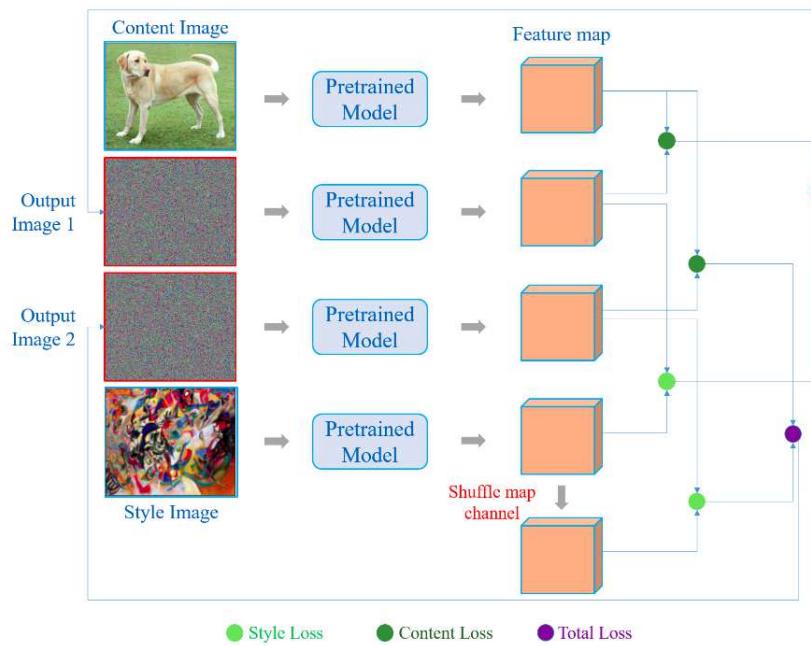
**Content Loss:** Đo lường mức độ nội dung của hình ảnh được tạo ra từ model sai lệch so với nội dung của hình ảnh nội dung gốc . Nó thường được tính toán bằng MSE của hình ảnh nội dung và hình ảnh được tạo ra trong một hoặc nhiều layer của CNN.

**Style Loss:** Đo lường sự khác biệt về phong cách giữa hình ảnh được tạo ra và hình ảnh tham khảo phong cách . MSE được sử dụng giữa các ma trận Gram của các biểu diễn phong cách của hình ảnh được tạo ra và hình ảnh phong cách qua nhiều layer của CNN.

**Total Loss:** Là hàm kết hợp content và style loss, thường là tổng hoặc kết hợp với các tham số để điều chỉnh lượng loss phù hợp

### 4. Multi-modal Style Transfer:

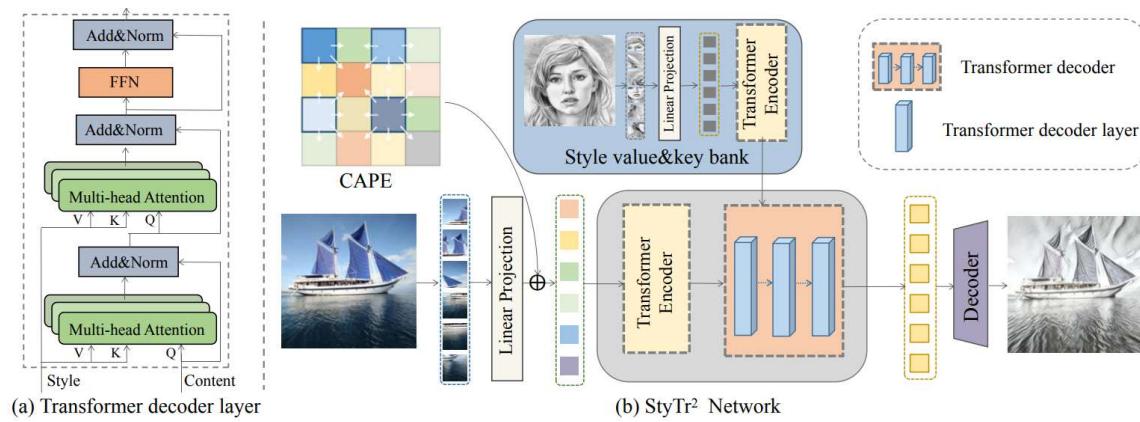
Bằng cách biến đổi feature map của ảnh phong cách (style), ta có thể tạo ra nhiều phong cách khác nhau chỉ với một ảnh phong cách đầu vào



Hình 49.3: Multi-modal Style Transfer

### 5. Image Style Transfer With Transformers:

Đây là phương pháp sử dụng transformer model cho chuyển đổi phong cách hình ảnh. Phương pháp này giải quyết những hạn chế của CNN trong việc nắm bắt thông tin toàn cục. Nó sử dụng hai transformer encoder cho nội dung và phong cách tương ứng, và một multi-layer transformer decoder để tạo phong cách cho nội dung. Một Content-Aware Positional Encoding (CAPE) được đưa vào để cải thiện positional encoding cho style transfer.



Hình 49.4: Style Transfer with Transforrmer

## 49.2 Bài Tập và Gợi Ý

Input đầu vào cho bài toán style transfer thông thường là ảnh nội dung (content) và ảnh (style) và sinh ra ảnh output. Ma trận Gram được dùng ở hàm loss để xác định mức độ ảnh output chứa các đặc trưng có trong ảnh phong cách (style).

Trong bài tập này các bạn sẽ xây dựng các biến thể khác của bài toán style transfer. Chúng ta sẽ tập trung vào các feature của ảnh style để sáng tạo ra một biến thể phong cách (style) mới bằng cách kết hợp feature của 2 phong cách (style) hoặc tác động trực tiếp biến đổi vào feature của phong cách (style) để tạo ra một phong cách mới.

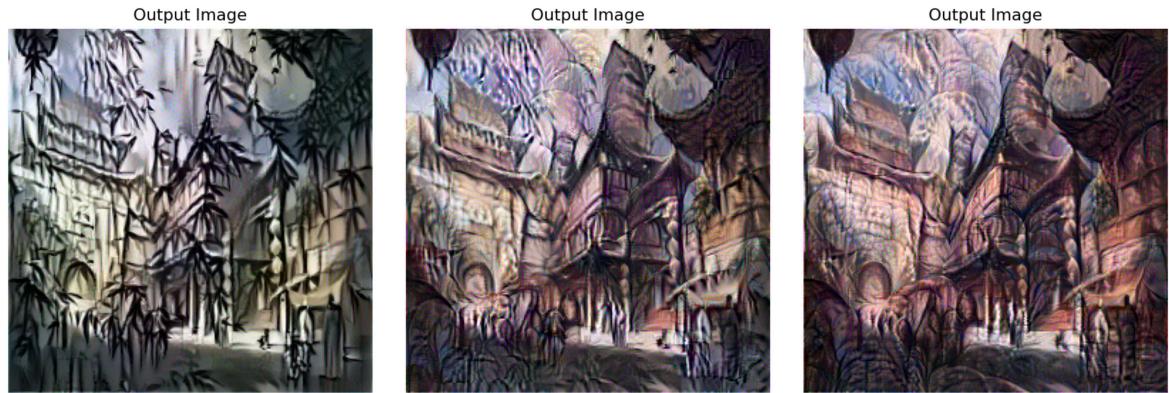
**Bài 1:** Dual-style Transfer, input nhận 1 ảnh content và 2 ảnh style. Chúng ta cần thực hiện trích xuất feature của cả 2 ảnh style, sau đó combine theo tỉ lệ  $1/4$  và  $3/4$  để tạo ra style mới. Cuối cùng transfer style mới này lên ảnh content để tạo ra output cuối cùng

Ảnh đầu vào của bài tập 1, ảnh trái là ảnh style 1, ảnh giữa là ảnh style 2, và ảnh phải là ảnh content



Hình 49.5

Ảnh trái là ảnh khi transfer theo style 1, ảnh giữa là ảnh khi transfer theo  $1/2$  style 1 và  $1/2$  style 2, và ảnh phải là ảnh output của bài tập 1 khi transfer theo  $1/4$  style1 +  $3/4$  style 2.

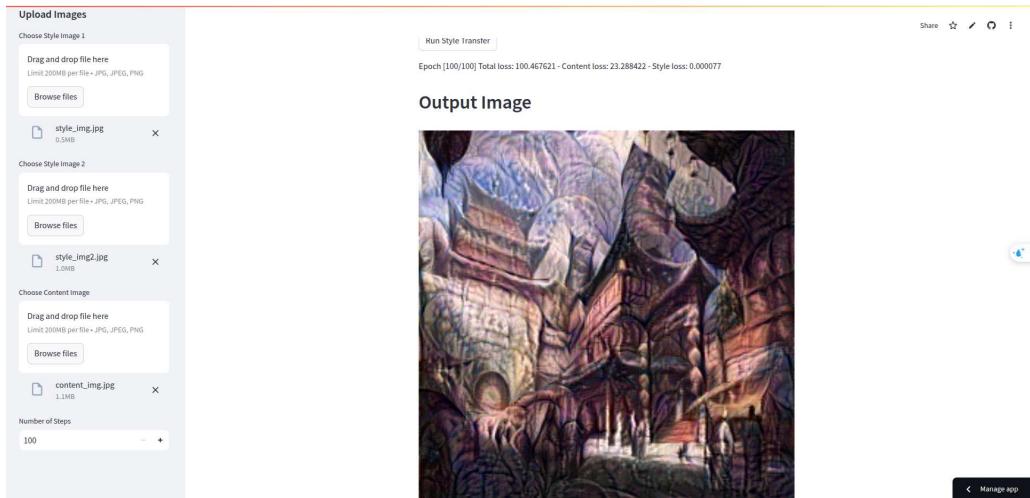


Hình 49.6: Bài 1 (a) input và (b) output (chỉ ảnh bên phải)

**Deploy:** Các bạn có thể tham khảo kết quả từ chương trình được deploy trên streamlit:

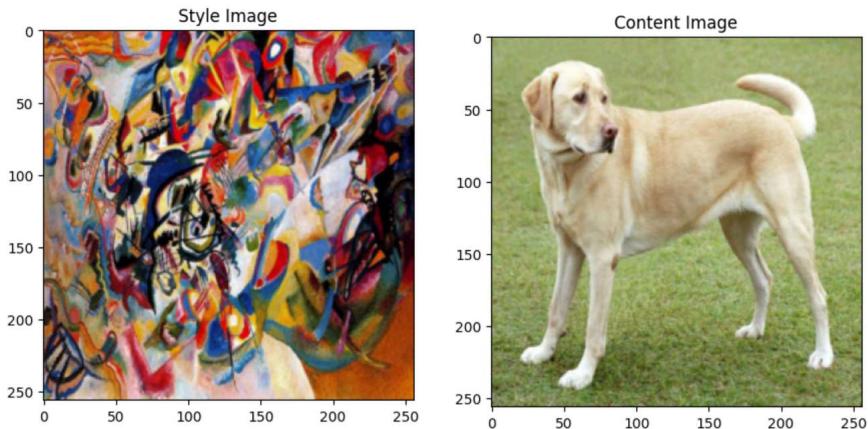
- [Github](#)
- Ảnh minh họa khi deploy bài tập 1 trên streamlit

Hình 49.7: Bài 1 upload style1, style2 và content image



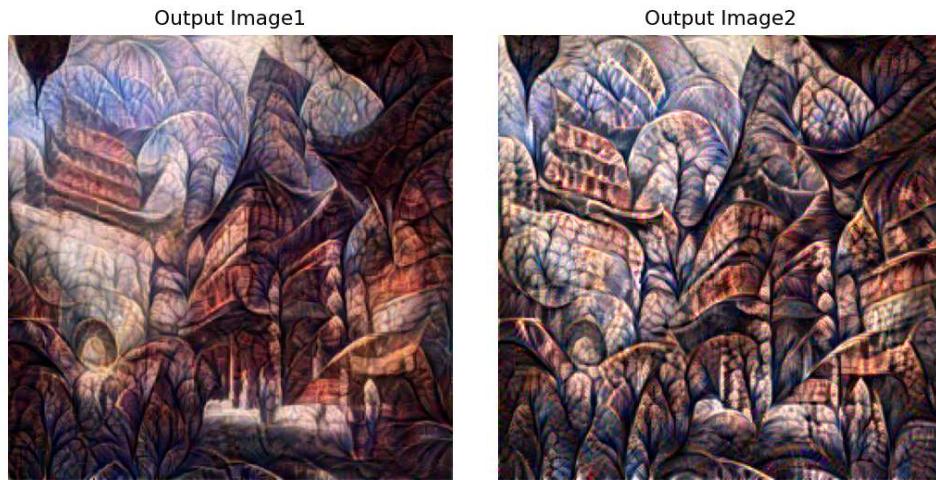
Hình 49.8: Bài 1 kết quả

**Bài 2:** Input nhận 1 ảnh content và 1 ảnh style. Chúng ta cần thực hiện trích xuất feature của ảnh style như thông thường, sau đó tạo ra 2 biến thể mới bằng cách xoay 90, và 180 độ. Tiếp theo tạo ra style mới theo công thức được lấy ý tưởng từ thuật toán differential evolution  $x_{new} = x + (x_{90} - x_{180})$ . Cuối cùng tạo ra 2 output được transfer theo style cũ và mới này lên ảnh content.



Hình 2.1 Ảnh đầu vào của bài tập 2, ảnh trái là ảnh style, và ảnh phải là ảnh content

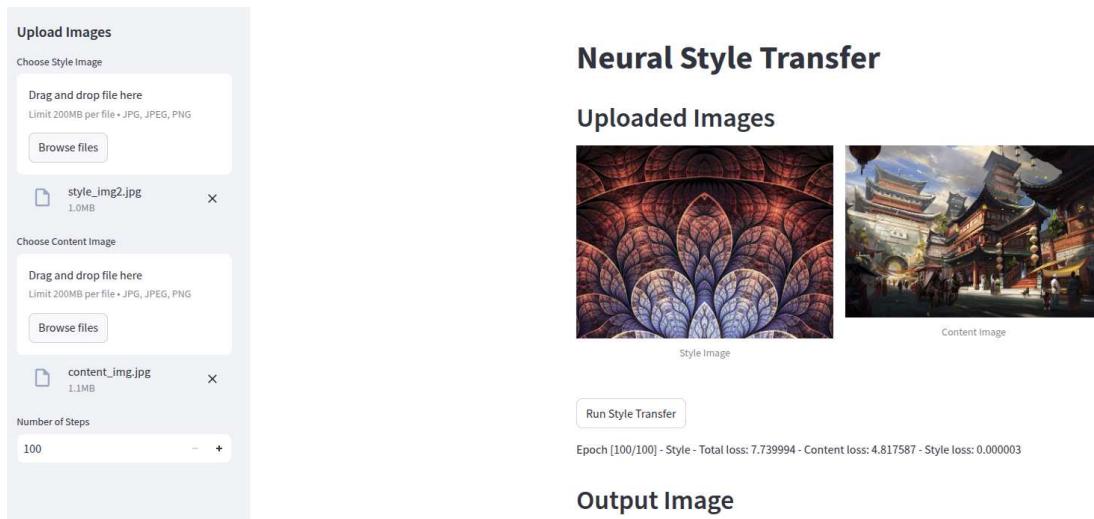
Ảnh output của bài tập 2, ảnh trái là ảnh khi transfer theo style và ảnh phải là ảnh khi transfer theo style đã thực hiện các biến đổi rotate 90, 180 và Differential Evolution



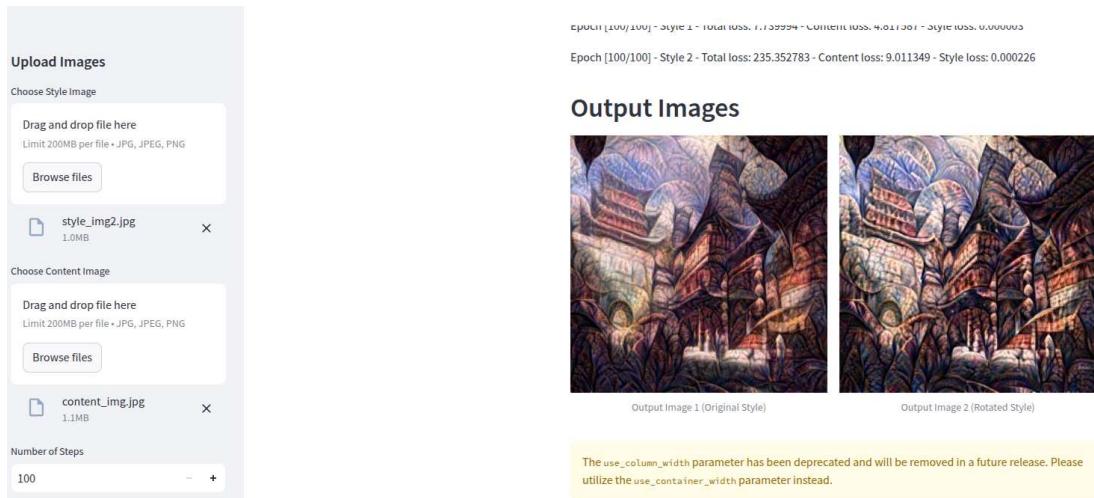
Hình 2.2 Bài 2 (a) input và (b) output

**Deploy:** Các bạn có thể tham khảo kết quả từ chương trình được deploy trên streamlit:

- [Github](#)
- Ảnh minh họa khi deploy bài tập 2 trên streamlit



Hình 2.3 Bài 2 upload style và content image



Hình 2.4 Bài 2 kết quả (100 steps). Khuyến khích chạy từ 500 steps và chạy trên local machine do cloud của streamlit giới hạn và chạy chậm

Các bạn tham khảo code hướng dẫn bên dưới để hoàn thành bài tập

1. **Load data:** Đầu tiên chúng ta cần load ảnh content và style bằng thư viện PIL, rồi resize về cùng kích thước 256x256 và convert thành tensor để sử dụng với Pytorch. Đối với bài 1 chúng ta cần load 3 ảnh 2 style và 1 content, còn đối với bài 2 chỉ cần 1 style và 1 content

```

1 from PIL import Image
2 import torchvision.transforms as transforms
3
4 imszie = 256
5
6 img_transforms = transforms.Compose([
7 transforms.Resize((imszie, imszie)),
8 transforms.ToTensor(),
9])
10
11 def image_loader(image_name):
12 image = Image.open(image_name)
13 image = img_transforms(image).unsqueeze(0)
14 return image.to(device, torch.float)
15
16 style_img1 = image_loader("style_img.jpg")
17 style_img2 = image_loader("style_img2.jpg")
18 content_img = image_loader("content_img.jpg")

```

2. **Loss Function:** Đoạn code bên dưới thiết lập các hàm loss được sử dụng trong style transfer để cân bằng giữa việc giữ content của hình ảnh trong khi áp dụng style của một hình ảnh khác. Content loss đảm bảo rằng các đặc điểm chính của hình ảnh content được giữ nguyên, trong khi style loss đảm bảo rằng phong cách của hình ảnh style được chuyển một cách hiệu quả sang hình ảnh được tạo ra.

```

1 # Content Loss
2 content_weight = 1
3 ContentLoss = nn.MSELoss()
4
5 # Style Loss
6 def gram_matrix(tensor):
7 a, b, c, d = tensor.size()
8 tensor = tensor.view(a * b, c * d)
9 G = torch.mm(tensor, tensor.t())
10 return G.div(a * b * c * d)
11 style_weight = 1e6
12 StyleLoss = nn.MSELoss()

```

### Content Loss:

- **content\_weight:** trọng số cho content loss trong hàm loss tổng. Trọng số này điều chỉnh tầm quan trọng của content được giữ lại so với các thành phần khác của loss tổng (chẳng hạn như mất đi kiểu dáng).
- **ContentLoss:** Sử dụng Mean Squared Error (MSE) Loss. Trong bối cảnh của style transfer, MSE được sử dụng để đo lường sự khác biệt giữa các đặc điểm nội dung của hình ảnh content và các đặc điểm của hình ảnh được tạo, khuyến khích hình ảnh được tạo giữ nguyên nội dung chính của hình ảnh content.

### Style Loss:

- **style\_weight:** trọng số cho style loss trong hàm loss tổng. Trọng số này điều chỉnh cường độ của phong cách được transfer vào ảnh output.
  - **StyleLoss:** Sử dụng Mean Squared Error (MSE) Loss
  - **gram\_matrix(tensor):** hàm tính toán ma trận Gram của một tensor. Ma trận Gram được sử dụng trong style loss để giữ lại các thành phần của style (kết cấu, hoa văn, màu sắc, v.v.) của hình ảnh.
3. **Model:** Chúng ta sẽ không build model từ đầu như các bài toán supervised learning khác mà ở đây ta sẽ tận dụng pre-trained VGG19 model để trích xuất feature của cả ảnh content và ảnh style. Ta sẽ cho ảnh đi qua pre-trained VGG19 model và trích xuất các feature map bên trong model. Thiết lập này thường được sử dụng trong style transfer, trong đó các feature layer của VGG19 được sử dụng để trích xuất các đặc trưng của ảnh style và ảnh content. Sau đó, đặc trưng này được sử dụng để hướng dẫn chuyển đổi hình ảnh content theo phong cách của hình ảnh style trong khi vẫn giữ nguyên nội dung của nó.

```
1 from torchvision.models import vgg19, VGG19_Weights
2
3 VGG19_pretrained = vgg19(weights=VGG19_Weights.DEFAULT).
4 features.eval()
5 VGG19_pretrained.to(device)
```

```

Sequential(
(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU(inplace=True)
(2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(3): ReLU(inplace=True)
(4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(6): ReLU(inplace=True)
(7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(8): ReLU(inplace=True)
(9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(11): ReLU(inplace=True)
(12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(13): ReLU(inplace=True)
(14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(15): ReLU(inplace=True)
(16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(17): ReLU(inplace=True)
(18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(20): ReLU(inplace=True)
(21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(22): ReLU(inplace=True)
(23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(24): ReLU(inplace=True)
(25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(26): ReLU(inplace=True)
(27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(29): ReLU(inplace=True)
(30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(31): ReLU(inplace=True)
(32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(33): ReLU(inplace=True)
(34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(35): ReLU(inplace=True)
(36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)

```

Hình 49.9: Các layer trong VGG19

Tiếp theo dựa và các thành layer trong VGG19 chúng ta sẽ lấy một số layer dùng cho trích xuất style và ảnh. Đoạn code bên dưới thể hiện cách trích xuất các feature từ các layer cụ thể của VGG19 cho bài toán style transfer. Xác định các layer nào được xem xét để biểu diễn nội dung (conv\_4) và phong cách (conv\_1, conv\_2, conv\_3, conv\_4, conv\_5)

```

1 content_layers = ["conv_4"]
2 style_layers = ["conv_1", "conv_2", "conv_3", "conv_4", "conv_5"]

```

```

3
4 def get_features(pretrained_model, image):
5 layers = {
6 "0": "conv_1",
7 "5": "conv_2",
8 "10": "conv_3",
9 "19": "conv_4",
10 "28": "conv_5"
11 }
12 features = []
13 x = image
14 x = normalization(x)
15 for name, pretrained_layer in pretrained_model.
16 _modules.items():
17 x = pretrained_layer(x)
18 if name in layers:
19 features[layers[name]] = x
20 return features

```

**get\_features(pretrained\_model, image):** Hàm này được thiết kế để lấy model được đào tạo trước (pretrain\_model) và tensor hình ảnh đầu vào (image) và trả về một từ điển chứa các feature từ các layer được chỉ định

Đối với bài 1 chúng ta cần thực hiện trích xuất feature của cả 2 ảnh style, sau đó combine theo tỉ lệ 1/2 và 1/4 để tạo ra style mới. Do đó ta cần hàm get\_dual\_style như code bên dưới:

```

1 def get_dual_style(style_features1, style_features2,
2 style_layers):
3 final_style_features = {}
4 for layer in style_layers:
5 sf1 = style_features1[layer]
6 sf2 = style_features2[layer]
7 ##### YOUR CODE HERE #####
8 # 1. Calculate the size of the first portion of
9 # sf1 to be used.
10 # This is a quarter of the number of channels
11 # (dimension 1).
12 # 2. Concatenate the first portion of sf1 with
13 # the second portion of sf2
14 # along the channel dimension.
15 # - The first portion of sf1 should contain
16 # the first "sf1_size" channels.

```

```

12 # - The second portion of sf2 should contain
13 # the channels starting
14 # from "sf1_size" to the end.
15 #
##########
15 return final_style_features

```

`get_dual_style(style_features1, style_features2, style_layers):`

- Hàm này nhận ba arguments: `style_features1` và `style_features2`, là các từ điển chứa các feature được trích xuất từ hai hình ảnh có style khác nhau bằng cách sử dụng hàm `get_features`. `style_layers`, lF danh sách tên layer mà các feature này được trích xuất.
- **step1** tính một 1/4 số lượng của tổng channel trong `sf1` (feature của style 1)
- **step2** lấy 1/4 số lượng channel của style 1 và 3/4 số lượng channel của style 2 sau đó concatenate lại tạo thành feature của style mới.

Đối với bài 2 chúng ta cần thực hiện trích xuất feature của ảnh style như thông thường, sau đó tạo ra 2 biến thế mới bằng cách xoay 90, và 180 độ. Tiếp theo tạo ra style mới theo công thức được lấy ý tưởng từ thuật toán differential evolution  $x_{new} = x + (x_{90} - x_{180})$ . Do đó ta cần hàm `rot_style_features` như code bên dưới:

```

1 def rot_style_features(style_features, style_layers):
2 final_rot_style_features = {}
3 for layer in style_layers:
4 sf = style_features[layer].clone()
5 ##### YOUR CODE HERE #####
6 # 2. Rotate the cloned tensor 90 degrees in the
6 # spatial dimensions (2, 3).
7 # 3. Rotate the 90-degree rotated tensor another
7 # 90 degrees (180 degrees total).
8 # 4. Calculate the final rotated feature by
8 # adding the original feature
9 # to the difference between the 90-degree and
9 # 180-degree rotations.
10 #
##########
11 final_rot_style_features[layer] = final_rot
12 return final_rot_style_features

```

### **rot\_style\_features(style\_features, style\_layers):**

- Hàm này nhận hai arguments: style\_features, là một từ điển chứa các feature của style được trích xuất cho từng layer được chỉ định của VGG19 và style\_layers, là danh sách các tên layer mà từ đó các feature của style được trích xuất
- **step1** rotate feature 90 độ
- **step2** rotate feature 180 độ
- **step3** Tạo ra feature mới theo công thức  $x_{new} = x + (x_{90} - x_{180})$

4. **Train:** Cách train của style transfer sẽ khác với các bài toán supervised learning thông thường ở cái mà ta xem là biến và cần update. Đối với bài toán thông thường ta sẽ xem model là các biến và update các biến này để minimize hàm loss và input là cố định. Tuy nhiên đối với style transfer thì model là cố định và ảnh input là biến và sẽ được update qua các vòng lặp để mỗi bước có thể transfer style của ảnh style và giữ các đặc điểm nội dung chính của ảnh content.

Đầu tiên ta cần khai báo ảnh input và optimizer, ở đây ta sẽ khai báo ảnh input chính là ảnh content, từ đó ta sẽ transfer style vào ảnh input này.

```

1 import torch.optim as optim
2 target_img = content_img.clone().requires_grad_(True).to(
 device)
3 optimizer = optim.Adam([target_img], lr=0.02)

```

- **line 2:** Copy ảnh content thành ảnh input đồng thời cho phép tính gradient trên ảnh input để update ảnh qua các train step
- **line 3:** Sử dụng Adam optimizer và nhận ảnh input để update theo thuật toán của Adam với learning rate = 0.02

Chúng ta cần thực hiện hàm transfer cho ảnh input theo 1 step. Chúng ta sẽ trích xuất content feature và style của ảnh input. Tiếp theo ta tính toán content loss và style loss (qua ma trận gram). Sau đó ta tính hàm loss tổng bằng tổng trọng số của content loss và style loss. Kế đến ta tính gradient và update gradient này lên ảnh input.

```

1 def style_tranfer_(model, optimizer, target_img,
2 content_features, style_features,
3 style_layers, content_weight,
4 style_weight):
5
6 optimizer.zero_grad()
7 with torch.no_grad():
8 target_img.clamp_(0, 1)
9 target_features = get_features(model, target_img)
10
11 content_loss = ContentLoss(content_features["conv_4"]
12 , target_features["conv_4"])
13
14 style_loss = 0
15 for layer in style_layers:
16 target_gram = gram_matrix(target_features[layer])
17 style_gram = gram_matrix(style_features[layer])
18 style_loss += StyleLoss(style_gram, target_gram)
19
20 total_loss = content_loss*content_weight + style_loss
21 *style_weight
22 total_loss.backward(retain_graph=True)
23 optimizer.step()
24
25 return total_loss, content_loss, style_loss

```

**style\_tranfer\_**: Hàm được thiết kế để thực hiện một step lặp lại quy trình style transfer, trong đó mục tiêu là cập nhật target\_img (ảnh input) sao cho kết hợp nội dung của hình ảnh content và phong cách của (các) hình ảnh style. Hàm này bao gồm việc tính toán content loss và style loss, tính toán tổng loss và cập nhật target\_img bằng cách sử dụng gradient

- **line 5:** Đặt lại tất cả gradient được tính toán trong lần lặp trước. Trong PyTorch, gradient tích lũy theo mặc định cho mỗi lệnh gọi `backward()`, vì vậy chúng cần được đặt về 0 trước khi tính toán gradient cho lần lặp tiếp theo.
- **line 8, 10:** trích xuất feature của ảnh input gồm style và content. Sau đó tính content loss của ảnh input và ảnh content
- **line 12-16:** Tính style loss của các feature map ta đã trích xuất trong VGG19. mỗi feature map sẽ được lấy ra của cả ảnh input và ảnh style. Sau đó tính toán ma trận gram của cả 2 và dùng MSE

để đo lường style loss. Tích luỹ các style loss qua các feature map ta đã trích xuất

- **line 18:** Tính tổng có trọng số của content loss và style loss
- **line 19:** Tính gradient cho ảnh input từ loss tổng
- **line 20:** update gradient vừa tính được vào ảnh input
- Hàm này được gọi lặp đi lặp lại, với mỗi lần gọi sẽ cập nhật target\_img (ảnh input) để phù hợp hơn với nội dung và style mong muốn. Qua nhiều lần lặp lại, target\_img dự kiến sẽ trở thành phiên bản có content\_img kết hợp các thành phần của (các) hình ảnh style.

Đối với bài 1 khi thực hiện 1 step transfer ta chỉ cần gọi hàm style\_transfer\_1 lần. Với đoạn code bên dưới ta thực hiện quy trình tối ưu hóa lặp đi lặp lại của việc transfer phong cách với 500 step, cập nhật hình ảnh input ở mỗi bước để dần dần phù hợp hơn với nội dung của hình ảnh content và phong cách của hình ảnh style.

```

1 STEPS = 500
2
3 for step in range(STEPS):
4 optimizer.zero_grad()
5 with torch.no_grad():
6 target_img.clamp_(0, 1)
7
8 total_loss, content_loss, style_loss = style_transfer_
9 (
10 VGG19_pretrained, optimizer,
11 target_img, content_features,
12 final_style_features,
13 style_layers, content_weight,
14 style_weight)
15 if step % 100 == 99:
16 print(f"Epoch [{step+1}/{STEPS}] Total loss: {total_loss.item():.6f} - \
17 Content loss: {content_loss.item():.6f} - \
18 Style loss: {style_loss.item():.6f}")
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
317
318
319
319
320
321
322
323
324
325
326
327
327
328
329
329
330
331
332
333
334
335
336
337
337
338
339
339
340
341
342
343
344
345
346
347
347
348
349
349
350
351
352
353
354
355
356
357
357
358
359
359
360
361
362
363
364
365
366
367
367
368
369
369
370
371
372
373
374
375
376
377
377
378
379
379
380
381
382
383
384
385
386
387
387
388
389
389
390
391
392
393
394
395
396
397
397
398
399
399
400
401
402
403
404
405
406
407
407
408
409
409
410
411
412
413
414
415
415
416
417
417
418
419
419
420
421
422
423
424
425
425
426
427
427
428
429
429
430
431
432
433
434
435
435
436
437
437
438
439
439
440
441
442
443
444
445
445
446
447
447
448
449
449
450
451
452
453
454
455
455
456
457
457
458
459
459
460
461
462
463
464
464
465
466
466
467
468
468
469
469
470
471
472
473
474
474
475
476
476
477
478
478
479
479
480
481
482
483
484
484
485
486
486
487
488
488
489
489
490
491
492
493
493
494
494
495
496
496
497
497
498
498
499
499
500
500
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
146
```

```
18 with torch.no_grad():
19 target_img.clamp_(0, 1)
```

- **line 18-19:** Ta cần chặn dưới và trên ảnh output trong range từ 0 đến 1. Vì ảnh thường được biểu diễn trong range 0-255 hoặc đã normalize trong range 0-1

Đối với bài 2 khi thực hiện 1 step transfer ta cần gọi hàm style\_transfer\_ 2 lần. Lần thứ nhất với ảnh style gốc và lần thứ 2 với style mới đã được biến đổi ở trên. Output sẽ là 2 ảnh đã được chuyển đổi theo 2 phong cách khác nhau .Với đoạn code bên dưới ta thực hiện quy trình tối ưu hóa lặp đi lặp lại của việc transfer phong cách với 500 step, cập nhật hình ảnh input ở mỗi bước để dần dần phù hợp hơn với nội dung của hình ảnh content và phong cách của hình ảnh style.

```
1 STEPS = 500
2
3 for step in range(STEPS):
4
5 total_loss1, content_loss1, style_loss1 =
6 style_transfer_
7
8 VGG19_pretrained, optimizer1,
9
10 target_img1, content_features,
11
12 style_features1, style_layers,
13
14 content_weight, style_weight)
15
16 total_loss2, content_loss2, style_loss2 =
17 style_transfer_
18
19 VGG19_pretrained, optimizer2,
20
21 target_img2, content_features,
22
23 final_rot_style_features,
24
25 style_layers, content_weight,
26
27 style_weight)
```

```

18 if step % 100 == 99:
19 print(f"Epoch [{step+1}/{STEPS}] Total loss1: {total_loss1.item():.6f} - \
20 Content loss1: {content_loss1.item():.6f}
21 - Style loss1: {style_loss1.item():.6f}")
22 print(f"Epoch [{step+1}/{STEPS}] Total loss2: {total_loss2.item():.6f} - \
23 Content loss2: {content_loss2.item():.6f}
24 - Style loss2: {style_loss2.item():.6f}")

25 with torch.no_grad():
26 target_img1.clamp_(0, 1)
27 target_img2.clamp_(0, 1)

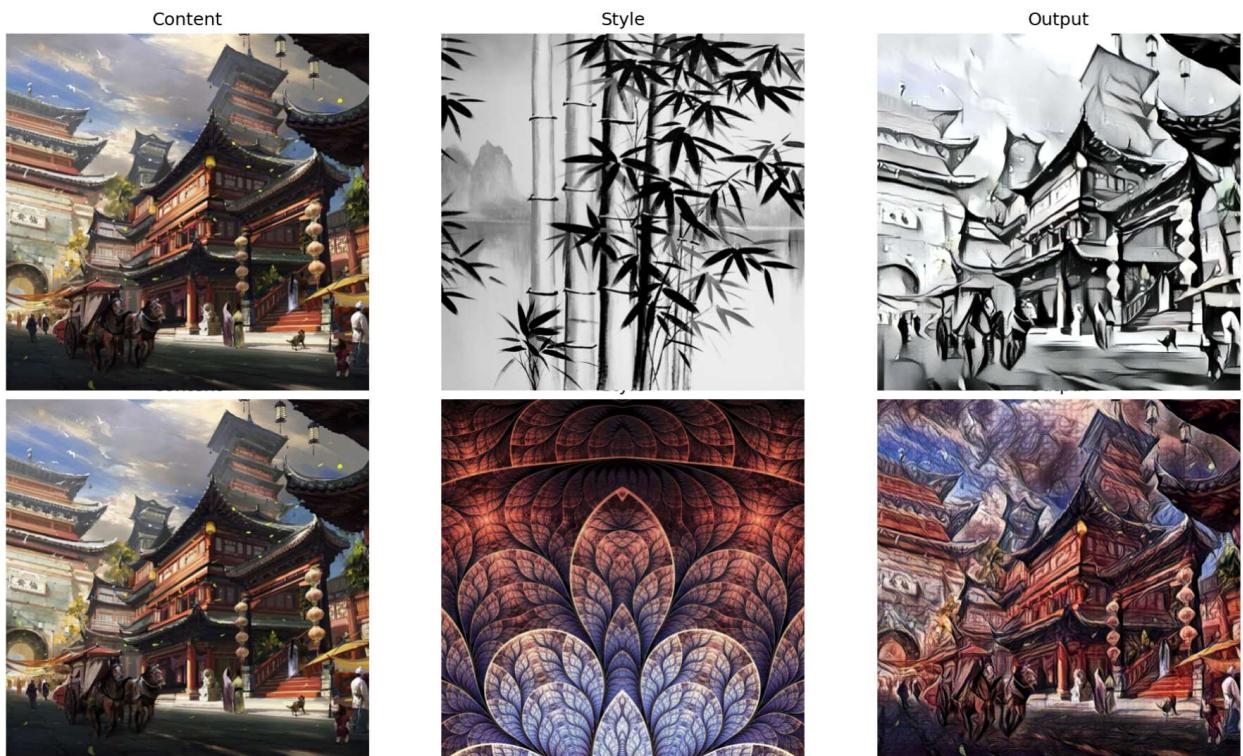
```

**Bài 3:** Mục tiêu của bài này chỉ sử dụng code thư viện đã đã có sẵn để chạy style transfer với transformer model và so sánh kết quả với CNN (VGG19). Các bạn có thể tham khảo cách sử dụng ở file hint chỉ với vài dòng code đơn giản.

```

1 import torch
2 import matplotlib.pyplot as plt
3 from StyTR.util.utils import process_images
4 from StyTR.util.utils import network as StyTR_model
5 import numpy as np
6
7 content_path = "/content/content_img.jpg"
8 style_path = "/content/style_img2.jpg"
9
10 content, style = process_images(content_path, style_path)
11
12 with torch.no_grad():
13 output= StyTR_model(content,style)
14 output = output[0].cpu()
15
16 fig, ax = plt.subplots(1, 3, figsize=(15, 5))
17 ax[0].imshow(content.cpu()[0].permute(1, 2, 0))
18 ax[0].set_title("Content")
19 ax[1].imshow(style.cpu()[0].permute(1, 2, 0))
20 ax[1].set_title("Style")
21 ax[2].imshow(output[0].permute(1, 2, 0))
22 ax[2].set_title("Output")
23 plt.show()

```



Hình 49.10: Style Transfer with Transformer

### 49.3 Trắc Nghiệm

1. Style transfer là sự kết hợp giữa?  
 (A). Trí tuệ nhân tạo và ngôn ngữ tự nhiên  
 (B). Trí tuệ nhân tạo và nghệ thuật  
 (C). Xử lý ảnh và lập trình máy tính  
 (D). Trí tuệ nhân tạo và âm thanh
2. Phương pháp nào thường được sử dụng để biểu diễn phong cách (style) của hình ảnh trong style transfer?  
 (A). Gram matrix  
 (B). Fourier transform  
 (C). Sobel operator  
 (D). Histogram
3. Style transfer thường dựa trên việc sử dụng mạng nơ-ron nào?

- (A). RNN (Recurrent Neural Network) (B). CNN (Convolutional Neural Network)  
(C). LSTM (Long Short-Term Memory) (D). MLP (Multilayer Perceptron)
4. Các loại model nào thường được sử dụng trong Style transfer?  
(A) VGG (Very Deep Convolutional Networks)  
(B) ResNet (Residual Neural Network)  
(C) Tất cả đều sai  
(D) Các pretrain CNN model
5. Đối với Style transfer, content loss thường được tính toán như thế nào?  
(A). Histogram của hình ảnh  
(B). Mean Squared Error (MSE) giữa biểu diễn nội dung của hình ảnh gốc và hình ảnh được tạo  
(C). Gradient của hình ảnh  
(D). Mean Squared Error (MSE) giữa biểu diễn Gram matrix của hình ảnh phong cách và hình ảnh được tạo
6. Đối với Style transfer, style loss thường được tính toán như thế nào??  
(A). Histogram của hình ảnh  
(B). Mean Squared Error (MSE) giữa biểu diễn nội dung của hình ảnh gốc và hình ảnh được tạo  
(C). Gradient của hình ảnh  
(D). Mean Squared Error (MSE) giữa biểu diễn Gram matrix của hình ảnh phong cách và hình ảnh được tạo
7. Trong style transfer, loss cuối cùng dùng để cập nhật ảnh được tạo ra là ?  
(A). Style loss (B). Content Loss  
(C). Tổng có trọng số của style và content (D). Tất cả đều đúng
8. Style transfer thường được áp dụng trong lĩnh vực nào sau đây?  
(A). Phân loại hình ảnh y tế (B). Nghệ thuật số và thiết kế đồ họa  
(C). Phát hiện gian lận tài khoản ngân hàng (D). Tổ chức sự kiện

9. Cách nào sau đây có thể thực hiện được multi-modal style transfer?
- (A). Bằng cách biến đổi feature map của ảnh nội dung (content), ta có thể tạo ra nhiều phong cách khác nhau chỉ với một ảnh phong cách đầu vào.
- (B). Bằng cách biến đổi feature map của ảnh output được tạo ra, ta có thể tạo ra nhiều phong cách khác nhau chỉ với một ảnh phong cách đầu vào.
- (C). Bằng cách biến đổi feature map của ảnh phong cách (style), ta có thể tạo ra nhiều phong cách khác nhau chỉ với một ảnh phong cách đầu vào.
- (D). Tất cả đều đúng.
10. Mục tiêu của style transfer là gì?
- (A). Chuyển đổi một hình ảnh thành một hình ảnh khác với cùng một phong cách nhưng nội dung khác nhau.
- (B). Giảm kích thước của hình ảnh để tiết kiệm không gian lưu trữ.
- (C). Tạo ra một hình ảnh có chất lượng cao hơn so với hình ảnh gốc.
- (D). Chuyển đổi một hình ảnh thành một hình ảnh khác với cùng một nội dung nhưng phong cách khác nhau.
11. Đoạn code nào sau đây là phù hợp với yêu cầu bài tập 1:

```
1 def get_dual_style(style_features1, style_features2,
2 style_layers):
3 final_style_features = []
4 for layer in style_layers:
5 sf1 = style_features1[layer]
6 sf2 = style_features2[layer]
7 ##### YOUR CODE HERE #####
8 # 1. Calculate the size of the first portion of
9 # sf1 to be used.
10 # This is a quarter of the number of channels
11 # (dimension 1).
12 # 2. Concatenate the first portion of sf1 with
13 # the second portion of sf2
14 # along the channel dimension.
15 # - The first portion of sf1 should contain
16 # the first "sf1_size" channels.
```

```

12 # - The second portion of sf2 should contain
13 # the channels starting
14 # from "sf1_size" to the end.
15 #
16 #####
17
18 return final_style_features

```

(A).

```

1 sf1_size = int(sf1.size()[1] / 4)
2 final_style_features[layer] = torch.concatenate([
3 sf1[:, :sf1_size
4 , :, :],
5 sf2[:, sf1_size
5 , :, :]
5], dim=1)

```

(B).

```

1 sf1_size = int(sf1.size()[1] * 4)
2 final_style_features[layer] = torch.concatenate([
3 sf1[:, :sf1_size
4 , :, :],
5 sf2[:, sf1_size
5 , :, :]
5], dim=1)

```

(C).

```

1 sf1_size = int(sf1.size()[1] / 4)
2 final_style_features[layer] = torch.concatenate([
3 sf1[:, :sf1_size
4 , :, :],
5 sf1[:, sf1_size
5 , :, :]
5], dim=1)

```

(D).

```

1 sf1_size = int(sf1.size()[1] * 4)
2 final_style_features[layer] = torch.concatenate([
3 sf2[:, :sf1_size
4 , :, :],
5 sf2[:, sf1_size
5 , :, :]
5], dim=1)

```

12. Đoạn code nào sau đây là phù hợp với yêu cầu bài tập 2:

```

1 def rot_style_features(style_features, style_layers):
2 final_rot_style_features = {}
3 for layer in style_layers:
4 sf = style_features[layer].clone()
5 ##### YOUR CODE HERE
6 # 2. Rotate the cloned tensor 90 degrees in the
7 # spatial dimensions (2, 3).
8 # 3. Rotate the 90-degree rotated tensor another
9 # 90 degrees (180 degrees total).
10 # 4. Calculate the final rotated feature by
11 # adding the original feature
12 # to the difference between the 90-degree and
13 # 180-degree rotations.
14 #
15 #####
16 final_rot_style_features[layer] = final_rot
17 return final_rot_style_features

```

(A).

```

1 rot90 = torch.rot90(sf.clone(), 1, (2, 3))
2 rot180 = torch.rot90(rot90.clone(), 1, (2, 3))
3 final_rot = sf + (rot90 - rot180)

```

(B).

```

1 rot90 = torch.rot90(sf.clone(), 1, (2, 3))
2 rot180 = torch.rot180(rot180.clone(), 1, (2, 3))
3 final_rot = sf + (rot90 - rot180)

```

(C).

```

1 rot90 = torch.rot90(sf.clone(), 1, (2, 3))
2 rot180 = torch.rot90(rot180.clone(), 1, (2, 3))
3 final_rot = sf + (rot90 - rot180)

```

(D). Tất cả đều sai

## 49.4 Phụ Lục

- (a) **Datasets:** Các file data có thể tải tại [Link](#)
- (b) **Hint:** Các file code gợi ý có thể tải tại [Link](#)
- (c) **Solution:** Các file code cài đặt hoàn chỉnh và phâ trả lời nội dung trắc nghiệm có thể tả tại [Link](#) (**Lưu ý** Sáng thứ 3 khi hết deadline phần bài tập ad mới copy các nội dung bài giải nêu trên vào đường dẫn)
- (d) **Rubric:**

Style Transfer - Rubric		
Câu	Kiến Thức	Đánh Giá
1	<ul style="list-style-type: none"> <li>- Khái niệm về Neural Style Transfer</li> <li>- Khái niệm về Content Loss trong Style Transfer</li> <li>- Khái niệm về Style Loss trong Style Transfer khi kết hợp 2 style feautre để tạo ra syle mới</li> <li>- Hiểu biết về cách trích xuất đặc trưng từ hai ảnh style</li> <li>- Kỹ năng kết hợp các đặc trưng để tạo ra style mới.</li> </ul>	<ul style="list-style-type: none"> <li>- Biết cách code kết hợp đặc trưng một cách sáng tạo để tạo ra style mới</li> <li>- Biết cách code trích xuất feature của các layer trong pre-trained model</li> <li>- Biết cách code full luồng cho bài toán style transfer</li> </ul>
2	<ul style="list-style-type: none"> <li>- Khái niệm về Neural Style Transfer</li> <li>- Khái niệm về Content Loss trong Style Transfer</li> <li>- Khái niệm về Style Loss trong Style Transfer khi áp dụng các thuật toán làm biến đổi feature map để tạo ra style mới</li> <li>- Hiểu biết về cách biến đổi feature map của style để tạo ra style mới</li> </ul>	<ul style="list-style-type: none"> <li>- Biết cách code kết biến đổi đặc trưng (áp dụng các thuật toán khác nhau) một cách sáng tạo để tạo ra style mới</li> <li>- Biết cách code trích xuất feature của các layer trong pre-trained model</li> <li>- Biết cách code full luồng cho bài toán style transfer</li> </ul>

# Chương 50

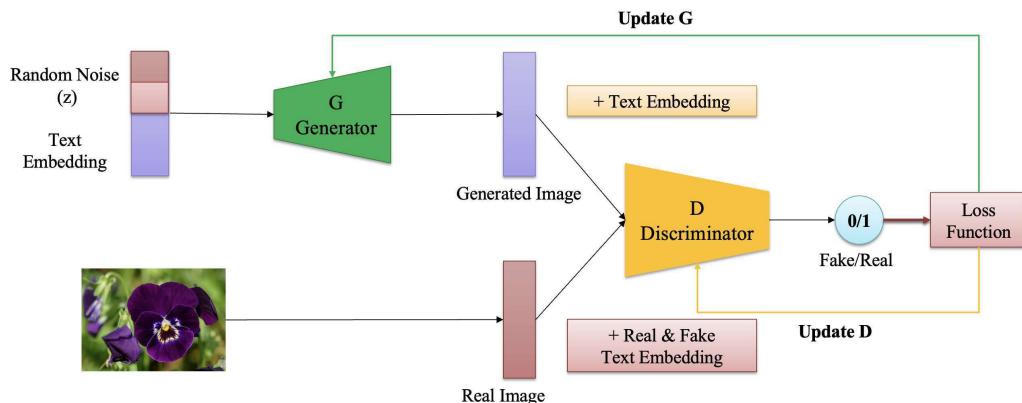
## GAN và DCGAN

### 50.1 Giới thiệu

The flower has petals that  
are bright pinkish purple  
with white stigma



Hình 50.1: Ví dụ sinh văn bản thành hình ảnh sử dụng mô hình DCGAN.



Hình 50.2: Mô hình DCGAN sinh văn bản thành hình ảnh.

**Sinh hình ảnh từ văn bản hay tổng hợp văn bản thành hình ảnh (Text to Image)** là bài toán ngày càng được ứng dụng mạnh mẽ trong lĩnh vực trí tuệ nhân tạo. Các mô hình được huấn luyện với đầu vào là đoạn văn bản và đầu ra là hình ảnh mô tả hoặc chứa các đối tượng được mô tả

trong đoạn văn bản. Ví dụ về tổng hợp hình ảnh từ văn bản được mô tả trong Hình 1.

Hiện nay, có nhiều mô hình có thể xây dựng và giải quyết bài toán này có thể kể đến như GANs, Diffusion Models,... Ở trong phần này chúng ta sẽ sử dụng mô hình DCGAN (Deep Convolution Generative Adversarial Networks) để huấn luyện mô hình giải quyết bài toán này.

Mô hình DCGAN gồm 2 mạng là Generator và Discriminator. Trong đó:

1. Generator: Bao gồm các lớp mạng CNN nhận đầu vào là Noise ( $z$ ) và Text Embedding ( $e$  - Biểu diễn của đoạn văn bản đầu vào). Đầu ra của khối Generator là ma trận ảnh được sinh ra  $G(z, e)$ .
2. Discriminator: Bao gồm các lớp mạng CNN nhận đầu vào là biểu diễn của ảnh (ảnh thật hoặc ảnh fake được sinh ra từ khối Generator) sau đó kết hợp với Text Embedding để dự đoán ảnh thật hay ảnh fake.

## 50.2 Text To Image Synthesis using DCGAN-BERTs

Trong phần này chúng ta sẽ xây dựng và huấn luyện mô hình DCGAN trên bộ dữ liệu flowers. Bộ dữ liệu flowers có thể được tải về [tại đây](#).

Các bước để huấn luyện mô hình bao gồm:

- (a) Dataset: Tải bộ dữ liệu Flower.
- (b) Caption Tokenizer: Sử dụng mô hình BERT sử dụng sentence-transformers để biểu diễn văn bản thành vector. Với mỗi văn bản sẽ được biểu diễn thành một vector tương ứng.
- (c) Preprocessing: Xây dựng dữ liệu I/O cho mô hình.
- (d) Model: Xây dựng mô hình Generator và Discriminator.
- (e) Training: Huấn luyện mô hình.
- (f) Deployment: Triển khai ứng dụng sử dụng Streamlit.

### 1. Dataset

Thực thi đoạn code sau để tải về bộ dữ liệu, sau đó giải nén thu được 2 thư mục "flowers" chứa các file .txt chứa các đoạn văn bản mô tả hình ảnh, trong đó mỗi hình ảnh sẽ lấy một mô tả đầu tiên để huấn luyện mô hình và "images" chứa các hình ảnh.

```

1 # Download dataset
2 !gdown 1JJjMiNjeTz7xYs6UeVqd02M3DW4fnEfU
3 !unzip cvpr2016_flowers.zip
4
5 # Load captions
6 import os
7
8 def load_captions(captions_folder, image_folder):
9 captions = {}
10 image_files = os.listdir(image_folder)
11
12 for image_file in image_files:
13
14 image_name = image_file.split('.')[0]

```

```

15 caption_file = os.path.join(captions_folder,
16 image_name + ".txt")
17 with open(caption_file, "r") as f:
18 caption = f.readlines()[0].strip()
19 if image_name not in captions:
20 captions[image_name] = caption
21
22
23 captions_folder = "./cvpr2016_flowers/captions"
24 image_folder = "./cvpr2016_flowers/images"
25
26 captions = load_captions(captions_folder, image_folder)
27 captions

```

## 2. Caption Encoder

Trong phần này, chúng ta sử dụng mô hình BERTs để biểu diễn mỗi câu mô tả thành một vector có kích thước 768 chiều.

```

1 import torch
2 import numpy as np
3 from sentence_transformers import SentenceTransformer
4
5 device = torch.device("cuda" if torch.cuda.is_available()
6 else "cpu")
6 bert_model = SentenceTransformer("all-mpnet-base-v2").to(
 device)
7
8 def encode_captions(captions):
9 encoded_captions = {}
10 for image_name in captions.keys():
11 caption = captions[image_name]
12 encoded_captions[image_name] = {
13 "embed": torch.tensor(bert_model.encode(caption))
14 ,
15 "text": caption
16 }
17 return encoded_captions
17 encoded_captions = encode_captions(captions)

```

## 3. Preprocessing

Thực thi đoạn code sau đây để chuẩn bị I/O cho mô hình:

```

1 from PIL import Image
2 from torch.utils.data import Dataset
3

```

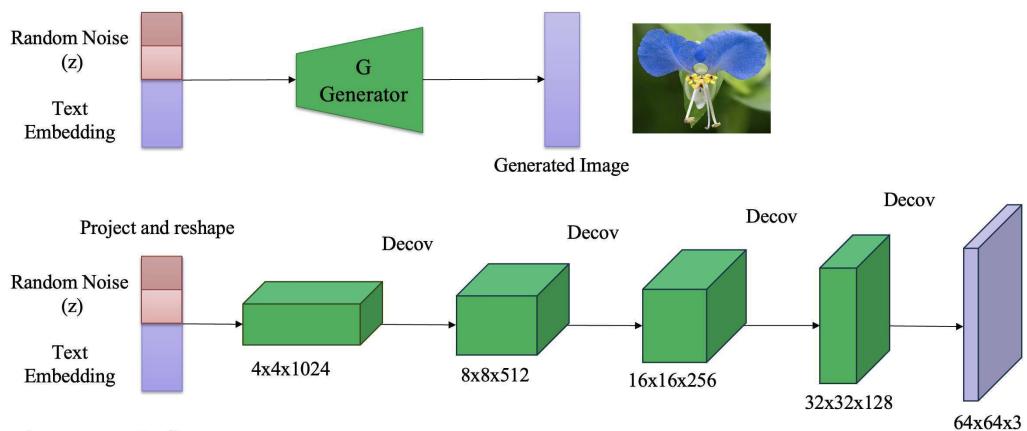
```
4 class FlowerDataset(Dataset):
5 def __init__(self, img_dir, captions, transform=None):
6 self.img_dir = img_dir
7 self.transform = transform
8 # Load captions
9 self.captions = captions
10 self.img_names = list(self.captions.keys())
11
12 def __len__(self):
13 return len(self.img_names)
14
15 def __getitem__(self, idx):
16 img_name = self.img_names[idx]
17 img_path = os.path.join(self.img_dir, img_name+".jpg")
18
19 image = Image.open(img_path).convert("RGB")
20 if self.transform:
21 image = self.transform(image)
22 encoded_caption = self.captions[img_name]["embed"]
23 caption = self.captions[img_name]["text"]
24 return {
25 "image": image,
26 "embed_caption": encoded_caption,
27 "text": caption
28 }
29
30
31 import torchvision.transforms as transforms
32
33 from torch.utils.data import DataLoader
34
35 IMG_SIZE = 128
36
37 transform = transforms.Compose([
38 transforms.Resize((IMG_SIZE, IMG_SIZE)),
39 transforms.ToTensor(),
40 transforms.Normalize([0.5], [0.5])
41])
42
43 ds = FlowerDataset(
44 img_dir="/content/cvpr2016_flowers/images",
45 captions=encoded_captions,
46 transform=transform
47)
48
49 BATCH_SIZE = 1024
50 dataloader = DataLoader(ds, batch_size=BATCH_SIZE, shuffle=
```

True)

### 3. Model

Trong phần này chúng ta xây dựng mô hình DCGAN để sinh văn bản thành hình ảnh bao gồm 2 mạng Generator và Discriminator:

#### 3.1. Generator



Hình 50.3: Khối Generator trong DCGAN.

Khối Generator sinh ra ảnh từ văn bản đầu vào:

- Input: Nhận đầu vào là vector: Random Noise (z) có kích thước R, được nối với Vector Embedding (Vector e - biểu diễn cho cả đoạn văn bản đầu vào) có kích thước là D. Vì vậy, vector đầu vào mạng Generator là R + D.
- Output: Sau khi học mối quan hệ để sinh ảnh, giá trị đầu ra của Generator sẽ là biểu diễn các điểm ảnh dự đoán trong không gian 3 chiều có kích thước là Channel x Width x Height (Ví dụ, CxWxH - 3x64x64)

```

1 import torch.nn as nn
2
3 class Generator(nn.Module):
4
5 def __init__(self, noise_size, feature_size, num_channels,
6 embedding_size, reduced_dim_size):

```

```
6 super(Generator, self).__init__()
7 self.reduced_dim_size = reduced_dim_size
8 #768-->256
9 self.textEncoder = nn.Sequential(
10 nn.Linear(in_features = embedding_size,
11 out_features = reduced_dim_size),
12 nn.BatchNorm1d(num_features = reduced_dim_size),
13 nn.LeakyReLU(negative_slope = 0.2, inplace = True
14)
15)
16
17 self.upsamplingBlock = nn.Sequential(
18 #256+100 --> 1024
19 nn.ConvTranspose2d(noise_size + reduced_dim_size,
20 feature_size * 8, 4, 1, 0, bias = False),
21 nn.BatchNorm2d(feature_size * 8),
22 nn.LeakyReLU(negative_slope = 0.2, inplace = True
23),
24 # 1024 --> 512
25 nn.ConvTranspose2d(feature_size * 8, feature_size
26 * 4, 4, 2, 1, bias = False),
27 nn.BatchNorm2d(feature_size * 4),
28 nn.ReLU(True),
29 # 512 --> 256
30 nn.ConvTranspose2d(feature_size * 4, feature_size
31 * 2, 4, 2, 1, bias=False),
32 nn.BatchNorm2d(feature_size * 2),
33 nn.ReLU(True),
34 # 256 --> 128
35 nn.ConvTranspose2d(feature_size * 2, feature_size
36 , 4, 2, 1, bias=False),
37 nn.BatchNorm2d(feature_size),
38 nn.ReLU(True),
39 # 128 --> 128
40 nn.ConvTranspose2d(feature_size, feature_size, 4,
41 2, 1, bias=False),
42 nn.BatchNorm2d(feature_size),
43 nn.ReLU(True),
44 # 128 --> 3
45 nn.ConvTranspose2d(feature_size, num_channels, 4,
46 2, 1, bias=False),
47 nn.Tanh()
48)
49
50 def forward(self, noise, text_embeddings):
```

```

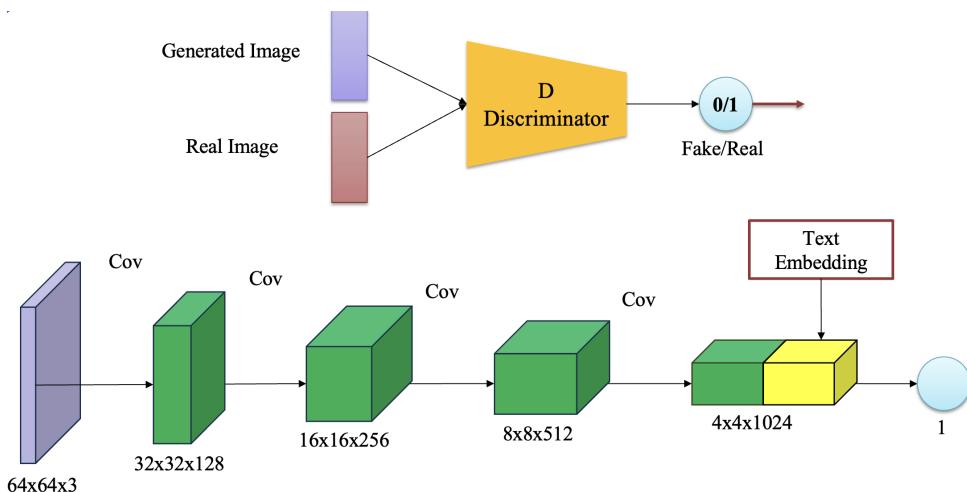
42 encoded_text = self.textEncoder(text_embeddings)
43 concat_input = torch.cat([noise, encoded_text], dim =
44 1).unsqueeze(2).unsqueeze(2)
45 output = self.upsamplingBlock(concat_input)
 return output

```

### 3.2. Discriminator

Khối Discriminator dự đoán hình ảnh Fake/Real, bao gồm các lớp CNN với:

- (a) Input: Nhận đầu vào là ma trận điểm ảnh và Text Embedding. Ma trận điểm ảnh ( $C \times H \times W$ ) sau khi qua các lớp CNN để học các đặc trưng của ảnh đầu vào sẽ được nối với vector embedding.
- (b) Output: Vector sau khi được nối sẽ được sử dụng để dự đoán 0(Fake) hoặc 1(Real).



Hình 50.4: Khối Discriminator trong DCGAN.

```
1 class Discriminator(nn.Module):
2
3 def __init__(self, num_channels, feature_size,
4 embedding_size, reduced_dim_size):
5 super(Discriminator, self).__init__()
6 self.reduced_dim_size = reduced_dim_size
7 self.imageEncoder = nn.Sequential(
8 # 3 -> 128
9 nn.Conv2d(num_channels, feature_size, 4, 2, 1,
10 bias=False),
11 nn.LeakyReLU(0.2, inplace=True),
12 # 128 -> 128
13 nn.Conv2d(feature_size, feature_size, 4, 2, 1,
14 bias=False),
15 nn.LeakyReLU(0.2, inplace=True),
16 # 128 -> 256
17 nn.Conv2d(feature_size, feature_size * 2, 4, 2,
18 1, bias=False),
19 nn.BatchNorm2d(feature_size * 2),
20 nn.LeakyReLU(0.2, inplace=True),
21 # 256 -> 512
22 nn.Conv2d(feature_size * 2, feature_size * 4, 4,
23 2, 1, bias=False),
24 nn.BatchNorm2d(feature_size * 4),
25 nn.LeakyReLU(0.2, inplace=True),
26 # 512 -> 1024
27 nn.Conv2d(feature_size * 4, feature_size * 8, 4,
28 2, 1, bias=False),
29 nn.BatchNorm2d(feature_size * 8),
30 nn.LeakyReLU(0.2, inplace=True),
31)
32 self.textEncoder = nn.Sequential(
33 nn.Linear(in_features=embedding_size,
34 out_features=reduced_dim_size),
35 nn.BatchNorm1d(num_features=reduced_dim_size),
36 nn.LeakyReLU(negative_slope=0.2, inplace=True)
37)
38 self.finalBlock = nn.Sequential(
39 nn.Conv2d(feature_size * 8 + reduced_dim_size, 1,
40 4, 1, 0, bias=False),
41 nn.Sigmoid()
42)
43 def forward(self, input_img, text_embeddings):
44 image_encoded = self.imageEncoder(input_img)
45 text_encoded = self.textEncoder(text_embeddings)
```

```

38 replicated_text = text_encoded.repeat(4, 4, 1, 1).
39 permute(2, 3, 0, 1)
40 concat_layer = torch.cat([image_encoded,
41 replicated_text], 1)
42 x = self.finalBlock(concat_layer)
43 return x.view(-1, 1), image_encoded

```

#### 4. Training

Trước khi training mô hình, chúng ta định nghĩa mô số hàm để hiển thị hình ảnh được sinh ra sau mỗi 10 epochs của training.

```

1 import matplotlib.pyplot as plt
2 import torchvision
3
4 def show_grid(img):
5 npimg = img.numpy()
6 plt.imshow(np.transpose(npimg, (1, 2, 0)))
7 plt.show()
8
9 show_grid(torchvision.utils.make_grid(ds[0]["image"],
10 normalize=True))
11
11 def plot_output(generator):
12 plt.clf()
13 with torch.no_grad():
14
15 generator.eval()
16 test_images = generator(fixed_noise.to(device),
17 plt_o_text_embeddings.to(device))
18 generator.train()
19
20 grid = torchvision.utils.make_grid(test_images.cpu(),
21 normalize=True)
22 show_grid(grid)

```

Định nghĩa model, optimizer để huấn luyện model:

```

1 import torch.optim as optim
2
3 generator = Generator(100, 128, 3, 768, 256).to(device)
4 discriminator = Discriminator(3, 128, 768, 256).to(device)
5
6 optimizer_G = optim.Adam(generator.parameters(), lr=0.0002,
 betas=(0.5, 0.999))

```

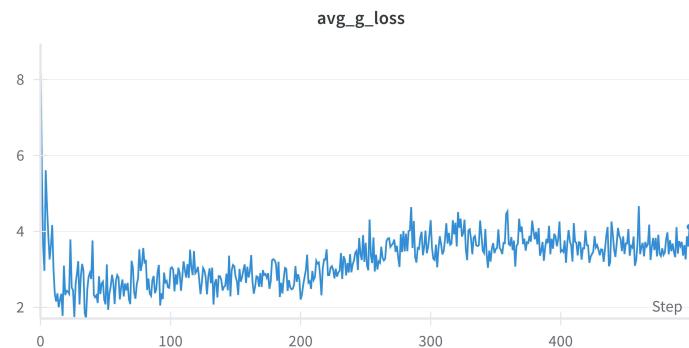
```
7 optimizer_D = optim.Adam(discriminator.parameters(), lr
=0.0002, betas=(0.5, 0.999))
8
9 bce_loss = nn.BCELoss()
10 l2_loss = nn.MSELoss()
11 l1_loss = nn.L1Loss()

1 import time
2
3 epochs = 500
4
5 for epoch in range(epochs):
6 d_losses, g_losses = [], []
7 epoch_time = time.time()
8
9 for batch in dataloader:
10 images = batch["image"].to(device)
11 embed_captions = batch["embed_caption"].to(device)
12 wrong_images = batch["wrong_image"].to(device)
13
14 # labels
15 real_labels = torch.ones(images.size(0), 1, device=device)
16 fake_labels = torch.zeros(images.size(0), 1, device=device)
17
18 # Training the discriminator
19 optimizer_D.zero_grad()
20
21 # Gen fake image
22 noise = torch.randn(size=(images.size(0), 100),
device=device)
23 fake_images = generator(noise, embed_captions)
24
25 # Compute real loss
26 outputs, _ = discriminator(images, embed_captions)
27 real_loss = bce_loss(outputs, real_labels)
28
29 # Compute contrastive loss for wrong image
30 outputs, _ = discriminator(wrong_images,
embed_captions)
31 wrong_loss = bce_loss(outputs, fake_labels)
32
33 # Compute fake loss
34 outputs, _ = discriminator(fake_images.detach(),
embed_captions)
```

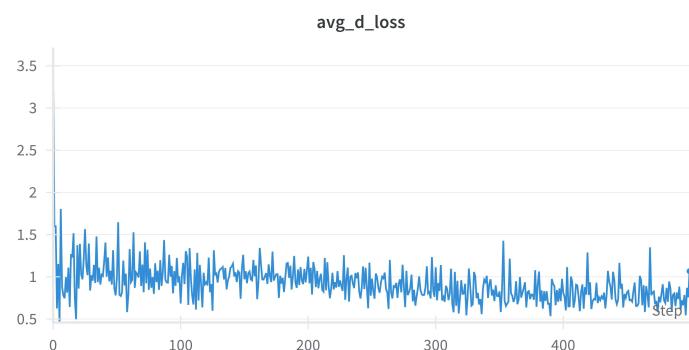
```
35 fake_loss = bce_loss(outputs, fake_labels)
36
37 d_loss = real_loss + fake_loss + wrong_loss
38
39 # Update weight
40 d_loss.backward()
41
42 optimizer_D.step()
43 d_losses.append(d_loss.item())
44
45 # Training generator
46 optimizer_G.zero_grad()
47 # create noise
48 noise = torch.randn(size=(images.size(0), 100),
49 device=device)
50 fake_images = generator(noise, embed_captions)
51
52 outputs, fake_features = discriminator(fake_images,
53 embed_captions)
54 _, real_features = discriminator(images,
55 embed_captions)
56 activation_fake = torch.mean(fake_features, 0)
57 activation_real = torch.mean(real_features, 0)
58
59 # Compute loss
60 real_loss = bce_loss(outputs, real_labels)
61 g_loss = real_loss + 100 * 12_loss(activation_fake,
62 activation_real.detach()) + 50 * 11_loss(fake_images,
63 images)
64 g_loss.backward()
65 optimizer_G.step()
66 g_losses.append(real_loss.item())
67
68 avg_d_loss = sum(d_losses)/len(d_losses)
69 avg_g_loss = sum(g_losses)/len(g_losses)
70 if (epoch+1) % 10 == 0:
71 plot_output(generator)
72
73 print("Epoch [{}/{}] loss_D: {:.4f} loss_G: {:.4f} time:
74 {:.2f}".format(epoch+1, epochs, avg_d_loss, avg_g_loss,
75 time.time() - epoch_time))
76
77 # save model
78 model_save_path = "./save_model"
79 torch.save(generator.state_dict(), os.path.join(
80 model_save_path, "generator.pth"))
```

```
29 torch.save(discriminator.state_dict(), os.path.join(
 model_save_path, "discriminator.pth"))
```

Kết quả training của model:



Hình 50.5: Generator loss.



Hình 50.6: Discriminator loss.

## 5. Deployment

Thử nghiệm mô hình sau khi huấn luyện:

```
1 generator.eval()
2 caption = "this pale pink flower has a large yellow and
3 green pistil."
4 embed_caption = torch.tensor(bert_model.encode(caption))
5 noise = torch.randn(size=(1, 100))
6 text_embedding = embed_caption.unsqueeze(0)
7 with torch.no_grad():
8 test_images = generator(noise.to(device),
9 text_embedding.to(device))
10 grid = torchvision.utils.make_grid(test_images.cpu(),
11 normalize=True)
12 show_grid(grid)
```



Hình 50.7: Kết quả thử nghiệm.

Triển khai mô hình trên streamlit. Tham khảo về [code](#) và [demo](#).

### Text To Image Using DCGAN-BERTs

**Model: DCGAN. Dataset: Flower. Text Encoder: BERTs**

Sentence:

this pale pink flower has a large yellow pistil.



Generated Image

Hình 50.8: Triển khai ứng dụng trên Streamlit.

### 50.3 Câu hỏi trắc nghiệm

**Câu hỏi 83** Mục đích của bài toán Text to Image là gì?

- a) Tạo hình ảnh tương ứng với thông tin mô tả từ văn bản
- b) Phân loại hình ảnh
- c) Phân loại văn bản
- d) Phát hiện cạnh của các đối tượng trong hình ảnh

**Câu hỏi 84** Mô hình nào được xây dựng để giải quyết bài toán Text-to-Image?

- a) GAN-CLS
- b) BERT
- c) ResNet
- d) VGG

**Câu hỏi 85** Khối Generator được sử dụng để làm gì?

- a) Sinh hình ảnh mới từ Noise và Text Embedding
- b) Dự đoán hình ảnh Real/Fake
- c) Sinh hình ảnh mới từ hình ảnh cũ
- d) Sinh văn bản mới từ Noise và Text Embedding

**Câu hỏi 86** Khối Discriminator được sử dụng để làm gì?

- a) Sử dụng kết hợp hình ảnh và Text Embedding dự đoán hình ảnh Fake/Real
- b) Sử dụng Text Embedding dự đoán hình ảnh Fake/Real
- c) Cả 2 đáp án trên đều đúng
- d) Cả 2 đáp án trên đều sai

**Câu hỏi 87** Kiến trúc của Generator và Discriminator không bao gồm layer nào sau đây?

- a) CNN
- b) Batch Normalization

- c) LeakyReLU Activation
- d) RNN

**Câu hỏi 88** Số chiều biểu diễn văn bản được sử dụng trong phần thực nghiệm 2 là bao nhiêu?

- a) 512
- b) 1024
- c) 768
- d) 2048

**Câu hỏi 89** Dựa vào code thực nghiệm trong phần 2, lớp Convolution cuối cùng trong khối Discriminator có tham số ‘in channels’ là bao nhiêu?

- a) 512
- b) 64
- c) 576
- d) 1024

**Câu hỏi 90** Hàm loss nào sau đây không được sử dụng trong phần thực nghiệm?

- a) BCELoss
- b) MSELoss
- c) L1Loss
- d) CTCLoss

**Câu hỏi 91** Mô hình pre-trained language model nào được sử dụng trong thực nghiệm?

- a) BERT
- b) DistilBERT
- c) RoBERTa
- d) T5

**Câu hỏi 92** Dựa vào phần code thực nghiệm trong phần 3, kích thước chiều embedding của text của mô hình pre-trained language model được sử dụng là?

- a) 512
- b) 768
- c) 576
- d) 1024

## 50.4 Phụ lục

1. **Hint:** Dựa vào file tải về [Text-to-Image Using DCGAN-BERTs](#) để hoàn thiện các đoạn code.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, admin mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

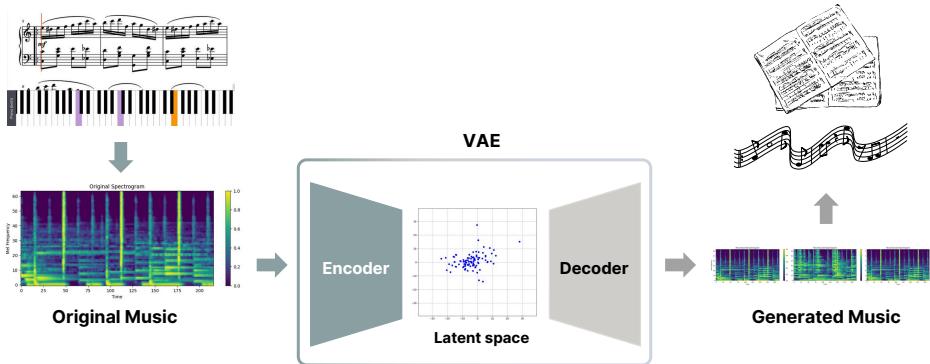
- *Hết* -

# Chương 51

## Project 1: Xây dựng hệ thống sinh nhạc dùng VAE

### 51.1 Giới thiệu

**Music Generation** là một bài toán thuộc lĩnh vực Xử lý âm thanh và các mô hình tạo sinh (generative models), liên quan đến việc xây dựng một hệ thống có khả năng tạo ra các đoạn nhạc mới dựa trên một đoạn nhạc mẫu (sound snippet) hoặc các tham số âm thanh nhất định. Trên thế giới, các dự án như [Magenta](#) của Google, [AIVA \(Artificial Intelligence Virtual Artist\)](#) hay [OpenAI Jukebox](#) đều là những ví dụ nổi tiếng trong lĩnh vực này. Mục tiêu của bài toán sinh nhạc không chỉ dừng lại ở việc sao chép phong cách của mẫu nhạc, mà còn hướng đến sự biến tấu sáng tạo, để có thể sinh ra các đoạn nhạc hoàn toàn mới và độc đáo.



Hình 51.1: Minh họa về bài toán sinh nhạc dựa trên một mẫu nhạc cho trước.

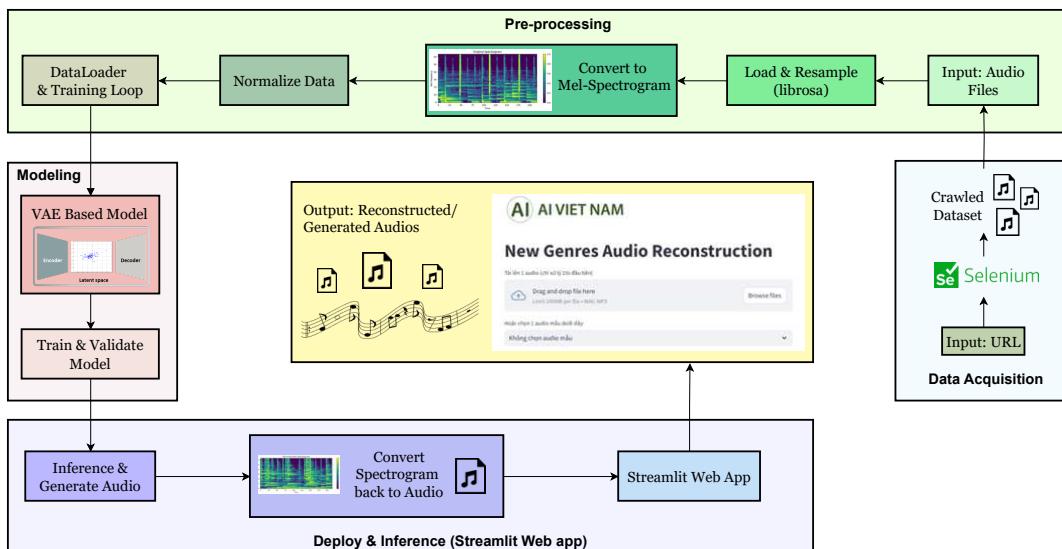
Trong project này, chúng ta sẽ cùng tìm hiểu phát triển một chương trình tạo sinh nhạc mới dựa trên một tín hiệu âm thanh gốc kèm theo một danh sách nhãn các thể loại nhạc. Tổng quan, Input và Output của bài toán như sau:

- **Input:** Một mẫu âm thanh và danh sách nhãn thể loại âm thanh đầu ra mong muốn.
- **Output:** Đoạn âm thanh mới được sinh ra, mang đặc trưng của đoạn âm thanh mẫu giao thoa với danh sách các nhãn thể loại đầu vào.

Mô hình được ứng dụng trong bài dựa trên kiến trúc mô hình *Variational Autoencoder (VAE)*. Theo đó, hệ thống sẽ học cách biểu diễn đặc trưng của âm thanh piano và nội dung các thể loại nhạc đi kèm thuộc các file âm thanh khác nhau vào trong một không gian tiềm ẩn (latent space), sau đó tận dụng không gian tiềm ẩn này để tạo ra những đoạn nhạc mới, mang đặc trưng tương đồng với mẫu nhạc gốc nhưng vẫn đảm bảo tính sáng tạo.

### 51.1.1 Project pipeline

Dựa trên các mô tả nội dung trên, ta có một pipeline tổng quát cho toàn bộ project được mô tả như ảnh sau:



Hình 51.2: Pipeline tổng quan của project.

Pipeline này có 4 giai đoạn chính:

- **Data Acquisition (Thu thập dữ liệu):** Dữ liệu âm thanh được thu thập từ web bằng Selenium.
- **Pre-processing (Tiền xử lý):** Dữ liệu được tải lên, xử lý và chuyển đổi sang định dạng Mel-Spectrogram.
- **Modeling (Huấn luyện mô hình):** Mô hình VAE được sử dụng để học cách biểu diễn âm thanh.
- **Deployment (Triển khai):** Mô hình được dùng để tạo dữ liệu âm thanh mới và hiển thị trên ứng dụng web.

### Data Acquisition (Thu thập dữ liệu)

Dữ liệu âm thanh đầu vào được thu thập từ nhiều nguồn khác nhau. Đầu tiên, ta xác định các nguồn dữ liệu âm thanh phù hợp như các website lưu trữ nhạc miễn phí hoặc kho dữ liệu công khai. Sau đó, ta sử dụng Selenium để tự động thu thập dữ liệu từ các trang web, trích xuất danh sách các tệp âm thanh và siêu dữ liệu liên quan (tên bài hát, nghệ sĩ, thể loại, v.v.). Các đường dẫn tệp âm thanh này sau đó được sử dụng để tải xuống dữ liệu và lưu trữ chúng theo định dạng có tổ chức (ví dụ: thư mục theo thể loại, JSON chứa metadata, v.v.).

### Pre-processing (Tiền xử lý)

Dữ liệu âm thanh sau khi thu thập được cần qua các bước tiền xử lý trước khi đưa vào mô hình. Đầu tiên, các tập tin âm thanh được tải và chuẩn hóa tần số lấy mẫu bằng thư viện `librosa`. Tiếp theo, các đoạn âm thanh dài được chia nhỏ để tối ưu hóa việc huấn luyện. Sau đó, dữ liệu được chuyển đổi sang dạng Mel-Spectrogram bằng `librosa.feature.melspectrogram()`, giúp trích xuất đặc trưng quan trọng từ tín hiệu âm thanh. Cuối cùng, dữ liệu được chuẩn hóa bằng Min-Max Scaling hoặc Standardization để đảm bảo giá trị đầu vào có cùng thang đo, giúp mô hình học tốt hơn. Sau đó, dữ liệu được đưa vào `torch.utils.data.DataLoader` để tạo bộ tải dữ liệu và thiết lập vòng lặp huấn luyện với batch size phù hợp.

### Modeling (Huấn luyện mô hình)

Mô hình chính được sử dụng là Biến Autoencoder (VAE), giúp học cách mã hóa và giải mã dữ liệu âm thanh. Mô hình gồm ba thành phần chính: **Encoder**, biến đổi đầu vào thành không gian tiềm ẩn; **Latent Space**, nơi mô hình học biểu diễn đặc trưng của âm thanh; và **Decoder**, tái tạo lại âm thanh từ không gian tiềm ẩn. Trong quá trình huấn luyện, ta tính toán hàm mất mát gồm *reconstruction loss* và *KL divergence loss*, sau đó kiểm tra mô hình trên tập validation để đánh giá hiệu suất.

### Deployment (Triển khai)

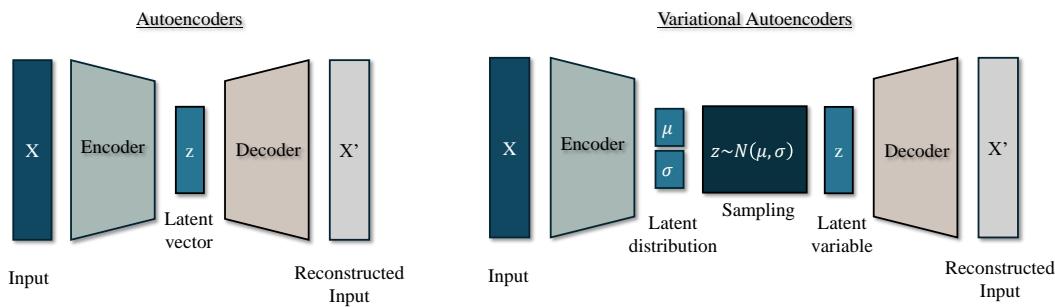
Sau khi huấn luyện, mô hình được sử dụng để tạo ra các mẫu âm thanh mới từ không gian tiềm ẩn. Kết quả Mel-Spectrogram sau đó được chuyển đổi lại thành dạng sóng âm thanh bằng *librosa*. Để người dùng có thể tương tác, ta hiển thị Mel-Spectrogram của âm thanh gốc và âm thanh tái tạo, đồng thời cho phép phát lại trực tiếp trên ứng dụng. Một ứng dụng web dựa trên Streamlit được xây dựng để người dùng dễ dàng thử nghiệm và nghe các mẫu âm thanh đã tạo. Cuối cùng, các tệp âm thanh đầu ra được lưu trữ để đánh giá và cải thiện mô hình trong tương lai.

#### 51.1.2 Tổng quan về VAE

Mục tiêu của pipeline là tạo ra một mô hình có khả năng không chỉ tái tạo mà còn sinh ra dữ liệu mới từ các đặc trưng trích xuất được. Để đạt được điều này, chúng ta hướng tới kiến trúc **Variational Autoencoder (VAE)**.

Khác với **Autoencoder (AE)** truyền thống, vốn chỉ ánh xạ dữ liệu đầu vào thành một điểm cụ thể trong không gian tiềm ẩn (*latent space*), VAE chuyển đổi đầu vào thành một phân phối xác suất trong latent space. Điều này cho phép mô hình sinh ra dữ liệu mới bằng cách lấy mẫu từ phân phối xác suất đó, từ đó duy trì tính mạch lạc và khả năng tổng quát của thông tin.

## Giới thiệu chung về Autoencoder (AE) và Variational Autoencoder (VAE)



Hình 51.3: Tổng quan về AE và VAE.

**Autoencoder (AE):** là một mạng nơ-ron sử dụng kiến trúc encoder-decoder để nén dữ liệu đầu vào vào một không gian thấp chiều (*low-dimensional latent space*) và tái tạo lại dữ liệu đó.

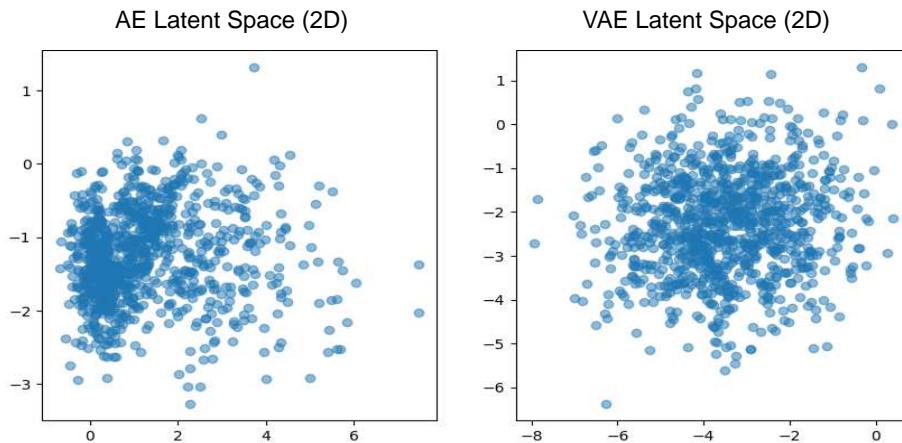
- **Encoder:** Biến đổi dữ liệu đầu vào thành một vector duy nhất trong latent space.
- **Decoder:** Tái tạo dữ liệu từ vector đó.

**Variational Autoencoder (VAE):** cải tiến AE bằng cách ánh xạ đầu vào thành một phân phối xác suất trong latent space (thường là Gaussian). Cụ thể:

- **Encoder** của VAE xuất ra hai tham số: **mean**  $\mu$  và **variance**  $\sigma^2$ , từ đó xây dựng phân phối  $q(z | x)$ .
- **Latent representation** không còn là một điểm duy nhất mà là một phân phối, cho phép sinh dữ liệu mới bằng cách lấy mẫu (sampling) từ đó.

Hình 51.4 dưới đây giúp ta thấy rõ nhược điểm của AE so với VAE. AE ánh xạ dữ liệu vào các điểm rời rạc trong không gian tiềm ẩn, dẫn đến sự phân bố không liên tục và có nhiều khoảng trống. Trong khi đó, VAE học được

một phân phối xác suất có dạng Gaussian, giúp latent space liên tục và có cấu trúc hơn.



Hình 51.4: So sánh latent space (không gian tiềm ẩn) 2D của AE và VAE.

### Các khái niệm cơ bản cần biết trong VAE

- **Latent Space (Không gian tiềm ẩn):** Là không gian trừu tượng nơi dữ liệu đầu vào được biểu diễn sau khi được mã hóa. Trong VAE, thay vì chỉ có một điểm, mỗi đầu vào được biểu diễn dưới dạng một phân phối (thường là Gaussian).
- **Latent Representation (Biểu diễn tiềm ẩn):** Là cách dữ liệu được biểu diễn trong latent space, thường dưới dạng các tham số như  $\mu$  (mean) và  $\sigma^2$  (variance) của phân phối Gaussian.
- **Tóm tắt cơ sở lý thuyết:**
  - $p(\mathbf{x})$ : *Data distribution* (phân phối dữ liệu), biểu diễn phân phối của toàn bộ tập dữ liệu đầu vào.
  - $p(\mathbf{z})$ : *Prior* (phân phối tiên nghiệm) của biến tiềm ẩn  $z$ , phản ánh giả định ban đầu về không gian tiềm ẩn trước khi quan sát dữ liệu.
  - $p(\mathbf{x}|\mathbf{z})$ : *Likelihood* (xác suất tái tạo dữ liệu  $x$  từ  $z$ ), mô tả cách dữ liệu được tạo ra từ không gian tiềm ẩn.

- $q(z|x)$ : *Posterior* (phân phối hậu nghiệm), đại diện cho phân phối của  $z$  sau khi đã quan sát  $x$ .

Trong thực tế, phân phối  $p(z)$  thường không được biết chính xác, khiến các tính toán trực tiếp trở nên khó khăn. Do đó, VAE giả định  $p(z)$  tuân theo một phân phối chuẩn  $\mathcal{N}(0, I)$  để đơn giản hóa mô hình.

Tuy nhiên, việc tính toán  $p(z|x)$  trực tiếp là không khả thi. Thay vào đó, VAE sử dụng  $q(z|x)$  để xấp xỉ phân phối này. Mục tiêu của mô hình là tối ưu hóa hàm *ELBO* (Evidence Lower Bound) để làm cho  $q(z|x)$  gần nhất với  $p(z|x)$ .

Lý thuyết VAE dựa trên định lý Bayes, trong đó  $q(z|x)$  được sử dụng để xấp xỉ  $p(z|x)$ . Để tìm hiểu sâu hơn có thể đọc thêm tại [đây](#).

### Loss function của VAE

Hàm mất mát của VAE được gọi là ELBO (Evidence Lower Bound) bao gồm hai thành phần chính:

$$L(x) = \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{KL}(q(z|x) \parallel p(z))$$

Trong đó:

- $\mathbb{E}_{q(z|x)}[\log p(x|z)]$ : **Reconstruction Loss** (thường dùng MSE hoặc BCE), đo lường khả năng tái tạo dữ liệu từ latent space.
- $D_{KL}(q(z|x) \parallel p(z))$ : **KL Divergence**, đo khoảng cách giữa phân phối xấp xỉ  $q(z|x)$  và phân phối chuẩn  $p(z)$  (thường là  $\mathcal{N}(0, I)$ ).

Mục tiêu của VAE là tối ưu hóa ELBO (Evidence Lower Bound):

$$\log p(x) \geq \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{KL}(q(z|x) \parallel p(z))$$

Bằng cách tối đa hóa ELBO, ta đồng thời giảm thiểu lỗi tái tạo và đảm bảo phân phối tiềm ẩn có cấu trúc hợp lý.

### Reparameterization Trick

Trong VAE, ta cần tối ưu hóa  $q(z|x)$ , tức là phải lấy mẫu  $z \sim \mathcal{N}(\mu, \sigma^2)$ . Tuy nhiên, phép lấy mẫu là một quá trình không khả vi, gây khó khăn cho việc lan truyền gradient trong quá trình huấn luyện mô hình.

Để giải quyết vấn đề này, VAE sử dụng **Reparameterization Trick**, giúp biến đổi phép lấy mẫu thành một phép toán khả vi:

$$z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$

Cách tiếp cận này cho phép tính đạo hàm qua các tham số  $\mu$  và  $\sigma$ , giúp huấn luyện mô hình bằng các thuật toán tối ưu hóa dựa trên gradient (như Adam).

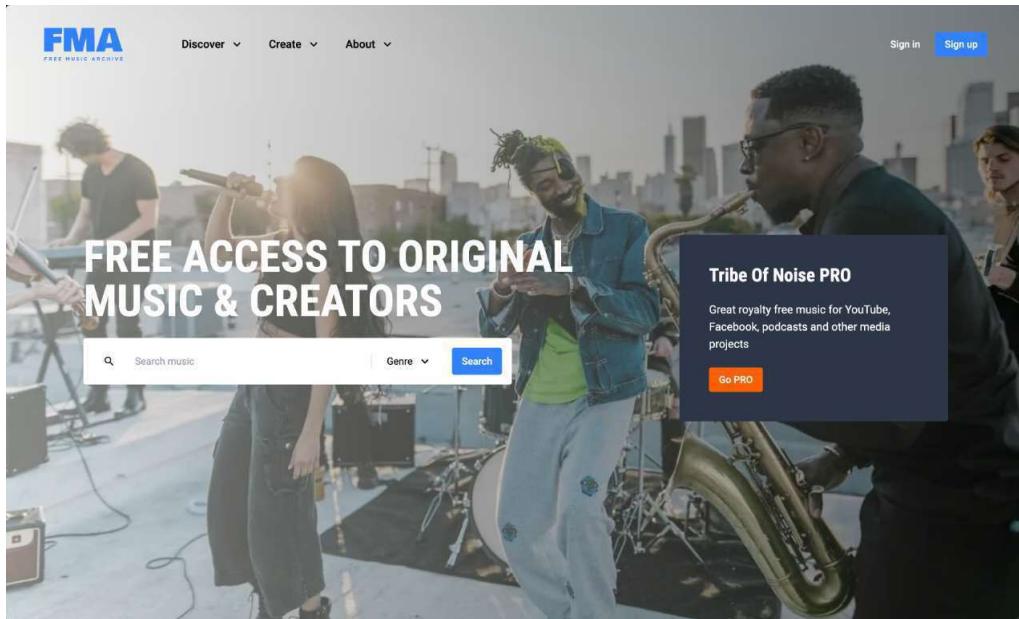
Variational Autoencoder (VAE) là một bước tiến vượt bậc so với Autoencoder truyền thống, nhờ vào việc sử dụng latent space có cấu trúc xác suất. Điều này không chỉ giúp tái tạo dữ liệu đầu vào mà còn cho phép mô hình sinh ra các mẫu dữ liệu mới, mở ra nhiều ứng dụng trong tạo ảnh, tổng hợp văn bản, và nhiều lĩnh vực khác.

## 51.2 Cài đặt chương trình

Trong phần này, chúng ta sẽ tìm hiểu về quá trình xây dựng toàn bộ chương trình Music Generation sử dụng VAE. Trong đó, bao gồm hai phần nội dung lớn là thu thập bộ dữ liệu để huấn luyện mô hình và xây dựng mô hình VAE.

### 51.2.1 Thu thập bộ dữ liệu

Tại project này, chúng ta giả định trong trường hợp chưa có một bộ dữ liệu có sẵn. Vì vậy, việc thu thập dữ liệu cần phải được thực hiện. Dựa vào nội dung của project, một lượng các file âm thanh có tiếng piano cần được thu thập để có thể huấn luyện mô hình VAE. Có rất nhiều nguồn cũng như cách thức để thu thập dữ liệu, trong đó, nguồn từ internet là một trong những nguồn thu thập dễ tiếp cận nhất. Đối với phạm vi của project này, chúng ta sẽ sử dụng thư viện Selenium để thực hiện thu thập dữ liệu trên một trang web chuyên lưu trữ các file âm nhạc.



Hình 51.5: Trang chủ của website [freemusicarchive.org](https://freemusicarchive.org).

**Lưu ý:** Các bạn có thể tải trực tiếp bộ dữ liệu đã được thu thập sẵn ở phần 60.4 và bỏ qua phần này.

## Cài đặt Chrome Driver và Selenium

Chúng ta sẽ thực hiện việc thu thập dữ liệu bằng Selenium trên Colab. Để sử dụng Selenium, chúng ta cần thực thi một số lệnh cài đặt phức tạp sau đây. Tóm quan, Colab sẽ cần được cài đặt Chrome Driver và Selenium. Đoạn mã bash có nội dung như sau:

```
1 %%shell
2 # Ubuntu no longer distributes chromium-browser outside of snap
3 #
4 # Proposed solution: https://askubuntu.com/questions/1204571/how-to-
5 # install-chromium-without-snap
6
7 # Add debian buster
8 cat > /etc/apt/sources.list.d/debian.list << "EOF"
9 deb [arch=amd64 signed-by=/usr/share/keyrings/debian-buster.gpg] http://
10 deb.debian.org/debian buster main
11 deb [arch=amd64 signed-by=/usr/share/keyrings/debian-buster-updates.gpg]
12 http://deb.debian.org/debian buster-
13 updates main
14 deb [arch=amd64 signed-by=/usr/share/keyrings/debian-security-buster.gpg]
15 http://deb.debian.org/debian-security
16 buster/updates main
17
18 EOF
19
20 # Add keys
21 apt-key adv --keyserver keyserver.ubuntu.com --recv-keys DCC9EFBF77E11517
22 apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 648ACFD622F3D138
23 apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 112695A0E562B32A
24
25 apt-key export 77E11517 | gpg --dearmour -o /usr/share/keyrings/debian-
26 buster.gpg
27 apt-key export 22F3D138 | gpg --dearmour -o /usr/share/keyrings/debian-
28 buster-updates.gpg
29 apt-key export E562B32A | gpg --dearmour -o /usr/share/keyrings/debian-
30 security-buster.gpg
31
32 # Prefer debian repo for chromium* packages only
33 # Note the double-blank lines between entries
34 cat > /etc/apt/preferences.d/chromium.pref << "EOF"
35 Package: *
36 Pin: release a=eoan
37 Pin-Priority: 500
38
39
```

```
30 Package: *
31 Pin: origin "deb.debian.org"
32 Pin-Priority: 300
33
34
35 Package: chromium*
36 Pin: origin "deb.debian.org"
37 Pin-Priority: 700
38 EOF
39
40 # Install chromium and chromium-driver
41 apt-get update
42 apt-get install chromium chromium-driver
43
44 # Install selenium
45 pip install selenium
```

## Import các thư viện cần thiết

Sau khi tải và cài đặt thành công Selenium và Chrome Driver, ta tiến hành import các thư viện cần thiết để sử dụng được đoạn code thu thập dữ liệu như sau:

```
1 import os
2 import requests
3 import time
4 import json
5
6 from tqdm import tqdm
7 from selenium import webdriver
8 from selenium.webdriver.chrome.service import Service
9 from selenium.webdriver.common.by import By
10 from selenium.webdriver.support.ui import WebDriverWait
11 from selenium.webdriver.support import expected_conditions as EC
```

## Khởi tạo trình duyệt

Bắt đầu chương trình, ta cần khởi tạo một Chrome Driver để có thể truy cập vào các trang web thông qua driver này. Một số cài đặt `chrome_options` được thêm vào trong đoạn code dưới đây liên quan đến driver để phù hợp với môi trường Colab:

```

1 WEBDRIVER_DELAY_TIME_INT = 10
2 TIMEOUT_INT = 10
3 service = Service(executable_path=r"/usr/bin/chromedriver")
4 webdriver = Chrome(service=service)
5 chrome_options = webdriver.ChromeOptions()
6 chrome_options.add_argument("--headless")
7 chrome_options.add_argument("--no-sandbox")
8 chrome_options.add_argument("--disable-dev-shm-usage")
9 chrome_options.add_argument("window-size=1920x1080")
10 chrome_options.headless = True
11 driver = webdriver.Chrome(service=service, options=chrome_options)
12 driver.implicitly_wait(TIMEOUT_INT)
13 wait = WebDriverWait(driver, WEBDRIVER_DELAY_TIME_INT)

```

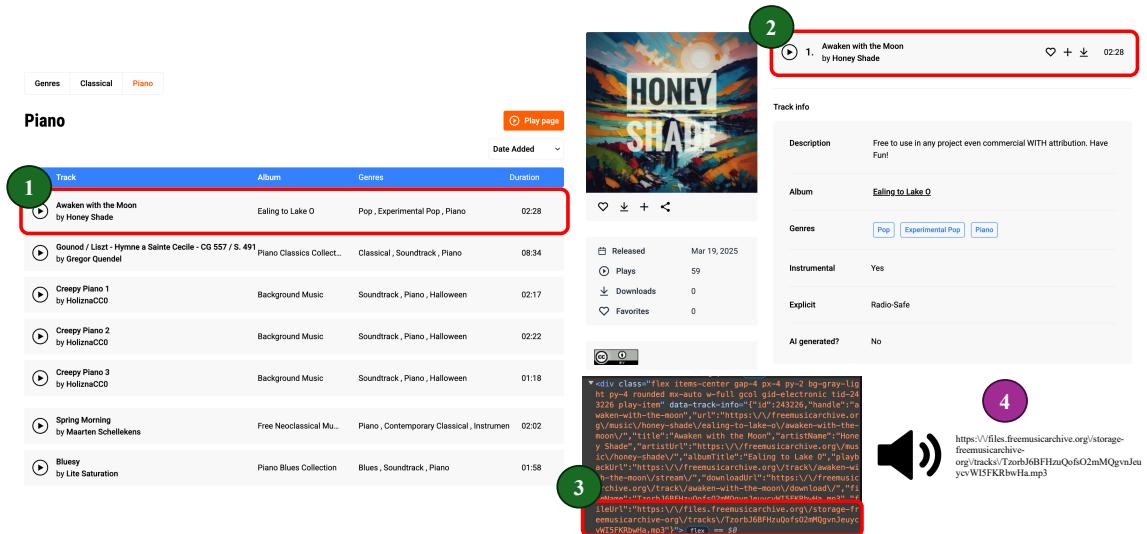
### Xây dựng hàm tách thông tin từ danh sách nhạc

Ở trang web nhạc này, mục tiêu của chúng ta là, với một bảng danh sách các nhạc của một thể loại (genres) nào đó, xây dựng một chương trình trích xuất và duyệt qua được từng hàng nội dung âm thanh một. Nguyên nhân là vì,

```

1 def extract_audio_links_from_menu(menu_url, driver):
2 driver.get(menu_url)
3 container = wait.until(EC.presence_of_element_located(
4 (By.CSS_SELECTOR, "div.w-full.flex.flex-col.gap-3.pt-3")
5))
6 play_items = container.find_elements(By.CSS_SELECTOR, "div.play-item")
7 links = []
8 for item in play_items:
9 try:
10 a_tag = item.find_element(By.CSS_SELECTOR, ".pxt-track a")
11 link = a_tag.get_attribute("href")
12 links.append(link)
13 except Exception:
14 continue
15 return links

```



Hình 51.6: Tổng quan các bước để tìm và trích xuất được nội dung file âm thanh có thể tải về máy tự động. Các file âm thanh này thuộc từng hàng nội dung bài nhạc trên bảng menu.

### Xây dựng hàm tải file âm thanh

Ta xây dựng hàm tải file với đầu vào là một đường dẫn cho trước. Về mặt kỹ thuật, hàm này hoàn toàn có thể sử dụng để tải bất cứ file dữ liệu nào trên mạng, song trong trường hợp này chúng ta chỉ dùng để tải file âm thanh. Nội dung hàm này như sau:

```

1 def download_audio_file(file_url, filepath):
2 response = requests.get(file_url, stream=True)
3 if response.status_code == 200:
4 with open(filepath, "wb") as f:
5 for chunk in response.iter_content(chunk_size=8192):
6 f.write(chunk)
7 else:
8 print(f"Error downloading file from {file_url}; status: {response.status_code}")

```

### Xây dựng các hàm trích xuất nội dung và thông tin trang nhạc

Quan sát các thành phần/nội dung được hiển thị trên trang web của bài nhac, có thể nhận thấy bên cạnh file âm thanh còn có những thông tin hữu ích mà ta hoàn toàn có thể kéo về đồng thời. Các thông tin đi kèm này hoàn toàn có thể được sử dụng cho các mục đích, ý tưởng khác nhằm mở rộng bài project sau này.

The screenshot shows a music player interface with the following details:

1. Awaken with the Moon  
by Honey Shade      02:28

Track info

Description	Free to use in any project even commercial WITH attribution. Have Fun!
Album	<a href="#">Ealing to Lake O</a>
Genres	Pop   Experimental Pop   Piano
Instrumental	Yes
Explicit	Radio-Safe
AI generated?	No

Hình 51.7: Ví dụ minh họa về một bảng thông tin được cung cấp sẵn trên trang web cho một bài nhạc.

Dựa trên những gì được hiển thị trong hình 51.7, ta sẽ xây dựng các hàm riêng biệt để trích xuất giá trị các trường thông tin. Các hàm đều nhận đầu vào là chrome driver đã được truy cập sẵn vào website của bài nhạc. Ý nghĩa của các đoạn code được điều chỉnh phỏng theo cấu trúc HTML hiện thời của

trang web để có thể trích được thông tin tương ứng.

### 1. Hàm tách nội dung đường dẫn chứa file âm thanh

```

1 def extract_track_info(driver):
2 audio_div = WebDriverWait(driver, 15).until(
3 EC.presence_of_element_located((By.CSS_SELECTOR, "div[data-track-
4 info]"))
5)
6 return json.loads(audio_div.get_attribute("data-track-info"))

```

### 2. Hàm tách các nhãn thẻ loại nhạc

```

1 def extract_genres(driver):
2 try:
3 genre_elem = driver.find_element(By.CSS_SELECTOR, "span.md\\\:col-
4 span-6.flex.flex-wrap.gap-3")
5 return [a.text.strip() for a in genre_elem.find_elements(By.
6 TAG_NAME, "a") if a.text.strip()]
7 except Exception:
8 return []

```

### 3. Hàm tách thời lượng nhạc

```

1 def extract_duration(driver):
2 try:
3 duration_elem = driver.find_element(By.CSS_SELECTOR, "span.w-12.ml-
4 -auto.md\\\:ml-0.col-span-2.inline-
5 flex.justify-end.items-center")
6 return duration_elem.text.strip()
7 except Exception:
8 return ""

```

### 4. Hàm tách một số thông tin khác

Hiện tại hàm này sẽ tách thông tin cho ta biết rằng liệu bài nhạc có phải là sản phẩm từ mô hình AI hay không? Bài nhạc này có lời hay không?

```

1 def extract_extra_info(driver):
2 instrumental = "No"
3 ai_generated = "No"
4 try:
5 info_container = driver.find_element(By.CSS_SELECTOR, "div.px-8.py-
6 -2.bg-gray-light.flex.flex-col.divide-
7 y.divide-gray")
8 info_divs = info_container.find_elements(By.CSS_SELECTOR, "div.
9 grid.grid-cols-1.md\\\:grid-cons-8.py-

```

```

 6")
7 for div in info_divs:
8 label = div.find_element(By.CSS_SELECTOR, "span.font-\\"500
9 \\"].md\\":col-span-2").text.strip()
10 value = div.find_element(By.CSS_SELECTOR, "span.md\\":col-span-
11 6").text.strip()
12 if "Instrumental" in label:
13 instrumental = value
14 if "AI generated?" in label:
15 ai_generated = value
16 except Exception:
17 pass
18 return instrumental, ai_generated

```

Cuối cùng, tổng hợp tất cả các hàm trên với từng vai trò riêng biệt, ta tạo một hàm nhận đầu vào là một đường dẫn đến website của một bài nhạc bất kì. Hàm này sau đó sử dụng driver để truy cập vào trang web và lần lượt gọi các hàm đã định nghĩa ở trên để trích xuất thông tin, cuối cùng tổng hợp lại thành một dictionary để lưu xuống máy dưới dạng file .json cũng như tải file âm thanh tương ứng.

```

1 def process_audio_page(audio_url, driver, index):
2 driver.get(audio_url)
3
4 track_info = extract_track_info(driver)
5 file_url = track_info.get("fileUrl", "")
6 audio_name = track_info.get("title", "").strip()
7 author = track_info.get("artistName", "").strip()
8
9 genres = extract_genres(driver)
10 duration = extract_duration(driver)
11 instrumental, ai_generated = extract_extra_info(driver)
12
13 metadata = {
14 "audioName": audio_name,
15 "author": author,
16 "genres": genres,
17 "instrumental": instrumental,
18 "ai_generated": ai_generated,
19 "duration": duration,
20 "audio_url": audio_url
21 }
22
23 audio_filename = f"audio_{index:04d}.mp3"
24 meta_filename = f"audio_{index:04d}.json"

```

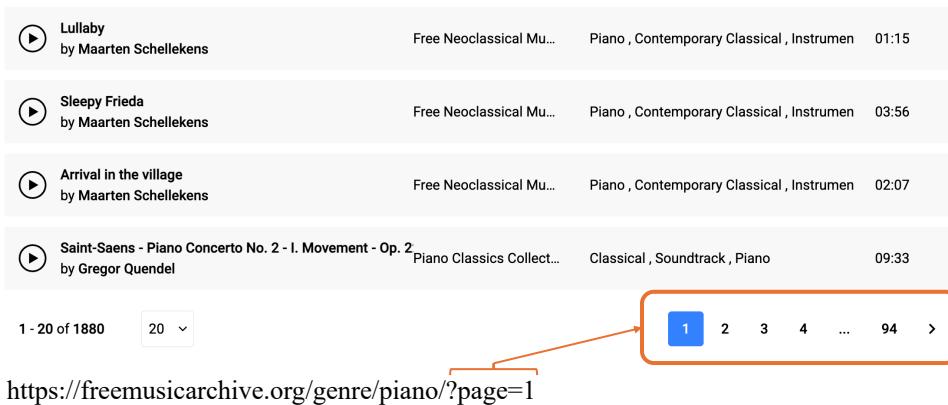
```

25 audio_filepath = os.path.join("crawled_data", "audio", audio_filename)
26 meta_filepath = os.path.join("crawled_data", meta_filename)
27
28 download_audio_file(file_url, audio_filepath)
29
30 with open(meta_filepath, "w", encoding="utf-8") as f:
31 json.dump(metadata, f, ensure_ascii=False, indent=4)
32
33 return metadata

```

### Xây dựng hàm duyệt từng danh sách nhạc

Có thể thấy ở bảng menu danh sách nhạc, ta có thể chuyển sang nhiều menu (page) khác sau đó. Dựa theo danh sách ở thẻ loại Piano, có đến 94 trang menu với các bài nhạc khác nhau. Chính vì vậy, chúng ta cần một hàm có thể tự động chuyển sang các page khác để có thể lấy nhiều bài nhạc hơn. Dựa trên thông tin có ở trang web, ta xây dựng hàm này như sau:



Hình 51.8: Minh họa các thông tin giúp ta có thể duyệt qua các page danh sách nhạc tự động.

```

1 def loop_over_menu_pages(base_url, total_pages, driver):
2 all_links = []
3 for page in tqdm(range(1, total_pages + 1), desc="Extracting Links",
4 unit="page"):
4 page_url = f"{base_url}?page={page}"
5 try:

```

```

6 links = extract_audio_links_from_menu(page_url, driver)
7 all_links.extend(links)
8 except Exception as e:
9 print(f"Error on page {page}: {e}")
10 return all_links

```

## Thực thi quá trình thu thập dữ liệu

Cuối cùng, tổng hợp toàn bộ các code phía trên, ta triển khai việc thu thập dữ liệu âm thanh trên trang web [freemusicarchive.org](https://freemusicarchive.org). Trong nội dung project này, ta chỉ xem xét thu thập các bài nhạc có thể loại là **Piano**. Song, các bạn hoàn toàn có thể điều chỉnh để thu thập các thể loại nhạc khác. Code triển khai như sau:

```

1 os.makedirs("crawled_data", exist_ok=True)
2 os.makedirs(os.path.join("crawled_data", "audio"), exist_ok=True)
3
4 base_url = "https://freemusicarchive.org/genre/piano/"
5 total_pages = 10
6 sample_idx = 1
7 audio_links = loop_over_menu_pages(base_url, total_pages, driver)
8 print(f"Total audio links extracted: {len(audio_links)}")
9
10 for audio_url in tqdm(audio_links, desc="Downloading Audio and Metadata",
11 unit="audio"):
12 try:
13 process_audio_page(audio_url, driver, sample_idx)
14 sample_idx += 1
15 except:
16 print(f'Error at: {audio_url}')
17 continue
18 time.sleep(0.5)
19 driver.quit()

```

Quá trình thu thập sẽ diễn ra nhanh hay chậm tùy thuộc vào số lượng trang page mà chúng ta muốn tải về. Khi kết thúc thực thi đoạn mã trên, ta nhận được một thư mục **crawled\_data** có cấu trúc như sau:

Như vậy, quá trình thu thập dữ liệu đã hoàn tất, và chúng ta đã có được một bộ dữ liệu âm thanh Piano để tiến hành huấn luyện mô hình trong phần sau.

## 51.2.2 Xây dựng mô hình

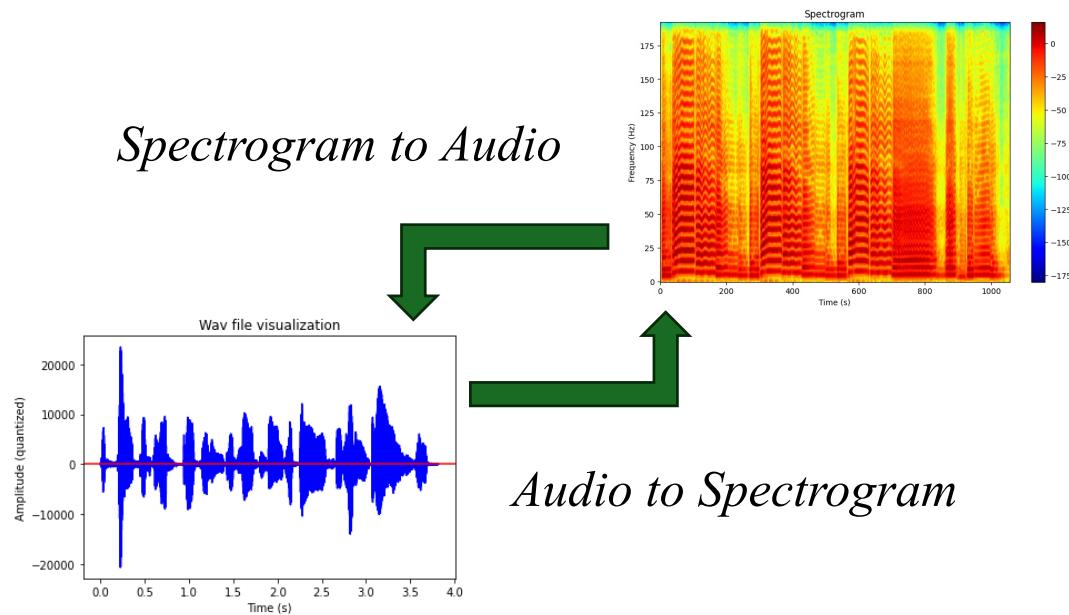
### Import các thư viện cần thiết

Trước tiên, chúng ta cần tải các thư viện không có sẵn trên colab với phiên bản phù hợp:

```
1 !pip install numba==0.61.0 torchaudio==2.6.0 librosa==0.10.2.post1
```

Sau đó, thực hiện import các thư viện cần thiết cho chương trình project này:

```
1 import os
2 import torch
3 import torch.nn as nn
4 import torch.optim as optim
5 import torchaudio
6 import torchaudio.transforms as T
7 import numpy as np
8 import librosa
9 import librosa.display
10 import IPython.display as ipd
11 import torch.nn.functional as F
12 import matplotlib.pyplot as plt
13 import json
14
15 from sklearn.preprocessing import MinMaxScaler
16 from torch.utils.data import Dataset, DataLoader
17 from IPython.display import Audio
18 from tqdm import tqdm
19
20 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```



Hình 51.9: Minh họa về việc chuyển đổi tín hiệu âm thanh sang ảnh mel spectrogram và ngược lại.

### Tải bộ dữ liệu

Ta tải bộ dữ liệu đã được thu thập từ phần trước và giải nén để sử dụng:

```

1 !gdown 1u2WzsWUlyZbbPDfXAWXuLRMTwFkT21wa
2 !unzip -q crawled_piano_audio_40_pages.zip -d piano_audio_files_40_pages
3 os.remove("crawled_piano_audio_40_pages.zip")

```

### Tiền xử lý bộ dữ liệu âm thanh

Để mô hình có thể xử lý và hiểu được các đặc trưng từ dữ liệu âm thanh một cách hiệu quả, chúng ta không trực tiếp sử dụng dữ liệu đọc được từ file âm thanh gốc (tín hiệu âm thanh miền thời gian) mà sẽ thực hiện một vài bước chuyển đổi để chuyển thành dạng biểu diễn mel spectrogram của âm thanh (hình 51.9). Đây là một kiểu biểu diễn truyền thống trong xử lý tín hiệu âm thanh vì nó mang lại nhiều thông tin có lợi hơn cho việc học của mô hình. Theo đó, chúng ta sẽ khai báo một số hàm như sau cho quá trình tiền xử lý toàn bộ dữ liệu:

- Đọc tệp json và lấy thông tin thể loại `load_and_get_genres()`.
- Đọc và lấy mẫu âm thanh `load_and_resample_audio()`.
- Chuyển dữ liệu âm thanh gốc sang dạng mel spectrogram `audio_to_melspec()` và ngược lại `melspec_to_audio()`.
- Hiển thị biểu đồ mel spectrogram `show_spectrogram()`.
- Chuẩn hoá các giá trị về miền (0, 1) để phù hợp khi đưa vào mô hình `normalize_melspec()` và khử chuẩn hoá để đưa về giá trị trước khi chuẩn hoá `denormalize_melspec()`.
- Hiển thị âm thanh với `display_audio_files()`.

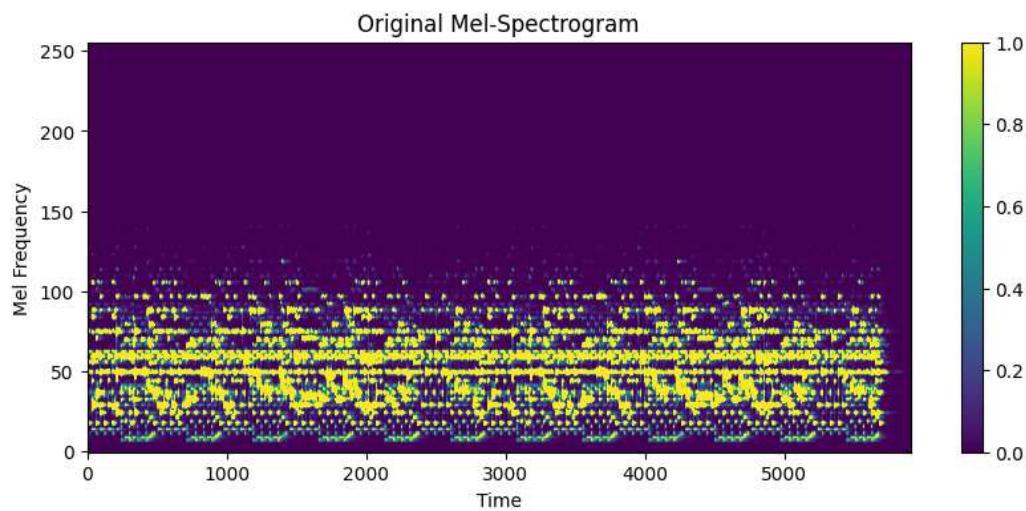
```
1 def load_and_get_genres(json_path):
2 with open(json_path, "r") as f:
3 data = json.load(f)
4
5 return data.get('genres', [])
6
7 def load_and_resample_audio(file_path, target_sr=22050):
8 audio, sr = librosa.load(file_path, sr=None)
9 if sr != target_sr:
10 audio = librosa.resample(audio, orig_sr=sr, target_sr=target_sr)
11
12 return audio, target_sr
13
14 def audio_to_melspec(audio, sr, n_mels, n_fft=2048, hop_length=512, to_db=False):
15 spec = librosa.feature.melspectrogram(
16 y=audio,
17 sr=sr,
18 n_fft=n_fft,
19 hop_length=hop_length,
20 win_length=None,
21 window="hann",
22 center=True,
23 pad_mode="reflect",
24 power=2.0,
25 n_mels=n_mels
26)
27 if to_db:
28 spec = librosa.power_to_db(spec, ref=np.max)
```

```
29 return spec
30
31
32 def normalize_melspec(melspec, norm_range=(0, 1)):
33 scaler = MinMaxScaler(feature_range=norm_range)
34 melspec = melspec.T
35 melspec_normalized = scaler.fit_transform(melspec)
36
37 return melspec_normalized.T
38
39
40 def denormalize_melspec(melspec_normalized, original_melspec, norm_range=(0, 1)):
41 scaler = MinMaxScaler(feature_range=norm_range)
42 melspec = original_melspec.T
43 scaler.fit(melspec)
44 melspec_denormalized = scaler.inverse_transform(melspec_normalized.T)
45
46 return melspec_denormalized.T
47
48
49 def melspec_to_audio(melspec, sr, n_fft=2048, hop_length=512, n_iter=64):
50 if np.any(melspec < 0):
51 melspec = librosa.db_to_power(melspec)
52
53 audio_reconstructed = librosa.feature.inverse.mel_to_audio(
54 melspec,
55 sr=sr,
56 n_fft=n_fft,
57 hop_length=hop_length,
58 win_length=None,
59 window="hann",
60 center=True,
61 pad_mode="reflect",
62 power=2.0,
63 n_iter=n_iter
64)
65 return audio_reconstructed
66
67 def display_audio_files(reconstructed_audio, sr, title="", original_audio=None):
68 if original_audio is not None:
69 print("Original Audio:")
70 ipd.display(ipd.Audio(original_audio, rate=sr))
71 print("Reconstructed Audio (from Mel Spectrogram):")
```

```

72 else:
73 print(title)
74
75 ipd.display(ipd.Audio(reconstructed_audio, rate=sr))
76
77 def show_spectrogram(spectrogram, title="Mel-Spectrogram", denormalize=
78 False, is_numpy=False):
79 if not is_numpy:
80 spectrogram = spectrogram.squeeze().cpu().numpy()
81 plt.figure(figsize=(10, 4))
82 if denormalize:
83 plt.imshow(spectrogram, aspect="auto", origin="lower", cmap=
84 "viridis")
85 else:
86 plt.imshow(spectrogram, aspect="auto", origin="lower", cmap=
87 "viridis", vmin=0, vmax=1)
88 plt.title(title)
89 plt.xlabel("Time")
90 plt.ylabel("Mel Frequency")
91 plt.colorbar()
92 plt.show()

```



Hình 51.10: Biểu diễn phổ tần số Mel của tín hiệu âm thanh theo thời gian, với các màu sắc phản ánh cường độ tần số ở từng điểm thời gian.

Tiếp đến, chúng ta duyệt qua các tệp json chứa thông tin về tệp âm thanh

tương ứng, để xem có tổng cộng bao nhiêu thể loại trên toàn bộ tập dữ liệu.

```

1 json_dir = os.path.join("piano_audio_files", "crawled_data")
2 all_genres = []
3
4 for filename in os.listdir(json_dir):
5 if filename.endswith('.json'):
6 json_path = os.path.join(json_dir, filename)
7 genres = load_and_get_genres(json_path)
8 all_genres.extend(genres)
9
10 unique_genres = set(all_genres)
11 max_genres = len(unique_genres)
12 print(f"Total unique genres: {max_genres}")
13 print(f"Unique genres: {unique_genres}")

```

Ngoài ra, để bổ sung thông tin về các thể loại nhạc được gán nhãn cho file âm thanh, ta sử dụng One Hot Encoding. Do đó, cần định nghĩa thêm một số hàm sau:

```

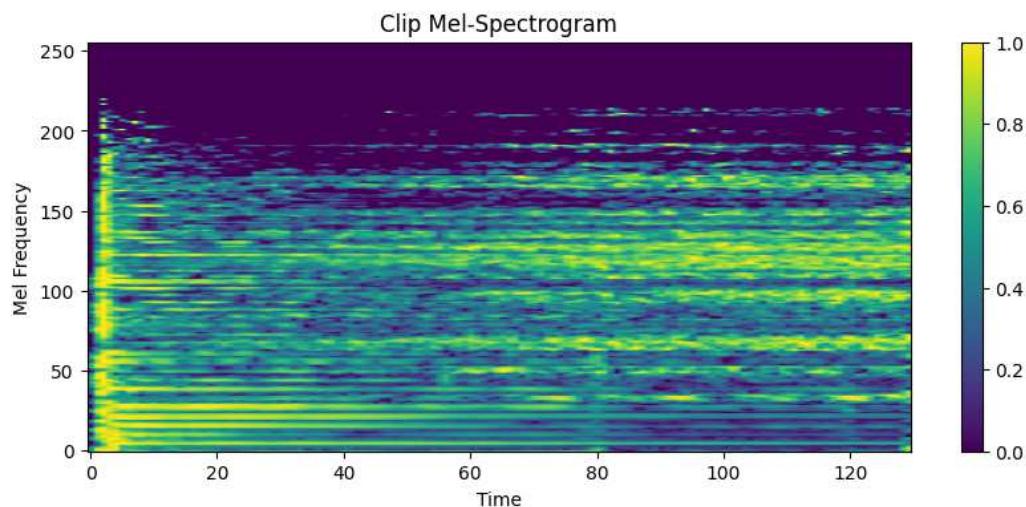
1 genres2idx = {genre: idx for idx, genre in enumerate(unique_genres)}
2 idx2genres = {idx: genre for genre, idx in genres2idx.items()}
3
4 def tokenize(genres):
5 return [genres2idx[genre] for genre in genres if genre in genres2idx]
6
7 def detokenize_tolist(tokens):
8 return [idx2genres[token] for token in tokens if token in idx2genres]
9
10 def onehot_encode(tokens, max_genres):
11 onehot = np.zeros(max_genres)
12 onehot[tokens] = 1
13 return onehot
14
15 def onehot_decode(onehot):
16 return [idx for idx, val in enumerate(onehot) if val == 1]

```

## Xây dựng PyTorch dataset

Trong phần này chúng ta sẽ xây dựng một lớp Dataset để đọc các tệp âm thanh từ *data\_dir* và thể loại tương ứng từ *json\_dir*. Với danh sách thể loại đọc được từ tệp json, ta chuyển về dạng onehot vector, trong khi mỗi tệp âm thanh sẽ được chia nhỏ thành các đoạn âm thanh ngắn hơn, cụ thể là 3

giây (*duration*). Như vậy, mỗi mẫu dữ liệu được lấy ra từ Dataset này gồm: đoạn âm thanh 3 giây đã chuẩn hoá *mel\_spec\_norm*; thể loại *genres\_input* và đoạn âm thanh 3 giây chưa chuẩn hoá *mel\_spec* dùng để giải chuẩn hoá sau này. Ngoài ra, ta sẽ trích một phần dữ liệu cho tập test.



Hình 51.11: Hình ảnh spectrogram của một mẫu âm thanh 3 giây được chuẩn hoá.

```

15 files in {data_dir}", unit="file",
16 total=len(self.files)):
17 audio, sr = load_and_resample_audio(file_path, target_sr=
18 sample_rate)
19 genres_list = load_and_get_genres(json_file_path)
20
21 genres_tokens = tokenize(genres_list)
22 genres_input = onehot_encode(genres_tokens, n_genres)
23 genres_input = torch.tensor(genres_input, dtype=torch.long).
24 unsqueeze(0)
25
26 n_samples = len(audio)
27 n_segments = n_samples // self.fixed_length
28
29 for i in range(n_segments):
30 start = i * self.fixed_length
31 end = (i + 1) * self.fixed_length
32 segment = audio[start:end]
33 mel_spec = audio_to_melspec(segment, sr, self.n_mels,
34 to_db=True)
35 mel_spec_norm = normalize_melspec(mel_spec)
36 mel_spec = torch.tensor(mel_spec, dtype=torch.float32).
37 unsqueeze(0)
38 mel_spec_norm = torch.tensor(mel_spec_norm, dtype=torch.
39 float32).unsqueeze(0)
40 audios.append((mel_spec_norm, genres_input, mel_spec))
41
42 self.audios = audios[:len(audios) - testset_amount]
43 self.testset = audios[len(audios) - testset_amount:]
44 print(f"Loaded {len(self.audios)} audio segments from {len(self.
45 files)} files, each with shape: {self.
46 .audios[0][0].shape}, {self.audios[0].
47 [1].shape}, duration: {duration}
48 seconds")
49 print(f"Test set: {len(self.testset)} audio segments")
50
51 def __len__(self):
52 return len(self.audios)
53
54 def __getitem__(self, idx):
55 mel_spec_part, genres_input, mel_spec = self.audios[idx]
56 return mel_spec_part, genres_input, mel_spec

```

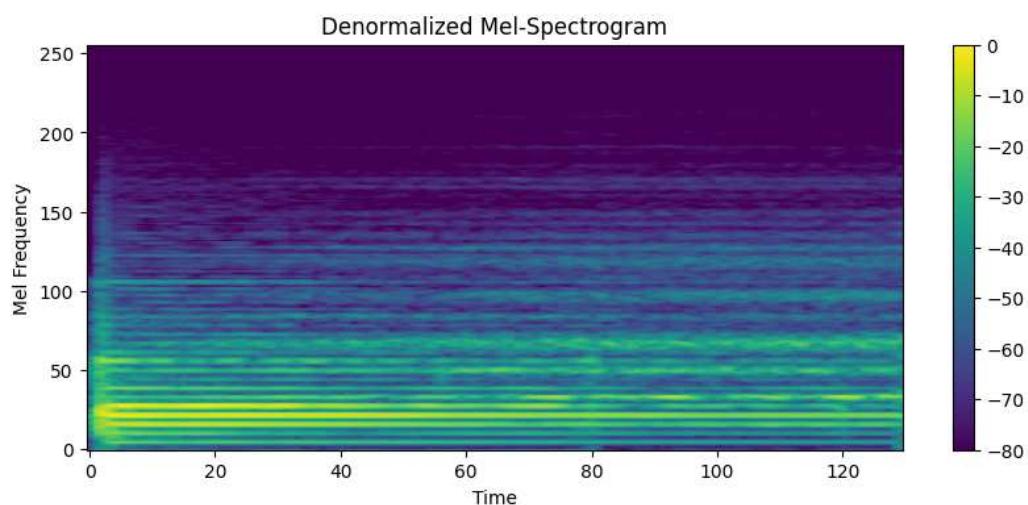
Đến đây, ta khởi tạo lớp `AudioDataset` như đã giải thích bên trên. Sau đó,

tạo trainloader và testloader.

```

1 sample_rate = 22050
2 duration = 3
3 n_mels = 256
4 batch_size = 128
5
6 audio_dir = os.path.join("piano_audio_files", "crawled_data", "audio")
7 json_dir = os.path.join("piano_audio_files", "crawled_data")
8
9 testset_amount = 32
10 trainset = AudioDataset(audio_dir, json_dir, sample_rate, duration,
11 n_mels, max_genres, testset_amount=testset_amount)
12 testset = trainset.testset
13
14 if len(trainset) == 0:
15 raise ValueError(f"No .wav file found in {audio_dir}.")
16
17 trainloader = DataLoader(trainset, batch_size=batch_size, shuffle=True,
18 num_workers=2)
18 testloader = DataLoader(testset, batch_size=testset_amount, shuffle=False,
 num_workers=2)

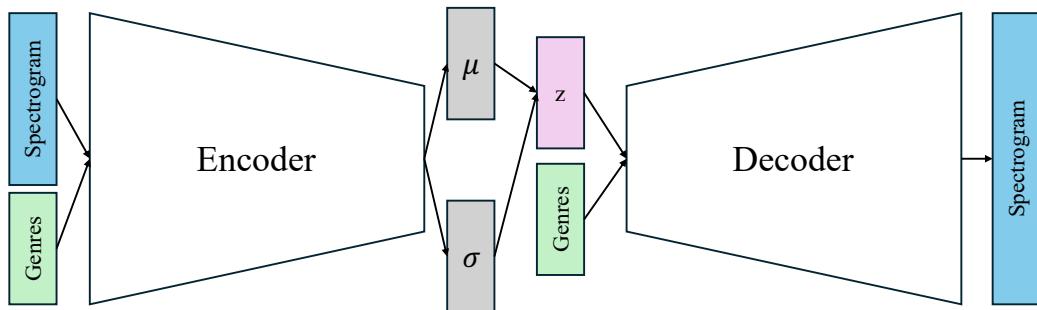
```



Hình 51.12: Hình ảnh spectrogram sau khi được khử chuẩn hoá từ hình spectrogram của hình 51.11.

## Xây dựng mô hình Conditional Variational Autoencoder (CVAE)

Trong project này, chúng ta sẽ dùng Conditional Variational Autoencoder (CVAE) nhằm mục đích tái tạo lại mẫu âm thanh nhưng với một thể loại khác. Tổng quan, kiến trúc mô hình sẽ gồm Encoder với 3 lớp Conv2d, đầu ra của mỗi lớp Conv2d có kích thước giảm gấp 2 lần; Latent space gồm 2 lớp Linear cho mu (giá trị trung bình) và logvar (log của phương sai); Decoder sẽ tương tự như Encoder nhưng có số chiều và kích thước ngược lại. Ngoài ra, ta dùng thêm Linear cho Decode (`decoder_input`) để chuyển về shape phù hợp với đầu vào. Cuối cùng, hàm Sigmoid để đầu ra có giá trị từ 0 tới 1 (giống các giá trị của đầu vào đã chuẩn hoá).



Hình 51.13: Minh họa kiến trúc CVAE với dữ liệu đầu vào là spectrogram và genres.

```

1 class CVAE(nn.Module):
2 def __init__(self, d_model, latent_dim, n_frames, n_mels, n_genres):
3 super(CVAE, self).__init__()
4 self.d_model = d_model
5 self.latent_dim = latent_dim
6 self.n_frames = int(np.ceil(n_frames / 2**3))
7 self.n_mels = int(np.ceil(n_mels / 2**3))
8 self.n_genres = n_genres
9 print(self.n_frames, self.n_mels)
10
11 # Encoder
12 self.encoder = nn.Sequential(
13 nn.Conv2d(1 + self.n_genres, d_model, kernel_size=3, stride=2,
14 padding=1),
15 nn.BatchNorm2d(d_model),
16 nn.SiLU(),
17 nn.Dropout2d(0.05),

```

```
17 nn.Conv2d(d_model, d_model * 2, kernel_size=3, stride=2,
18 padding=1),
19 nn.BatchNorm2d(d_model * 2),
20 nn.SiLU(),
21 nn.Dropout2d(0.1),
22
23 nn.Conv2d(d_model * 2, d_model * 4, kernel_size=3, stride=2,
24 padding=1),
25 nn.BatchNorm2d(d_model * 4),
26 nn.SiLU(),
27 nn.Dropout2d(0.15),
28
29 nn.AdaptiveAvgPool2d((1, 1)), # [B, 4*d, 1, 1]
30 nn.Flatten()
31)
32
33 # Latent space
34 self.fc_mu = nn.Linear(d_model * 4, latent_dim)
35 self.fc_logvar = nn.Linear(d_model * 4, latent_dim)
36
37 # Decoder
38 self.decoder_input = nn.Linear(latent_dim + self.n_genres, d_model
39 * 4 * self.n_frames * self.n_mels)
40 self.decoder = nn.Sequential(
41 nn.ConvTranspose2d(d_model * 4, d_model * 2, kernel_size=3,
42 stride=2, padding=1, output_padding=(1, 0)),
43 nn.BatchNorm2d(d_model * 2),
44 nn.SiLU(),
45 nn.Dropout2d(0.1),
46
47 nn.ConvTranspose2d(d_model * 2, d_model, kernel_size=3, stride
48 =2, padding=1, output_padding=(1, 0))
49 ,
50 nn.BatchNorm2d(d_model),
51 nn.SiLU(),
52 nn.Dropout2d(0.05),
53
54 nn.ConvTranspose2d(d_model, 1, kernel_size=3, stride=2,
55 padding=1, output_padding=1),
56 nn.Sigmoid()
57)
58
59 def reparameterize(self, mu, logvar):
```

```

54 std = torch.exp(0.5 * logvar)
55 eps = torch.randn_like(std)
56 return mu + eps * std
57
58 def forward(self, x, genres_input):
59 ori_genres_embed = genres_input.view(genres_input.size(0), -1)
60 genres_embed = ori_genres_embed.unsqueeze(-1).unsqueeze(-1)
61 genres_embed = genres_embed.expand(-1, -1, x.size(2), x.size(3))
62 x_genres = torch.cat((x, genres_embed), dim=1)
63
64 h = x_genres
65 shortcuts = []
66 for block in self.encoder:
67 h = block(h)
68 if isinstance(block, nn.SiLU):
69 shortcuts.append(h)
70
71 mu = self.fc_mu(h)
72 logvar = self.fc_logvar(h)
73
74 z = self.reparameterize(mu, logvar)
75 z_genres = torch.cat((z, ori_genres_embed), dim=1)
76
77 h_dec = self.decoder_input(z_genres)
78 h_dec = h_dec.view(-1, self.d_model * 4, self.n_frames, self.
79 n_mels)
80
81 for block in self.decoder:
82 if isinstance(block, nn.ConvTranspose2d) and shortcuts:
83 shortcut = shortcuts.pop()
84 h_dec = h_dec + shortcut
85 h_dec = block(h_dec)
86
87 recon = h_dec[:, :, :x.size(2), :x.size(3)]
88 return recon, mu, logvar

```

## Huấn luyện mô hình

Vì PyTorch không hỗ trợ trực tiếp hàm loss có sẵn cho VAE, chúng ta cần phải tự định nghĩa hàm loss này, đó là sự kết hợp giữa Reconstruction loss và KL-Divergence loss. Code triển khai như sau:

```

1 def loss_function(recon_x, x, mu, logvar):
2 recon_loss = nn.functional.mse_loss(recon_x, x, reduction="sum")

```

```

3 KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
4 return recon_loss + KLD

```

Khi đã có hàm loss, ta tiếp hành xây dựng một hàm huấn luyện mô hình cơ bản như sau:

```

1 def train_vae(model, dataloader, optimizer, scheduler, num_epochs,
 verbose_interval=50):
2 model.train()
3 losses = []
4 for epoch in tqdm(range(num_epochs), desc="Training", unit="epoch"):
5 train_loss = 0
6 for batch_idx, (data, genres_input, ori_data) in enumerate(
7 dataloader):
8 data = data.to(device)
9 genres_input = genres_input.to(device)
10 optimizer.zero_grad()
11
12 recon, mu, logvar = model(data, genres_input)
13 loss = loss_function(recon, data, mu, logvar)
14 loss.backward()
15 train_loss += loss.item()
16 optimizer.step()
17
18 scheduler.step()
19 avg_loss = train_loss / len(dataloader.dataset)
20 losses.append(avg_loss)
21 print(f"Epoch {epoch}/{num_epochs}, Loss: {avg_loss:.4f}, Lr: {scheduler.get_last_lr()[0]}")
22 if epoch == 0 or (epoch + 1) % verbose_interval == 0:
23 data = data[0].detach().cpu()
24 recon_img = recon[0].detach().cpu()
25 show_spectrogram(data, title="Original Spectrogram")
26 show_spectrogram(recon_img, title="Reconstructed Spectrogram")
27 return mu, logvar, losses

```

Cuối cùng, ta khai báo giá trị cho một vài tham số liên quan đến mô hình và việc huấn luyện để tiến hành thực hiện lời gọi hàm:

```

1 d_model = 64
2 latent_dim = 128
3 lr = 2e-4
4 num_epochs = 100
5 step_size = num_epochs//2
6 verbose_interval = num_epochs//10

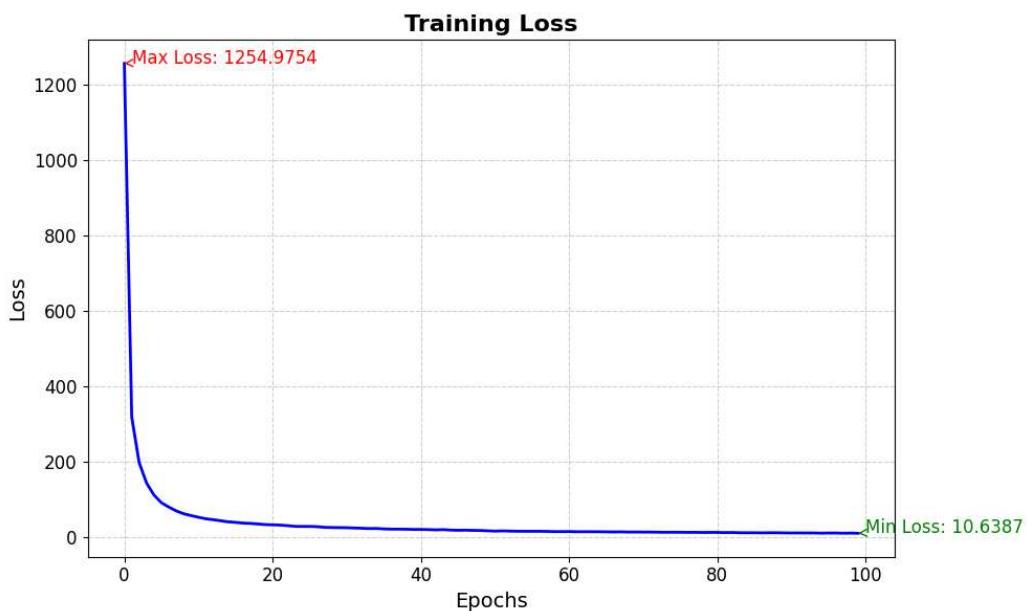
```

```

7 gamma = 0.5
8
9 model = CVAE(d_model, latent_dim, n_mels, frame, max_genres).to(device)
10 optimizer = optim.AdamW(model.parameters(), lr=lr)
11 scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=step_size,
12 gamma=gamma)
13 print(f"Total number of parameters: {sum(p.numel() for p in model.
14 parameters())}")
14 mu, logvar, losses = train_vae(model, trainloader, optimizer, scheduler,
15 num_epochs, verbose_interval=
16 verbose_interval)

```

Khi quá trình huấn luyện kết thúc, ta có được một mô hình với kết quả huấn luyện được trực quan hóa giá trị loss như sau:



Hình 51.14: Trực quan hóa giá trị loss xuyên suốt quá trình huấn luyện mô hình.

### Inference

Để sử dụng mô hình đã huấn luyện tạo ra dữ liệu âm thanh mới, ta triển khai hàm thực hiện inference với đầu vào là một mẫu dữ liệu âm thanh và

một danh sách các thể loại nhạc mà ta muốn chuyển đổi dữ liệu gốc thành. Trong trường hợp này, ta triển khai hàm với dữ liệu lấy từ testloader như sau:

```

1 def generate(model, dataloader, genres_list, num_samples=5, diff_level=1):
2 model.eval()
3 with torch.no_grad():
4 data, old_genres_input, ori_data = next(iter(dataloader))
5 data = data.to(device)
6
7 genres_tokens = tokenize(genres_list)
8 genres_input = onehot_encode(genres_tokens, model.n_genres)
9 genres_input = torch.tensor(genres_input, dtype=torch.long).
10 unsqueeze(0)
11 genres_input = genres_input.repeat(old_genres_input.shape[0], 1)
12 genres_input = genres_input.to(device)
13
14 recon, mu, logvar = model(data, genres_input)
15 ori_audios = []
16 recon_audios = []
17 for i in range(num_samples):
18 old_genres_list = detokenize_to_list(onehot_decode(
19 old_genres_input[i].squeeze().tolist()
20))
21 show_spectrogram(data[i], title="Original Spectrogram with
22 Genres: " + ", ".join(old_genres_list
23))
24 show_spectrogram(recon[i], title="Reconstructed Spectrogram
25 with Genres: " + ", ".join(
26 genres_list))
27
28 diff_spectrogram = torch.abs(data[i] - recon[i]) * diff_level
29 show_spectrogram(diff_spectrogram, title=f"Difference
30 Spectrogram (|Original -
31 Reconstructed|) * {diff_level}")
32 print("Loss: ", loss_function(recon[i], data[i], mu, logvar).
33 item())
34
35 spec_denorm = denormalize_melspec(recon[i].cpu().numpy().
36 squeeze(), ori_data[i].cpu().numpy().squeeze())
37 audio_reconstructed = melspec_to_audio(spec_denorm, sr=
38 sample_rate)
39 ori_audio = melspec_to_audio(ori_data[i].cpu().numpy().squeeze()
40 (), sr=sample_rate)

```

```
28 recon_audios.append(audio_reconstructed)
29 ori_audios.append(ori_audio)
30
31 display_audio_files(ori_audio, sample_rate, title=""
32 Reconstructed Audio with Genres: " +
33 ", ".join(old_genres_list))
34 display_audio_files(audio_reconstructed, sample_rate, title=""
35 Reconstructed Audio with Genres: " +
36 ", ".join(genres_list))
37
38 if num_samples > 1:
39 print("-"*100, "Connect all audio", "-"*100)
40 recon_ori_audios = np.concatenate(ori_audios)
41 display_audio_files(recon_ori_audios, sample_rate, title="
42 Connect all original audio")
43 recon_audios = np.concatenate(recon_audios)
44 display_audio_files(recon_audios, sample_rate, title="Connect
45 all reconstructed audio")
```

## Triển khai streamlit

Với mô hình đã huấn luyện được, chúng ta có thể triển khai thành ứng dụng web demo để sử dụng. Hình ảnh dưới đây là một ví dụ xây dựng thông qua streamlit, các bạn có thể truy cập trải nghiệm tại mục 60.4



## New Genres Audio Reconstruction

Tải lên 1 audio (chỉ xử lý 15s đầu tiên)



Drag and drop file here

Limit 200MB per file • WAV, MP3

Browse files

Hoặc chọn 1 audio mẫu dưới đây:

classical.00000.wav

▶ 0:00 / 0:30



Chọn thể loại

Soundtrack ×

Jazz ×

Christmas ×

New Age ×

Chill-out ×



Xử lý Âm Thanh

Kết quả:

▶ 0:00 / 0:14



Hình 51.15: Minh họa web demo cho chương trình tạo sinh âm thanh dựa trên âm thanh đầu vào và các thể loại nhạc.

### 51.3 Câu hỏi trắc nghiệm

1. Phát biểu nào sau đây là đúng về Variational Autoencoder (VAE)?
  - a) VAE là một mô hình phân loại.
  - b) VAE là một mô hình sinh dữ liệu.
  - c) VAE là một mô hình dự đoán một giá trị liên tục.
  - d) VAE là một mô hình clustering.
2. Reparameterization trick trong VAE dùng để:
  - a) Tách riêng encoder và decoder.
  - b) Cho phép gradient lan truyền qua quá trình sampling.
  - c) Tăng nhiễu cho đa dạng đầu ra.
  - d) Chuyển dữ liệu liên tục thành rời rạc.
3. Hàm loss của VAE truyền thống bao gồm:
  - a) Chỉ MSE.
  - b) Chỉ KL Divergence.
  - c) Cả MSE và KL Divergence.
  - d) MSE và Cross Entropy.
4. MSE trong loss của VAE đo lường:
  - a) Độ lệch trung bình dữ liệu.
  - b) Lỗi tái tạo (reconstruction error).
  - c) Sai số dự đoán.
  - d) Độ nhiễu trong tín hiệu.
5. KL Divergence trong loss của VAE đo lường:
  - a) Khoảng cách giữa latent distribution và normal distribution.
  - b) Sai số tái tạo.
  - c) Độ phân tán dữ liệu.

- d) Sự chênh lệch đầu vào - đầu ra.

6. Encoder trong VAE có chức năng:

- a) Sinh dữ liệu mới từ latent space.
- b) Nén đầu vào thành latent space.
- c) Tăng nhiễu dữ liệu.
- d) Tạo đầu ra từ latent.

7. Decoder trong VAE có nhiệm vụ:

- a) Lấy mẫu từ latent và tái tạo đầu ra.
- b) Nén dữ liệu vào latent.
- c) Tính toán hàm loss.
- d) Huấn luyện mô hình.

8. Thư viện **librosa** được dùng để:

```
1 import librosa
2 audio, sr = librosa.load('example.mp3', sr=22050)
3 mel_spec = librosa.feature.melspectrogram(y=audio, sr=sr)
```

- a) Chuyển audio thành văn bản.
- b) Tính toán đặc trưng âm thanh.
- c) Phục hồi ảnh từ audio.
- d) Trực quan hóa tín hiệu.

9. Hàm **librosa.feature.melspectrogram** dùng để:

```
1 mel_spec = librosa.feature.melspectrogram(y=audio, sr=sr, n_mels=128
)
```

- a) Tính toán mel-spectrogram từ audio.
- b) Chuyển đổi mel-spectrogram thành audio.
- c) Hiển thị biểu đồ năng lượng.
- d) Chuẩn hóa dữ liệu.

10. Tại sao khi code lại dùng `logvar` thay vì dùng giá trị variance trực tiếp?

```
1 KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
```

- a) Giúp ổn định số học, gradient hiệu quả.
- b) Giảm tham số mô hình, tăng tốc huấn luyện.
- c) Gây khó khăn trong quá trình lan truyền ngược (backpropagation),  
dễ tràn số.
- d) Tất cả các đáp án trên.

## 51.4 Phụ lục

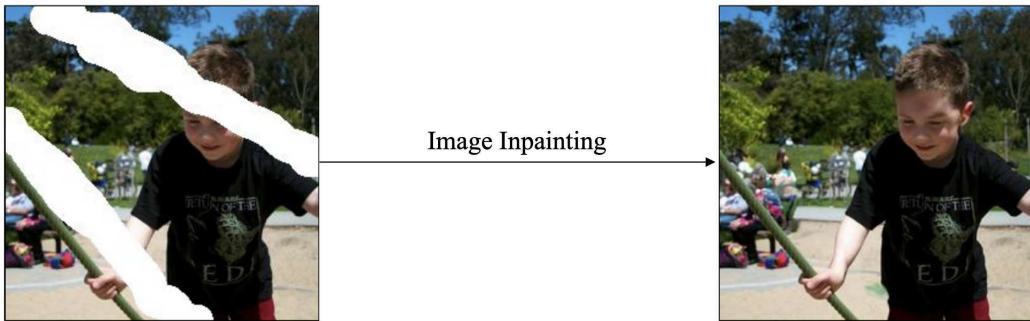
1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
2. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
3. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào link).
4. **Demo:** Web demo và mã nguồn của ứng dụng có thể được truy cập tại [đây](#).
5. **Rubric:**

Mục	Kiến Thức	Đánh Giá
I.	<ul style="list-style-type: none"> <li>- Kiến thức về thu thập dữ liệu thông qua kỹ thuật crawling data.</li> <li>- Kiến thức về thư viện Selenium.</li> <li>- Kiến thức cơ bản về HTML và cách xây dựng logic thu thập dữ liệu thông qua cấu trúc HTML của một trang web.</li> </ul>	<ul style="list-style-type: none"> <li>- Nắm được cách sử dụng thư viện Selenium để lấy dữ liệu từ một trang web.</li> <li>- Có khả năng phân tích cơ bản cấu trúc HTML để xây dựng từng bước xử lý để bốc tách được dữ liệu cần thu thập từ trang web.</li> </ul>
II.	<ul style="list-style-type: none"> <li>- Kiến trúc Variational Autoencoder (VAE) và Conditional VAE (CVAE).</li> <li>- Các kỹ thuật xử lý tín hiệu âm thanh cơ bản.</li> <li>- Cách xây dựng mô hình VAE sử dụng thư viện PyTorch.</li> <li>- Cách chuẩn bị và tổ chức dữ liệu huấn luyện âm thanh và văn bản cho VAE để có khả năng tái tạo âm nhạc.</li> </ul>	<ul style="list-style-type: none"> <li>- Nắm được các lý thuyết cơ bản về kiến trúc VAE và CVAE.</li> <li>- Khả năng lập trình PyTorch để xây dựng mô hình VAE và CVAE.</li> <li>- Cách ứng dụng CVAE trên dữ liệu âm thanh và văn bản để triển khai mô hình tái tạo ảnh spectrogram của âm thanh.</li> </ul>

# Chương 52

## Diffusion Models

### 52.1 Giới thiệu



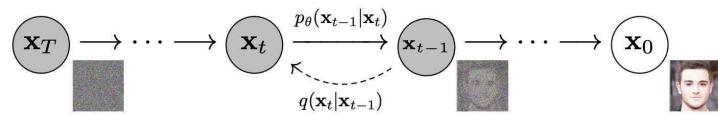
Hình 52.1: Ví dụ minh họa Image Inpainting sử dụng mô hình Denoising Diffusion Probabilistic Model.

**Diffusion Models** là một trong những mô hình sinh ngày càng được ứng dụng rộng rãi cho nhiều ứng dụng khác nhau như: Image Inpainting (Chỉnh sửa hình ảnh khuyết thiêng), Image Colorization (Tô màu hình ảnh),... Diffusion Models có nhiệm vụ tạo ra một phân phối cho dữ liệu đầu vào và xấp xỉ phân phối của dữ liệu được sinh ra với phân phối của dữ liệu gốc, từ đó giúp mô hình có thể sinh ra hình ảnh mới.

Trong phần này chúng ta sẽ tìm hiểu về ứng dụng Image Inpainting. Và sử dụng mô hình Denoising Diffusion Probabilistic Model để huấn luyện mô hình hoàn thiện các hình ảnh bị khuyết thiêng.

Diffusion Models bao gồm 2 quá trình: Forward Diffusion Process và Reverse Diffusion Process được mô tả như sau:

- (a) Forward Diffusion Process (FDP): Từ một điểm dữ liệu đầu vào  $x_0$  thuộc một phân phối biết trước  $x_0 \sim q(x)$ , FDP sẽ thêm từ từ lần lượt theo thời gian một lượng nhỏ nhiễu được lấy mẫu từ phân phối Gauss  $\epsilon \sim \mathcal{N}(\mu, \sigma^2)$  tạo ra các mẫu chứa nhiễu  $x_1, x_2, \dots, x_T$ , với  $T$  (steps) số bước thêm nhiễu vào.



Hình 52.2: Minh họa quá trình hoạt động của Diffusion Models.

- (b) Reverse Diffusion Process (RDP): Tại thời điểm  $x_T$  là mẫu chứa nhiễu, RDF sẽ tiến hành huấn luyện mô hình để khử nhiễu, khôi phục lại dữ liệu hình ảnh đầu vào. Mô hình được sử dụng trong quá trình decode khử nhiễu là UNet kết hợp với cơ chế Attention.

## 52.2 Image Inpainting using Denoising Diffusion Probabilistic Model

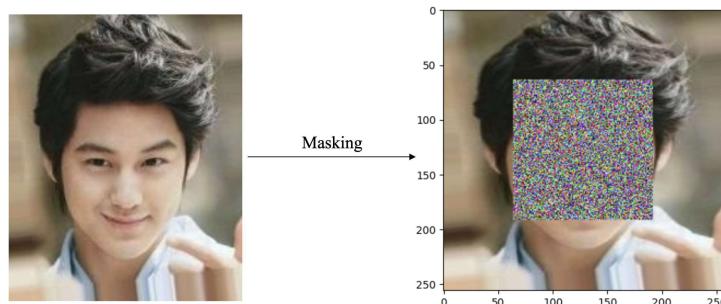
Trong phần này chúng ta sẽ huấn luyện mô hình Denoising Diffusion Probabilistic Model (DDP) ([paper](#)) để giải quyết bài toán Image Inpainting dựa vào bộ dữ liệu [CelebA](#)

Nội dung thực nghiệm bao gồm 5 phần:

- (a) Data Preparing: Chuẩn bị dữ liệu CelebA cho huấn luyện mô hình
- (b) Model: Xây dựng mô hình DDP
- (c) Loss, Metric: Xây dựng hàm mất mát và độ đo đánh giá
- (d) Trainer: Xây dựng các hàm để huấn luyện mô hình
- (e) Inference: Minh họa kết quả đạt được sau khi huấn luyện mô hình

### 1. Data Preparing

Bộ dữ liệu [CelebA](#) bao gồm hơn 200,000 ảnh khuôn mặt, vì vậy để huấn luyện mô hình Image Inpainting, trong phần này chúng ta sẽ tạo ra các ảnh mask. Có nhiều cách khác nhau để tạo ra ảnh mask từ ảnh gốc như: mask các điểm ảnh trung tâm, mask các điểm ảnh ở các góc hoặc mask các điểm ảnh ngẫu nhiên trên bức ảnh gốc. Để đơn giản, chúng ta sẽ chọn mask tại các vị trí điểm ảnh trung tâm của bức ảnh dưới dạng hình chữ nhật. Bên cạnh đó, chúng ta cũng thực hiện một số bước tiền xử lý như thay đổi kích thước các ảnh thành 256x256, chuẩn hóa các ảnh đầu vào.



Hình 52.3: Minh họa quá trình tạo ra ảnh mask từ ảnh gốc.

```
1 # Define mask
2 import os
3 import numpy as np
4
5 file_names = os.listdir('./img_align_celeba')
6 img_paths = ['./img_align_celeba/' + file_name for file_name
7 in file_names]
8
9 # train: valid split
10 num_train = 150000
11 train_imgpaths = img_paths[: num_train]
12 val_imgpaths = img_paths[num_train :]
13
14 # generate mask image
15 def bbox2mask(img_shape, bbox, dtype='uint8'):
16 """
17 Generate mask in ndarray from bbox.
18 bbox (tuple[int]): Configuration tuple, (top, left,
19 height, width)
20 """
21 height, width = img_shape[:2]
22 mask = np.zeros((height, width, 1), dtype=dtype)
23 mask[bbox[0]:bbox[0] + bbox[2], bbox[1]:bbox[1] + bbox
24 [3], :] = 1
25
26 return mask
27
28
29 import torch
30 from PIL import Image
31 from torchvision import transforms
32 from torch.utils.data import Dataset
33
34 # build dataset
35 class InpaintingDataset(Dataset):
36 def __init__(self, img_paths, mask_mode, image_size=[256,
37 256]):
38 self.img_paths = img_paths
39 self.tfs = transforms.Compose([
40 transforms.Resize((image_size[0], image_size
41 [1])),
42 transforms.ToTensor(),
43 transforms.Normalize(mean=[0.5, 0.5, 0.5],
44 std=[0.5, 0.5, 0.5])
45])
46 self.mask_mode = mask_mode
47 self.image_size = image_size
```

```

18 def __getitem__(self, index):
19 img_path = self.img_paths[index]
20 img = Image.open(img_path).convert('RGB')
21 img = self.tfs(img)
22 mask = self.get_mask()
23 cond_image = img*(1. - mask) + mask*torch.randn_like(
24 img)
25 mask_img = img*(1. - mask) + mask
26 return {
27 'gt_image': img,
28 'cond_image': cond_image,
29 'mask_image': mask_img,
30 'mask': mask,
31 'path': img_path
32 }
33
34 def __len__(self):
35 return len(self.img_paths)
36
37 def get_mask(self):
38 if self.mask_mode == 'center':
39 h, w = self.image_size
40 mask = bbox2mask(self.image_size, (h//4, w//4, h
41 //2, w//2))
42 else:
43 raise NotImplementedError(
44 f'Mask mode {self.mask_mode} has not been
45 implemented.')
46 return torch.from_numpy(mask).permute(2,0,1)

```

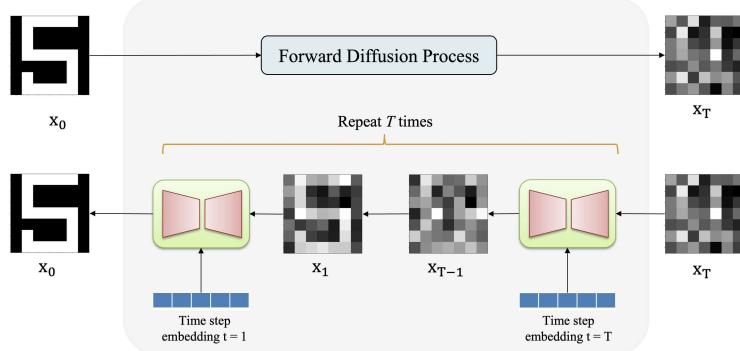
## 2. Model

Trong phần này chúng ta sẽ xây dựng mô hình cho quá trình RDP. Mô hình UNet sẽ được sử dụng làm mô hình khử nhiễu qua mỗi bước thời gian. Quá trình FDP sẽ thêm lần lượt theo bước thời gian  $T$  nhiễu Gauss vào ảnh đầu vào, vì vậy ở bước RDP chúng ta sẽ bổ sung thêm không gian embedding của bước thời gian vào mô hình UNet dựa vào hàm Sinusoidal Positional Embedding.

Phần xây dựng mô hình sẽ bao gồm 2 phần. Phần 1, chúng ta sẽ xây dựng mô hình UNet cơ bản chỉ bao gồm kiến trúc các Block của mô hình UNet gốc kết hợp với embedding của bước thời gian. Ở phần 2, chúng ta sẽ sử dụng mô hình UNet kết hợp thêm cơ chế Attention và Adaptive Group Normalization giúp cải tiến mô hình được giới thiệu trong bài [Diffusion Models Beat GAN](#)

## on Image Synthesis

### 2.1. Basic UNet Model



Hình 52.4: Minh họa quá trình RDP sử dụng kết hợp embedding bước thời gian vào mô hình UNet.

```

1 from torch import nn
2 import math
3 class Block(nn.Module):
4 def __init__(self, in_ch, out_ch, time_emb_dim, up=False)
5 :
6 super().__init__()
7 self.time_mlp = nn.Linear(time_emb_dim, out_ch)
8 if up:
9 self.conv1 = nn.Conv2d(2*in_ch, out_ch, 3,
10 padding=1)
11 self.transform = nn.ConvTranspose2d(out_ch,
12 out_ch, 4, 2, 1)
13 else:
14 self.conv1 = nn.Conv2d(in_ch, out_ch, 3, padding
15 =1)
16 self.transform = nn.Conv2d(out_ch, out_ch, 4, 2,
17 1)
18 self.conv2 = nn.Conv2d(out_ch, out_ch, 3, padding=1)
19 self.bnrm1 = nn.BatchNorm2d(out_ch)
20 self.bnrm2 = nn.BatchNorm2d(out_ch)
21 self.relu = nn.ReLU()
22 def forward(self, x, t,):
23 h = self.bnrm1(self.relu(self.conv1(x)))

```

```
19 time_emb = self.relu(self.time_mlp(t)) # Time
20 embedding
21 time_emb = time_emb[(...,) + (None,) * 2]
22 h = h + time_emb # Add time channel
23 h = self.bn2d(self.relu(self.conv2(h)))
24 return self.transform(h) # Down or Upsample
25
26
27
28
29
30
31
32
1 class SinusoidalPositionEmbeddings(nn.Module):
2 def __init__(self, dim):
3 super().__init__()
4 self.dim = dim
5 def forward(self, time):
6 device = time.device
7 half_dim = self.dim // 2
8 embeddings = math.log(10000) / (half_dim - 1)
9 embeddings = torch.exp(torch.arange(half_dim, device=
device) * -embeddings)
10 embeddings = time[:, None] * embeddings[None, :]
11 embeddings = torch.cat((embeddings.sin(), embeddings.
cos()), dim=-1)
12 return embeddings
13
14 class SimpleUnet(nn.Module):
15 def __init__(self):
16 super().__init__()
17 image_channels = 3
18 down_channels = (64, 128, 256, 512, 1024)
19 up_channels = (1024, 512, 256, 128, 64)
20 out_dim = 3
21 time_emb_dim = 32
22 self.time_mlp = nn.Sequential(
23 SinusoidalPositionEmbeddings(time_emb_dim),
24 nn.Linear(time_emb_dim, time_emb_dim), nn.ReLU()
25)
26 self.conv0 = nn.Conv2d(image_channels, down_channels
[0], 3, padding=1)
27 self.downs = nn.ModuleList([Block(down_channels[i],
down_channels[i+1], time_emb_dim) for i in range(len(
down_channels)-1)])
28 self.ups = nn.ModuleList([Block(up_channels[i],
up_channels[i+1], time_emb_dim, up=True) for i in range(
len(up_channels)-1)])
29 self.output = nn.Conv2d(up_channels[-1], out_dim, 1)
30 def forward(self, x, timestep):
31 t = self.time_mlp(timestep)
32 x = self.conv0(x)
```

```

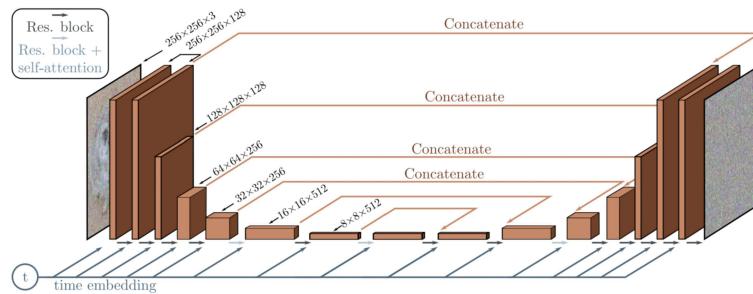
33 residual_inputs = []
34 for down in self.downs:
35 x = down(x, t)
36 residual_inputs.append(x)
37 for up in self.ups:
38 residual_x = residual_inputs.pop()
39 x = torch.cat((x, residual_x), dim=1)
40 x = up(x, t)
41 return self.output(x)

```

## 2.2. Improved UNet Model

Trong phần này chúng ta sẽ xây dựng mô hình UNet kết hợp với cơ chế Attention và Adaptive Group Normalization. Có thể tải về mã nguồn model [tại đây](#)

Kết trúc mô hình UNet sử dụng kết hợp với cơ chế Attention được mô tả như sau:



Hình 52.5: Mô hình UNet kết hợp cơ chế Attention trong DDP.

### 2.3. Gaussian Diffusion Model

Trong phần này chúng ta định nghĩa mô hình Diffusion Model đầy đủ với 2 bước: FDP ước lượng bước thời gian  $\beta$  dựa vào hàm linear, các giá trị xác suất để tính  $q(x_t|x_0)$  và  $q(x_{t-1}|x_t, x_0)$ , thuật toán huấn luyện và lấy mẫu.

- Xây dựng hàm tạo các tham số

```
1 from tqdm import tqdm
2 from functools import partial
3
4 def make_beta_schedule(schedule, n_timestep, linear_start=1e
-5, linear_end=1e-2):
5 if schedule == 'linear':
6 betas = np.linspace(
7 linear_start, linear_end, n_timestep, dtype=np.
float64
8)
9 else:
10 raise NotImplementedError(schedule)
11 return betas
12
13 def get_index_from_list(vals, t, x_shape=(1,1,1,1)):
14 """
15 Returns a specific index t of a passed list of values
16 vals
17 while considering the batch dimension.
18 """
19 batch_size, *_ = t.shape
20 out = vals.gather(-1, t)
21 return out.reshape(batch_size, *((1,) * (len(x_shape) -
1))).to(device)
```

- Xây dựng hàm xác định các giá trị xác suất cho quá trình forward và reverse process.

```

1 class InpaintingGaussianDiffusion(nn.Module):
2 def __init__(self, unet_config, beta_schedule, **kwargs):
3 super(InpaintingGaussianDiffusion, self).__init__(**kwargs)
4 self.denoise_fn = UNet(**unet_config)
5 self.beta_schedule = beta_schedule
6 def set_new_noise_schedule(self, device):
7 to_torch = partial(torch.tensor, dtype=torch.float32,
device=device)
8 betas = make_beta_schedule(**self.beta_schedule)
9 alphas = 1. - betas
10 timesteps, = betas.shape
11 self.num_timesteps = int(timesteps)
12 gammas = np.cumprod(alphas, axis=0) # alphas_cumprod
13 gammas_prev = np.append(1., gammas[:-1])
14 self.register_buffer("gammas", to_torch(gammas)) # q(
x_t | x_{t-1})
15 self.register_buffer("sqrt_recip_gammas", to_torch(np
.sqrt(1. / gammas)))
16 self.register_buffer("sqrt_recipm1_gammas", to_torch(
np.sqrt(1. / gammas - 1)))
17 posterior_variance = betas * (1. - gammas_prev) / (1.
- gammas) # q(x_{t-1} | x_t, x_0)
18 self.register_buffer("posterior_log_variance_clipped",
to_torch(np.log(np.maximum(posterior_variance, 1e-20))))
19 self.register_buffer("posterior_mean_coef1", to_torch(
betas * np.sqrt(gammas_prev) / (1. - gammas)))
20 self.register_buffer("posterior_mean_coef2", to_torch(
((1. - gammas_prev) * np.sqrt(alphas) / (1. - gammas)))
21
22 def set_loss(self, loss_fn):
23 self.loss_fn = loss_fn
24 def predict_start_from_noise(self, y_t, t, noise):
25 return (
26 get_index_from_list(self.sqrt_recip_gammas, t,
y_t.shape) * y_t -
27 get_index_from_list(self.sqrt_recipm1_gammas, t,
y_t.shape) * noise
28)
29 def q_posterior(self, y_0_hat, y_t, t):
30 # q(x_{t-1} | x_t, x_0)
31 posterior_mean = (
32 get_index_from_list(self.posterior_mean_coef1, t,

```

```

33 y_t.shape) * y_0_hat +
34 get_index_from_list(self.posterior_mean_coef2, t,
35 y_t.shape) * y_t
36)
37 posterior_log_variance_clipped = get_index_from_list(
38 self.posterior_log_variance_clipped, t, y_t.shape
39)
40 return posterior_mean, posterior_log_variance_clipped

1 def p_mean_variance(self, y_t, t, clip_denoised: bool,
2 y_cond=None):
3 noise_level = get_index_from_list(self.gammas, t,
4 x_shape=(1, 1)).to(y_t.device)
5 y_0_hat = self.predict_start_from_noise(
6 y_t, t=t, noise=self.denoise_fn(torch.cat([
7 y_cond, y_t], dim=1), noise_level))
8 if clip_denoised:
9 y_0_hat.clamp_(-1., 1.)
10 model_mean, posterior_log_variance = self.q_posterior(
11 y_0_hat=y_0_hat, y_t=y_t, t=t)
12 return model_mean, posterior_log_variance
13
14 def q_sample(self, y_0, sample_gammas, noise=None):
15 noise = noise if noise is not None else torch.
16 randn_like(y_0)
17 return (
18 sample_gammas.sqrt() * y_0 +
19 (1 - sample_gammas).sqrt() * noise
20)
21
22 def forward(self, y_0, y_cond=None, mask=None, noise=None
23):
24 # sampling from p(gammas)
25 b, *_ = y_0.shape
26 t = torch.randint(1, self.num_timesteps, (b,), device
27 =y_0.device).long()
28
29 gamma_t1 = get_index_from_list(self.gammas, t-1,
30 x_shape=(1, 1))
31 sqrt_gamma_t2 = get_index_from_list(self.gammas, t,
32 x_shape=(1, 1))
33 sample_gammas = (sqrt_gamma_t2-gamma_t1) * torch.rand(
34 ((b, 1), device=y_0.device)) + gamma_t1
35 sample_gammas = sample_gammas.view(b, -1)
36
37

```

```

28 noise = noise if noise is not None else torch.
29 randn_like(y_0)
30 y_noisy = self.q_sample(
31 y_0=y_0, sample_gammas=sample_gammas.view(-1, 1,
32 1, 1), noise=noise)
33
34 if mask is not None:
35 noise_hat = self.denoise_fn(torch.cat([y_cond,
36 y_noisy*mask+(1.-mask)*y_0], dim=1), sample_gammas)
37 loss = self.loss_fn(mask*noise, mask*noise_hat)
38
39 else:
40 noise_hat = self.denoise_fn(torch.cat([y_cond,
41 y_noisy], dim=1), sample_gammas)
42 loss = self.loss_fn(noise, noise_hat)
43
44 return loss
45
46 @torch.no_grad()
47 def p_sample(self, y_t, t, clip_denoised=True, y_cond=
48 None):
49 model_mean, model_log_variance = self.p_mean_variance
50 (
51 y_t=y_t, t=t, clip_denoised=clip_denoised, y_cond=
52 y_cond)
53 noise = torch.randn_like(y_t) if any(t>0) else torch.
54 zeros_like(y_t)
55 return model_mean + noise * (0.5 * model_log_variance
56).exp()
57 @torch.no_grad()
58 def restoration(self, y_cond, y_t=None, y_0=None, mask=
59 None, sample_num=8):
60 b, *_ = y_cond.shape
61
62 sample_inter = (self.num_timesteps//sample_num)
63
64 y_t = y_t if y_t is not None else torch.randn_like(
65 y_cond)
66 ret_arr = y_t
67 for i in reversed(range(0, self.num_timesteps)):
68 t = torch.full((b,), i, device=y_cond.device,
69 dtype=torch.long)
70 y_t = self.p_sample(y_t, t, y_cond=y_cond)
71 if mask is not None:
72 y_t = y_0*(1.-mask) + mask*y_t
73 if i % sample_inter == 0:
74 ret_arr = torch.cat([ret_arr, y_t], dim=0)

```

```
22 return y_t, ret_arr
```

### 3. Loss. Metric

Trong phần này chúng ta xây dựng hàm mất mát và độ đo đánh giá mô hình

```
1 import torch.nn.functional as F
2
3 def mse_loss(output, target):
4 return F.mse_loss(output, target)
5
6
7 def mae(input, target):
8 with torch.no_grad():
9 loss = nn.L1Loss()
10 output = loss(input, target)
11 return output
```

### 4. Trainer

Trong phần này chúng ta xây dựng hàm huấn luyện mô hình.

```
1 import time
2 class Trainer():
3 def __init__(self, model, optimizers, train_loader,
4 val_loader, epochs, sample_num, device, save_model):
5 self.model = model.to(device)
6 self.optimizer = torch.optim.Adam(list(filter(lambda
7 p: p.requires_grad, self.model.parameters())), **
8 optimizers)
9 self.model.set_loss(mse_loss)
10 self.model.set_new_noise_schedule(device)
11 self.sample_num = sample_num
12 self.train_loader = train_loader
13 self.val_loader = val_loader
14 self.device = device
15 self.epochs = epochs
16 self.save_model = save_model + "/best_model.pth"
17 def train_step(self):
18 losses = []
19 for batch in tqdm(self.train_loader):
20 gt_image = batch['gt_image'].to(self.device)
21 cond_image = batch['cond_image'].to(self.device)
22 mask = batch['mask'].to(self.device)
23 mask_image = batch['mask_image'].to(self.device)
24 batch_size = len(batch['path'])
25 self.optimizer.zero_grad()
26 loss = self.model(gt_image, cond_image, mask=mask)
```

```
)
24 loss.backward()
25 losses.append(loss.item())
26 self.optimizer.step()
27 return sum(losses)/len(losses)
28 def val_step(self):
29 losses, metrics = [], []
30 with torch.no_grad():
31 for batch in tqdm(self.val_loader):
32 gt_image = batch['gt_image'].to(self.device)
33 cond_image = batch['cond_image'].to(self.
device)
34 mask = batch['mask'].to(self.device)
35 mask_image = batch['mask_image'].to(self.
device)
36 loss = self.model(gt_image, cond_image, mask=
mask)
37 output, visuals = self.model.restoration(
cond_image, y_t=cond_image, y_0=gt_image, mask=mask,
sample_num=self.sample_num)
38 mae_score = mae(gt_image, output)
39 losses.append(loss.item())
40 metrics.append(mae_score.item())
41 return sum(losses)/len(losses), sum(metrics)/len(
metrics)

1 def train(self):
2 best_mae = 100000
3 for epoch in range(self.epochs):
4 epoch_start_time = time.time()
5 train_loss = self.train_step()
6 val_loss, val_mae = self.val_step()
7 if val_mae < best_mae:
8 torch.save(self.model.state_dict(), self.
save_model)
9 # Print loss, acc end epoch
10 print("-" * 59)
11 print(
12 "| End of epoch {:3d} | Time: {:.5.2f}s | "
Train Loss {:.8.3f} "
13 "| Valid Loss {:.8.3f} | Valid MAE {:.8.3f} ".
format(
14 epoch+1, time.time() - epoch_start_time,
train_loss, val_loss, val_mae
15)
16)
```

```

17 print("-" * 59)
18 self.model.load_state_dict(torch.load(self.save_model
19))
20 epochs = 200 # 5
21 sample_num = 8
22 save_model = "./save_model"
23 optimizers = { "lr": 5e-5, "weight_decay": 0}
24 device = "cuda" if torch.cuda.is_available() else "cpu"
25
26 unet_config = {
27 "in_channel": 6, "out_channel": 3, "inner_channel": 64,
28 "channel_mults": [1, 2, 4, 8], "attn_res": [16], "
29 num_head_channels": 32, "res_blocks": 2, "dropout": 0.2, "
30 image_size": 256
31 }
32
33 beta_schedule = {
34 "schedule": "linear", "n_timestep": 20, "linear_start": 1
35 e-4, "linear_end": 0.09
36 }
37 inpainting_model = InpaintingGaussianDiffusion(unet_config,
38 beta_schedule)
39
40 trainer = Trainer(
41 inpainting_model, optimizers, train_loader, val_loader,
42 epochs, sample_num, device, save_model
43)

```

## 5. Inference

Sau quá trình huấn luyện, checkpoint của mô hình tốt nhất có thể tải về [tại đây](#) và sử dụng để sinh ảnh mới từ ảnh mask.

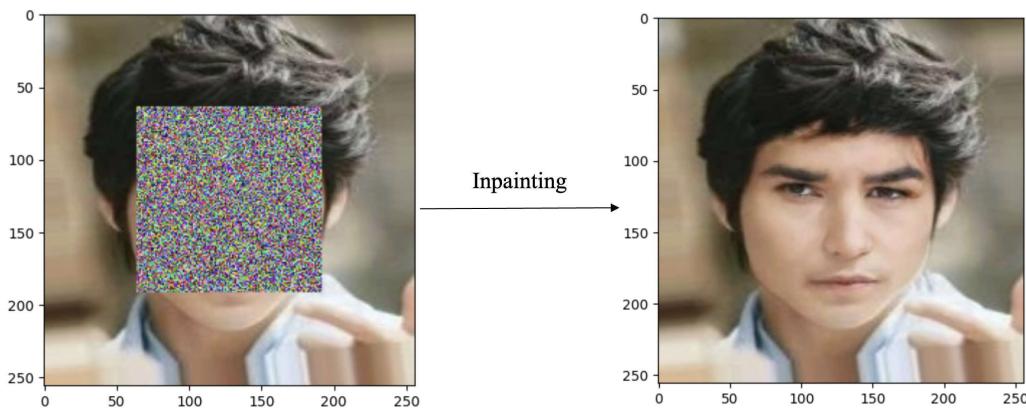
```

1 # load model
2 inpainting_model = InpaintingGaussianDiffusion(unet_config,
3 beta_schedule)
4 inpainting_model.set_new_noise_schedule(device)
5 load_state = torch.load('./save_model/best_model_200.pth')
6 inpainting_model.load_state_dict(load_state, strict=True)
7 inpainting_model.eval().to(device)
8
9 # test image
10 test_imgpath = img_paths[16]
11 test_dataset = InpaintingDataset([test_imgpath], mask_mode='
12 center')
13 test_sample = next(iter(test_dataset))

```

```
12
13 def inference(model, test_sample):
14 with torch.no_grad():
15 output, visuals = model.restoration(
16 test_sample['cond_image'].unsqueeze(0).to(device)
17 ,
18 y_t=test_sample['cond_image'].unsqueeze(0).to(
19 device),
20 y_0=test_sample['cond_image'].unsqueeze(0).to(
21 device),
22 mask=test_sample['mask'].unsqueeze(0).to(device)
23)
24 return output, visuals
25
26 output, visuals = inference(inpainting_model, test_sample)
27
28 # show result
29 def show_tensor_image(image, show=True):
30 reverse_transforms = transforms.Compose([
31 transforms.Lambda(lambda t: (t + 1) / 2),
32 transforms.Lambda(lambda t: t.permute(1, 2, 0)), # CHW to HWC
33 transforms.Lambda(lambda t: t * 255.),
34 transforms.Lambda(lambda t: t.numpy().astype(np.uint8)),
35 transforms.ToPILImage(),
36])
37
38 # Take first image of batch
39 if len(image.shape) == 4:
40 image = image[0, :, :, :]
```

Kết quả thử nghiệm



Hình 52.6: Kết quả thực nghiệm mô hình sau khi huấn luyện.

## 6. Deployment

Triển khai mô hình trên streamlit. Tham khảo về [code](#) và [demo](#).

**Image Inpainting using Denoising Diffusion Probabilistic Model**

**Model: DDPM - Repaint. Dataset: CelebA-HQ**

How would you like to give the input?

Upload Image File

Please upload an image

Drag and drop file here  
Limit 200MB per file • JPG, PNG, JPEG

Browse files

example.png 117.2KB

Hình 52.7: Triển khai ứng dụng trên Streamlit.

### 52.3 Câu hỏi trắc nghiệm

**Câu hỏi 93** Mục tiêu của Forward Diffusion Process trong mô hình Diffusion là gì?

- a) Thêm một lượng nhỏ nhiễu được lấy mẫu từ phân phối Gauss vào giá trị input  $x_0$  lần lượt theo bước nhảy  $T$  với lịch trình phương sai  $\beta_1, \dots, \beta_T$
- b) Khử nhiễu trong  $x_T$  để khôi phục giá trị đầu vào
- c) Cả 2 đáp án trên đều đúng
- d) Cả 2 đáp án trên đều sai

**Câu hỏi 94** Mục tiêu của Reverse Diffusion Process trong mô hình Diffusion là gì?

- a) Thêm một lượng nhỏ nhiễu được lấy mẫu từ phân phối Gauss vào giá trị input  $x_0$  lần lượt theo bước nhảy  $T$  với lịch trình phương sai  $\beta_1, \dots, \beta_T$
- b) Khử nhiễu trong  $x_T$  để khôi phục giá trị đầu vào
- c) Cả 2 đáp án trên đều đúng
- d) Cả 2 đáp án trên đều sai

**Câu hỏi 95** Dựa vào thực nghiệm trong phần 2, lịch trình phương sai  $\beta$  được tính dựa vào hàm nào sau đây?

- a) Linspace
- b) Sigmoid
- c) ReLU
- d) Tanh

**Câu hỏi 96** Dựa vào thực nghiệm trong phần 2, phương pháp mask nào sau đây được sử dụng?

- a) Mask các vị trí điểm ảnh trung tâm theo hình chữ nhật
- b) Mask các vị trí điểm ảnh ở góc trái trên theo hình chữ nhật
- c) Mask các vị trí điểm ảnh ở góc trái dưới theo hình chữ nhật
- d) Mask các vị trí điểm ảnh trung tâm theo hình tròn

**Câu hỏi 97** Dựa vào bài báo cáo nghiên cứu "Denoising Diffusion Probabilistic Model", công thức tính giá trị xác suất  $q(x_t|x_0)$  là?

- a)  $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$
- b)  $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \alpha_t}\epsilon$
- c)  $x_t = \sqrt{1 - \bar{\alpha}_t}x_0 + \sqrt{1 - \alpha_t}\epsilon$
- d)  $x_t = \sqrt{1 - \bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon$

**Câu hỏi 98** Dựa vào bài báo cáo nghiên cứu "Denoising Diffusion Probabilistic Model", công thức tính  $\tilde{\beta}_t$  trong  $q(x_{t-1}|x_t, x_0)$  là?

- a)  $\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}$
- b)  $\tilde{\beta}_t = \frac{1 - \bar{\alpha}_t}{1 - \bar{\alpha}_{t-1}}\beta_t$
- c)  $\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t$
- d)  $\tilde{\beta}_t = \frac{1}{1 - \bar{\alpha}_t}\beta_t$

**Câu hỏi 99** Dựa vào phần thực nghiệm 2, phương pháp embedding cho bước thời gian được sử dụng khi xây dựng mô hình Basic UNet là?

- a) ALiBi
- b) Rotary Embedding
- c) Conditional Positional Embedding
- d) Sinusoidal Positional Embedding

**Câu hỏi 100** Dựa vào phần thực nghiệm 2, bộ dữ liệu huấn luyện mô hình nào được sử dụng?

- a) CelebA
- b) MNIST
- c) CIFAR10
- d) CIFAR100

**Câu hỏi 101** Dựa vào phần thực nghiệm 2, hàm loss nào sau đây không được sử dụng?

- a) BCELoss

- b) MSELoss
- c) L1Loss
- d) CTCLoss

**Câu hỏi 102** Dựa vào phân thực nghiệm 2, độ đo đánh giá mô hình nào sau đây không được sử dụng?

- a) F1
- b) Recall
- c) Precision
- d) MAE

## 52.4 Phụ lục

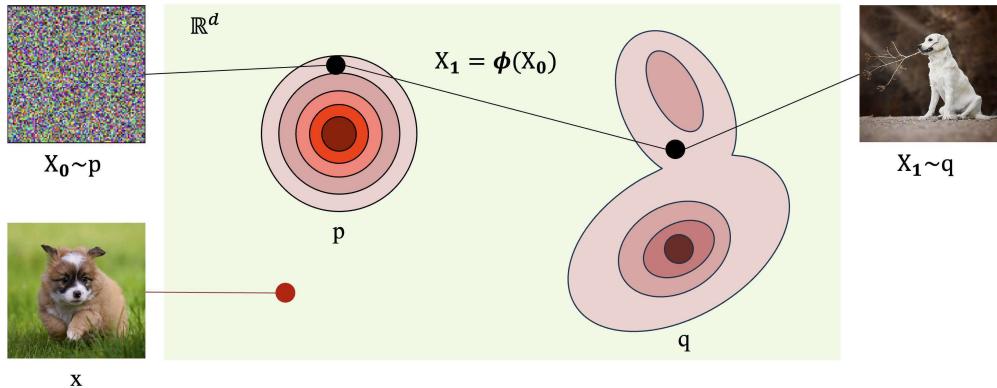
1. **Hint:** Dựa vào file tải về [Image-Inpainting-Denoising-Diffusion-Model](#) để hoàn thiện các đoạn code.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, admin mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

- *Hết* -

# Chương 53

## Flow Matching

### 53.1 Flow Matching



Hình 53.1: Flow Matching.

**Flow Matching** là một phương pháp hiệu quả để huấn luyện các mô hình sinh (generative models) dựa trên dòng (flow). Khác với Diffusion Models, Flow Matching tập trung vào việc học vận tốc chuyển đổi giữa phân phối dữ liệu và phân phối noise, sau đó tạo mẫu bằng cách đi theo vận tốc này.

Trong bài toán mô hình sinh, mục tiêu là tạo ra các mẫu mới từ một phân phối cơ bản (phân phối dữ liệu huấn luyện). Gọi:

- $p_0(x_0)$  là phân phối dữ liệu thực
- $p_1(x_1)$  là phân phối noise (thường là phân phối Gaussian)

Mục tiêu của mô hình sinh là học một hàm ánh xạ  $\phi$  sao cho:

- $X_0 \sim p_0$  (dữ liệu thực)
- $X_1 = \phi(X_0) \sim p_1$  (noise)

Flow Models định nghĩa một quá trình chuyển đổi liên tục giữa hai phân phối. Đối với mỗi thời điểm  $t \in [0, 1]$ , ta có một phân phối trung gian  $p_t$  và một ánh xạ  $\phi_{t+h|t}$  sao cho:

- $X_t \sim p_t$
- $X_{t+h} = \phi_{t+h|t}(X_t)$

Flow Matching là một phương pháp hiệu quả để huấn luyện các mô hình flow bằng cách:

- Học một trường vận tốc (velocity field) thay vì học trực tiếp hàm ánh xạ
- Lấy mẫu bằng cách theo dõi vận tốc này qua phương trình vi phân

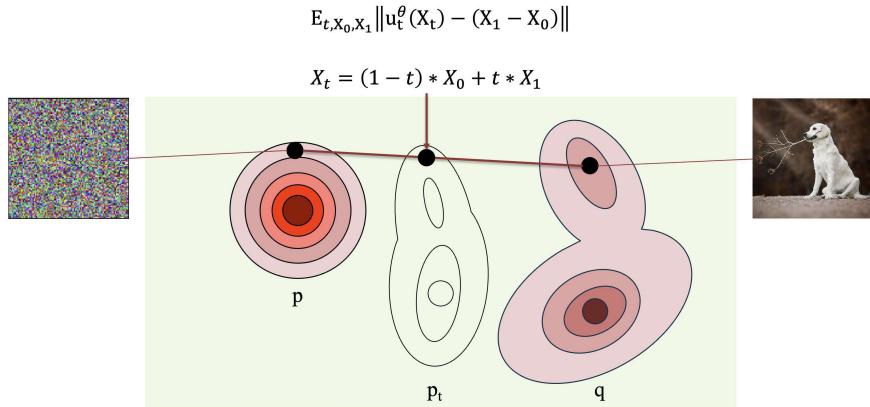
Trong Flow Matching, vận tốc đại diện cho tốc độ chuyển đổi giữa dữ liệu sạch và dữ liệu nhiễu. Cụ thể:

- $v_t$ : vận tốc tại thời điểm  $t$
- $\psi_t(x)$ : flow ánh xạ tại thời điểm  $t$
- $u_t(x)$ : vận tốc thực, được tính bằng  $x_1 - x_0$

### 1.1. Huấn luyện mô hình

Để cập nhật trọng số mô hình dự đoán  $\theta$ , quá trình huấn luyện Flow Matching cho mỗi epoch bao gồm các bước sau:

- (a) Chọn ngẫu nhiên một mẫu dữ liệu sạch  $x_0 \sim p_0$
- (b) Chọn ngẫu nhiên một mẫu noise  $x_1 \sim p_1$
- (c) Chọn ngẫu nhiên một thời điểm  $t \in [0, 1]$
- (d) Tính điểm nội suy (interpolated point)  $x_t = (1 - t)x_0 + tx_1$
- (e) Tính vận tốc thực  $u_t = x_1 - x_0$
- (f) Dự đoán vận tốc bằng mô hình  $v_t = f_\theta(x_t, t)$



Hình 53.2: Training Flow Matching.

- (g) Tính hàm mất mát  $l_t = |v_t - u_t|^2$
- (h) Cập nhật tham số mô hình  $\theta$  bằng gradient descent

Hàm mất mát tổng quát có thể viết dưới dạng:

$$\mathcal{L} = E_{t, X_0, X_1} |u_t^\theta(X_t) - (X_1 - X_0)|^2$$

Trong đó:

- $X_t = (1 - t)X_0 + tX_1$
- $X_0 \sim p_0$
- $X_1 \sim p_1$
- $t \sim \mathcal{U}(0, 1)$

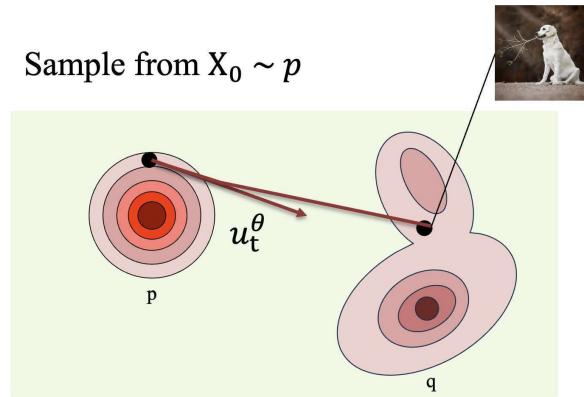
## 1.2. Lấy mẫu

Sau khi huấn luyện mô hình dự đoán vận tốc  $f_\theta(y, t)$ , chúng ta có thể lấy mẫu bằng cách giải phương trình vi phân:

$$\frac{dy}{dt} = f_\theta(y, t)$$

Phương pháp phổ biến để giải phương trình này là phương pháp Euler:

$$y(t + \Delta t) \approx y(t) + f_\theta(y, t)\Delta t$$



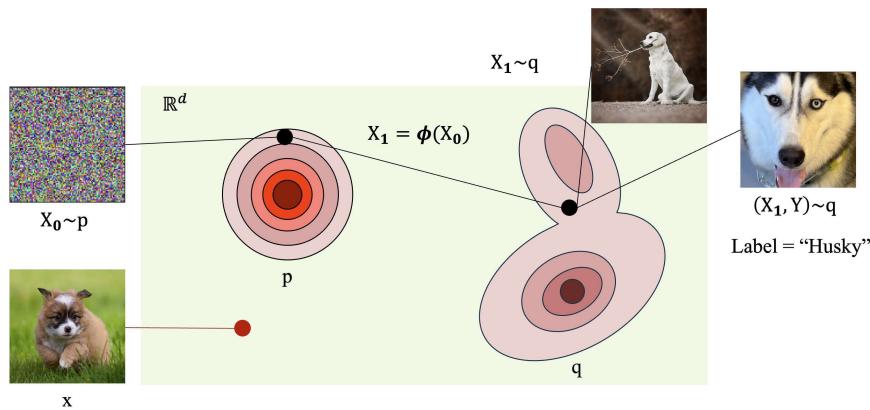
Hình 53.3: Sampling in Flow Matching.

Quá trình lấy mẫu bao gồm các bước:

- Khởi tạo  $y$  từ phân phối noise  $p_1$
- Chia khoảng thời gian  $[0, 1]$  thành  $n$  bước nhỏ:  $t_1, \dots, t_n$ , với  $\Delta t = t_i - t_{i-1}$
- Tại mỗi bước  $i$ :
  - Tính vận tốc  $f_\theta(y, t_i)$
  - Cập nhật  $y = y + f_\theta(y, t_i) \Delta t$
- Kết quả cuối cùng là mẫu từ phân phối  $p_0$

## 53.2 Conditional Flow Matching

Conditional Flow Matching mở rộng Flow Matching với khả năng điều kiện hóa (conditioning). Điều này cho phép mô hình tạo ra dữ liệu có điều kiện dựa trên các thuộc tính hoặc thông tin bổ sung.



Hình 53.4: Conditional Flow Matching.

### 2.1. Dự đoán theo điều kiện

Trong mô hình có điều kiện, chúng ta làm việc với:

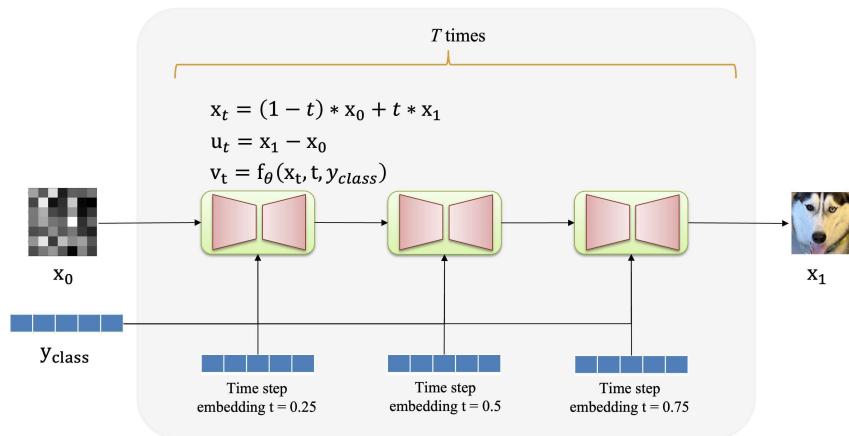
- $X_0 \sim p$ : dữ liệu thực
- $X_1 \sim q$ : noise
- $(X_1, Y) \sim q$ : cặp noise và thông tin điều kiện

Thông tin điều kiện  $Y$  có thể là các nhãn lớp, văn bản, hoặc các dữ liệu khác giúp hướng dẫn quá trình sinh.

### 2.2. Huấn luyện Conditional Flow Matching

Quá trình huấn luyện cho Conditional Flow Matching tương tự như Flow Matching, nhưng bổ sung thông tin điều kiện vào mô hình học sâu để dự đoán các kết quả theo điều kiện:

- a) Chọn ngẫu nhiên một mẫu có điều kiện  $(x_1, y_1) \sim q$



Hình 53.5: Flow Matching.

- (b) Chọn ngẫu nhiên một mẫu dữ liệu sạch  $x_0 \sim p_0$
- (c) Chọn ngẫu nhiên một thời điểm  $t \in [0, 1]$
- (d) Tính điểm nội suy  $x_t = (1 - t)x_0 + tx_1$
- (e) Tính vận tốc thực  $u_t = x_1 - x_0$
- (f) Dự đoán vận tốc bằng mô hình có điều kiện  $v_t = f_\theta(x_t, t, y_1)$
- (g) Tính hàm mất mát  $l_t = |v_t - u_t|^2$
- (h) Cập nhật tham số mô hình  $\theta$  bằng gradient descent

Hàm mất mát tổng quát trở thành:

$$\mathcal{L} = \mathbb{E}_{t, X_0, X_1, Y} |u_t^\theta(X_t, Y) - (X_1 - X_0)|^2$$

### 2.3. Lấy mẫu từ Conditional Flow Matching

Quá trình lấy mẫu cũng tương tự nhưng sử dụng thông tin điều kiện:

$$\begin{aligned} \frac{dy}{dt} &= f_\theta(y, t, y_{class}) \\ y(t + \Delta t) &\approx y(t) + f_\theta(y, t, y_{class})\Delta t \end{aligned}$$

Trong đó  $y_{class}$  là thông tin điều kiện (ví dụ: nhãn lớp) mà chúng ta muốn sử dụng để hướng dẫn quá trình sinh.

### 53.3 Image Inpainting using Conditional Flow Matching

Image Inpainting là kỹ thuật điền vào những phần bị thiếu trong hình ảnh, đảm bảo rằng kết quả:

1. Thực tế về mặt thị giác
2. Hợp lý về ngữ nghĩa

#### 3.1. Các kỹ thuật tạo mặt nạ (masking)

Có nhiều kỹ thuật để tạo mặt nạ cho Image Inpainting:

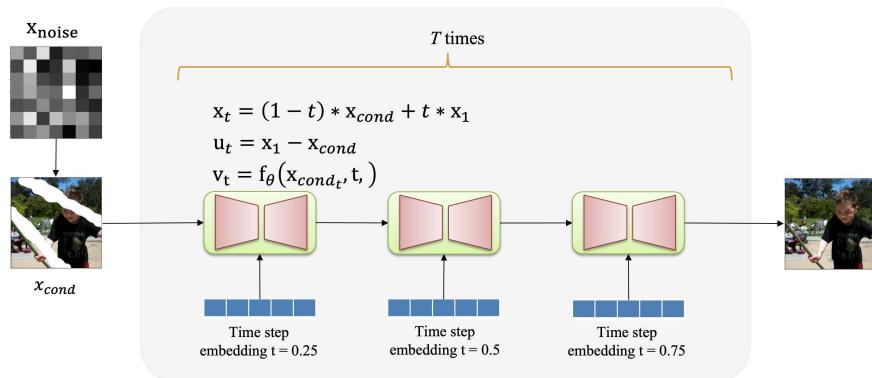
- Center Box Mask: mặt nạ hình chữ nhật ở giữa
- Random Box Mask: mặt nạ hình chữ nhật ở vị trí ngẫu nhiên
- Irregular Mask: mặt nạ có hình dạng không đều
- Free-Form Mask: mặt nạ tự do
- Hybrid Mask: kết hợp nhiều loại mặt nạ

#### 3.2. Mô hình Inpainting sử dụng Conditional Flow Matching

Trong Image Inpainting với Conditional Flow Matching, thông tin điều kiện là hình ảnh không đầy đủ (có phần bị che khuất)  $x_{cond}$ . Mô hình sẽ học cách điền vào phần bị thiếu dựa trên thông tin còn lại.

Quá trình huấn luyện:

- (a) Chọn một hình ảnh đầy đủ  $x_1$
- (b) Tạo mặt nạ và áp dụng lên  $x_1$  để tạo ra hình ảnh điều kiện  $x_{cond}$
- (c) Tính điểm nội suy  $x_t = (1 - t)x_{cond} + tx_1$
- (d) Tính vận tốc thực  $u_t = x_1 - x_{cond}$
- (e) Dự đoán vận tốc bằng mô hình  $v_t = f_\theta(x_t, t, x_{cond})$



Hình 53.6: Flow Matching.

(f) Tính hàm mất mát và cập nhật mô hình

Quá trình sinh (inference):

- Khởi tạo  $x_1$  từ phân phối noise
- Định nghĩa  $x_{cond}$  là hình ảnh không đầy đủ cần điền
- Chia khoảng thời gian  $[0, 1]$  thành các bước
- Tại mỗi bước:
  - Tính vận tốc  $v_t = f_\theta(x_t, t, x_{cond})$
  - Cập nhật  $x_t$  theo phương pháp Euler
- Kết quả cuối cùng là hình ảnh đã được hoàn thiện

### 3.3. Thực nghiệm

Trong phần này chúng ta sẽ tiến hành thực nghiệm huấn luyện mô hình với các bước sau:

- Tải về bộ dữ liệu:

```

1 # Dataset
2 !gdown 194DqJkUjjtlUCp7Sd2DkXgwu7SuBFsrj
3 !unzip celeba_hq_256.zip
4

```

```

5 import os
6 import random
7
8 file_names = os.listdir("./celeba_hq_256")
9 img_paths = ["./celeba_hq_256/" + file_name for file_name
 in file_names]
10 len(img_paths)
11 sample_size = int(len(img_paths) * 0.9)
12 train_imgpaths = random.sample(img_paths, sample_size)
13 val_imgpaths = [img_path for img_path in img_paths if
 img_path not in train_imgpaths]
14 len(train_imgpaths), len(val_imgpaths)

```

(b) Tiên xử lý dữ liệu:

```

1 import numpy as np
2 def bbox2mask(img_shape, bbox, dtype='uint8'):
3 ### bbox (tuple[int]): Configuration tuple, (top,
4 left, height, width)
5 height, width = img_shape[:2]
6 mask = np.zeros((height, width, 1), dtype=dtype)
7 mask[bbox[0]:bbox[0] + bbox[2], bbox[1]:bbox[1] +
 bbox[3], :] = 1
8 return mask
9
10 import torch
11 from PIL import Image
12 from torchvision import transforms
13 from torch.utils.data import Dataset
14 class InpaintingDataset(Dataset):
15 def __init__(self, img_paths, image_size=[256, 256]):
16 self.img_paths = img_paths
17 self.tfs = transforms.Compose([transforms.Resize(
18 (image_size[0], image_size[1])), transforms.ToTensor()
19])
20 self.image_size = image_size
21 def __getitem__(self, index):
22 img_path = self.img_paths[index]
23 img = Image.open(img_path).convert('RGB')
24 img = self.tfs(img)
25 mask = self.get_mask()
26 mask_img = img*(1. - mask) + mask
27 return {
28 "gt_image": img, "cond_image": cond_image,
29 "mask": mask, "path": img_path
30 }

```

```

20 def __len__(self):
21 return len(self.img_paths)
22 def get_mask(self):
23 h, w = self.image_size # Center mask
24 mask = bbox2mask(self.image_size, (h//4, w//4, h
25 //4, w//4))
26 return torch.from_numpy(mask).permute(2,0,1)
27
28 train_dataset = InpaintingDataset(train_imgpaths)
29 batch_size = 64 # (GPU 24GB)
30 train_loader = torch.utils.data.DataLoader(
31 train_dataset, batch_size=batch_size, shuffle=True,
32 drop_last=True
33)

```

Hiển thị một số hình ảnh để kiểm tra:



Hình 53.7: Example.

(c) Huấn luyện mô hình:

```

1 # Model
2 ! pip install -q torchcfm
3
4 from torchcfm.models.unet import UNetModel
5 from tqdm import tqdm
6 device = torch.device("cuda" if torch.cuda.is_available()
7 else "cpu")
8 model = UNetModel(dim=(3, 256, 256), num_channels=32,
9 num_res_blocks=1).to(device)
8 optimizer = torch.optim.Adam(model.parameters())
9

```

```

10 n_epochs = 1000
11 for epoch in range(n_epochs):
12 losses = []
13 for i, data in tqdm(enumerate(train_loader)):
14 optimizer.zero_grad()
15 x1 = data["gt_image"].to(device)
16 mask = data["mask"].to(device)
17 x0 = torch.randn_like(x1).to(device)
18
19 x_noise = (1.0 - mask) * x1 + mask * x0
20 t = torch.rand(x0.shape[0], 1, 1, 1).to(device)
21 xt = t * x1 + (1 - t) * x_noise
22 ut = x1 - x_noise
23
24 t = t.squeeze()
25 x_cond = xt * mask + (1.0 - mask) * x1
26 vt = model(t, x_cond)
27
28 loss = torch.mean(((vt - ut) ** 2) * mask)
29 loss.backward()
30 optimizer.step()
31 losses.append(loss.item())
32 avg_loss = sum(losses) / len(losses)
33
34 print(f"epoch: {epoch}, loss: {avg_loss:.4}")

```

(d) Dự đoán:

```

1 model.eval()
2 def euler_method(model, cond_image, t_steps, dt, mask):
3 y = cond_image
4 y_values = [y]
5 with torch.no_grad():
6 for t in t_steps[1:]:
7 t = t.reshape(-1, 1)
8 dy = model(t.to(device), y)
9 y = y + dy * dt
10 y = cond_image * (1. - mask) + mask * y
11 y_values.append(y)
12 return torch.stack(y_values)
13
14 # Initial random image and class (optional)
15 sample = next(iter(train_loader))
16 gt_image = sample['gt_image'].to(device)
17 noise = torch.randn_like(gt_image, device=device)
18 mask = sample['mask'].to(device)

```

```
19 cond_image = gt_image*(1. - mask) + mask*noise
20
21 # Time parameters
22 t_steps = torch.linspace(0, 1, 50, device=device) # Two
 time steps from 0 to 1
23 dt = t_steps[1] - t_steps[0] # Time step
24
25 # Solve the ODE using Euler method
26 traj = euler_method(model, cond_image, t_steps, dt, mask)
```

Kết quả thử nghiệm



Hình 53.8: Kết quả thực nghiệm mô hình sau khi huấn luyện.

(e) Triển khai ứng dụng:

Triển khai mô hình trên streamlit. Tham khảo về [code](#) và [demo](#).

## Image Inpainting using Conditional Flow Matching

**Model: Conditional Flow Matching. Dataset: CelebA-HQ**

How would you like to give the input?

Run Example Image

Hình 53.9: Kết quả thực nghiệm mô hình sau khi huấn luyện.

## 53.4 Câu hỏi trắc nghiệm

**Câu hỏi 1** Mục tiêu chính của Flow Matching là gì?

- a) Tạo mẫu ngẫu nhiên từ phân phối đã cho
- b) Học một trường vận tốc (velocity field) để biến đổi một phân phối dữ liệu sang phân phối khác
- c) Tính toán ma trận Jacobian của mạng nơ-ron
- d) Giảm nhiễu trong dữ liệu hình ảnh

**Câu hỏi 2** Mục tiêu của việc học velocity trong Flow Matching là gì?

- a) Dự đoán sự thay đổi giữa dữ liệu nhiễu và dữ liệu sạch
- b) Dự đoán sự phân bố của dữ liệu nhiễu
- c) Tạo dữ liệu có điều kiện từ phân phối gốc
- d) Giảm nhiễu trong dữ liệu âm thanh

**Câu hỏi 3** Công thức nào sau đây biểu diễn quá trình sampling trong Flow Matching?

- a)  $\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + f(\mathbf{x}(t), t)\Delta t$
- b)  $\mathbf{x}(t + \Delta t) = \mathbf{x}(t) - f(\mathbf{x}(t), t)\Delta t$
- c)  $\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \sigma(t) \cdot \epsilon(t)\Delta t$
- d)  $\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + f_\theta(\mathbf{x}(t), t)\Delta t$

**Câu hỏi 4** Công thức nào sau đây mô tả sự thay đổi giữa dữ liệu nhiễu và dữ liệu sạch trong Flow Matching?

- a)  $v_t = x_1 - x_0$
- b)  $v_t = x_1 + x_0$
- c)  $v_t = x_1/x_0$
- d)  $v_t = \sigma(t) + x_0$

**Câu hỏi 5** Quá trình image inpainting sử dụng Conditional Flow Matching có mục tiêu gì?

- a) Sinh hình ảnh từ nhãn lớp
- b) Phục hồi các phần bị thiếu trong hình ảnh sao cho chúng hợp lý về mặt trực quan và ngữ nghĩa
- c) Thêm nhiễu vào hình ảnh
- d) Dự đoán các giá trị bị thiếu trong dữ liệu âm thanh

**Câu hỏi 6** Trong Image Inpainting sử dụng Conditional Flow Matching, kỹ thuật masking nào sau đây được sử dụng để che khuất phần dữ liệu?

- a) Random Masking
- b) Irregular Masking
- c) Free-Form Masking
- d) Tất cả các kỹ thuật trên

**Câu hỏi 7** Trong image inpainting với Conditional Flow Matching, công thức nào sau đây mô tả quá trình cập nhật dữ liệu bị thiếu?

- a)  $x_t = (1 - t) \cdot x_{\text{cond}} + t \cdot x_1$
- b)  $x_t = x_1 + (1 - t) \cdot x_0$
- c)  $x_t = t \cdot x_{\text{cond}} + (1 - t) \cdot x_0$
- d)  $x_t = x_0 + x_1$

**Câu hỏi 8** Bộ dữ liệu được sử dụng trong thực nghiệm là?

- a) CelebA-HQ
- b) MNIST
- c) CIFAR10
- d) CIFAR100

**Câu hỏi 9** Kích thước đầu vào của mô hình UNET trong phần thực nghiệm là?

- a) 3x256x256
- b) 3x224x224
- c) 3x160x160
- d) 3x64x64

**Câu hỏi 10** Các timesteps trong phần Sampling được khởi tạo thông qua phương pháp nào?

- a) sin
- b) cos
- c) linspace
- d) tanh

### 53.5 Phụ lục

1. **Hint:** Dựa vào file tải về [Image-Inpainting-Conditional-Flow-Matching](#) để hoàn thiện các đoạn code.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, admin mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

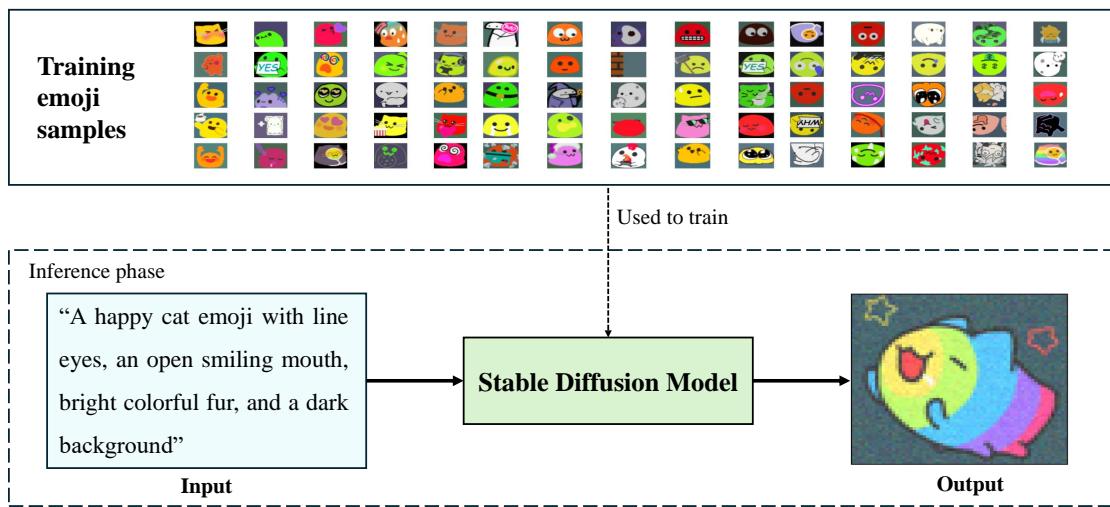
- *Hết* -

# Chương 54

## Project 2: Áp dụng Stable Diffusion cho bài toán tô màu ảnh

### 54.1 Giới thiệu

Emoji Generation (Tạm dịch: Tạo sinh ảnh biểu tượng cảm xúc) là một bài toán thuộc lĩnh vực Thị giác máy tính (Computer Vision) và Mô hình tạo sinh (Generative Models), tập trung vào việc xây dựng một hệ thống có khả năng tạo ra các emoji mới dựa trên mô tả văn bản hoặc các đặc trưng hình ảnh đầu vào. Trên thế giới, nhiều nghiên cứu và ứng dụng đã khai thác mô hình học sâu để tạo ra nội dung hình ảnh mới, chẳng hạn như [DALL-E](#) của OpenAI, [Imagen](#) của Google, hay [Stable Diffusion](#). Các mô hình này có thể tạo ra hình ảnh từ mô tả văn bản với độ chân thực và sáng tạo cao. Trong bài toán này, chúng ta sẽ áp dụng **Stable Diffusion Model**, một mô hình khuếch tán mạnh mẽ, để sinh ra các emoji theo yêu cầu. Mục tiêu của bài toán không chỉ dừng lại ở việc sao chép các emoji có sẵn, mà còn hướng đến khả năng sáng tạo, giúp tạo ra các emoji hoàn toàn mới nhưng vẫn giữ được phong cách nhất quán với một bộ emoji gốc.



Hình 54.1: Minh họa về bài toán sinh emoji bằng Stable Diffusion Model.

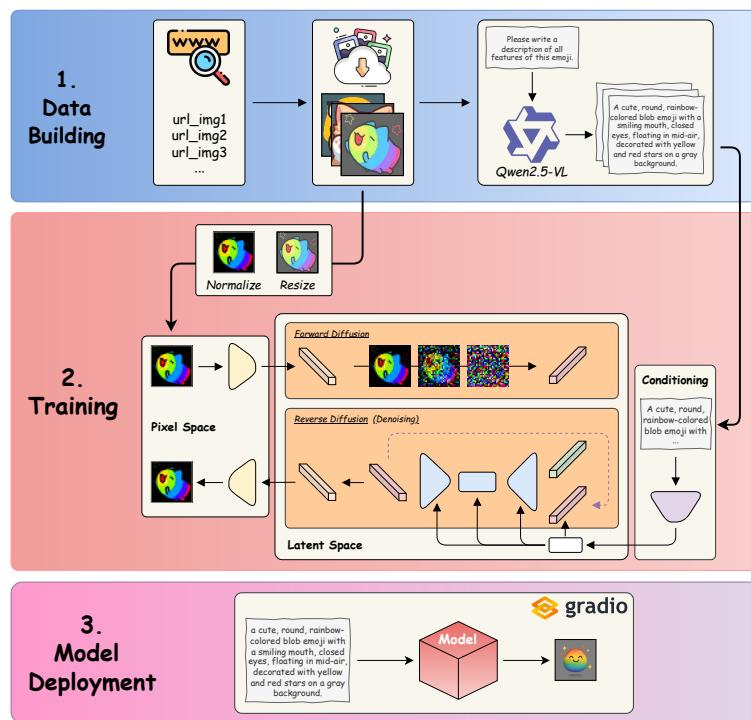
#### 54.1.1 Mô tả bài toán

Bài toán được mô tả như sau:

- **Input:** Một prompt mô tả emoji cần tạo (hình dáng, màu sắc, biểu cảm, nền). Ví dụ: "Một emoji chú mèo dễ thương, có màu cầu vồng, miệng cười, mắt nhắm, đang lơ lửng giữa không trung, nền tối màu".
- **Output:** Hình ảnh emoji mới được tạo theo prompt, mang phong cách từ tập dữ liệu emoji đã được huấn luyện.

#### 54.1.2 Project pipeline

Dựa trên các mô tả nội dung trên, ta có một pipeline tổng quát cho toàn bộ project được mô tả trong sơ đồ sau:



Hình 54.2: Pipeline tổng quát của bài toán sinh Emoji bằng Stable Diffusion Model.

Toàn bộ quy trình sinh ảnh emoji từ mô tả văn bản được tổ chức thành ba giai đoạn chính: **Data Building**, **Training**, và **Model Deployment**, như minh họa trong Hình 54.2.

### 1. Data Building (Xây dựng bộ dữ liệu)

Giai đoạn đầu tiên là xây dựng bộ dữ liệu để huấn luyện mô hình. Hệ thống tự động thu thập hình ảnh emoji từ Internet thông qua các đường dẫn URL. Sau khi tải về, ảnh được chuẩn hóa (normalize) và thay đổi kích thước (resize) về định dạng cố định. Đồng thời, mỗi ảnh được chú thích bằng văn bản mô tả chi tiết, chẳng hạn: "A happy cat emoji with line eyes, an open smiling mouth, bright colorful fur, and a dark background" - "Một emoji chú mèo dễ thương, có màu cầu vồng, miệng cười, mắt nhắm, đang lơ lửng giữa không trung, được trang trí bằng các ngôi sao màu vàng và đỏ trên nền xám.". Ta sử dụng mô hình ngôn ngữ đa phương thức **Qwen2.5-VL** để tự động hóa quá trình tạo

nhãn mô tả sử dụng nhằm sinh ra caption từ ảnh, đóng vai trò là điều kiện đầu vào (conditioning prompt) trong quá trình huấn luyện.

## 2. Training (Huấn luyện mô hình)

Mô hình huấn luyện chính là **Stable Diffusion** — một mô hình khuếch tán (diffusion model) hiện đại trong lĩnh vực sinh ảnh. Ảnh sau khi chuẩn hóa được mã hóa vào không gian điểm ảnh (pixel space), sau đó đưa vào pipeline khuếch tán. Trong quá trình khuếch tán (forward diffusion), ảnh gốc được thêm nhiễu qua nhiều bước để chuyển vào không gian tiềm ẩn (latent space). Tiếp đó, giai đoạn khuếch tán ngược (reverse diffusion) học cách tái tạo lại ảnh từ nhiễu, với sự hỗ trợ từ thông tin điều kiện (prompt) đã sinh.

Trong mã nguồn, quá trình huấn luyện được xây dựng dựa trên thư viện **diffusers** của Hugging Face. Thư viện này cung cấp sẵn các thành phần mô hình cần thiết để thực hiện quy trình khuếch tán, được tổ chức thành một pipeline gọi là **StableDiffusionPipeline**.

**StableDiffusionPipeline** bao gồm các thành phần chính như bộ mã hóa văn bản (text encoder, ví dụ: CLIP hoặc T5), bộ tạo nhiễu (noise scheduler, chẳng hạn: DDIMScheduler), và mô hình U-Net để dự đoán nhiễu trong quá trình huấn luyện. Quá trình huấn luyện bao gồm các bước: lấy mẫu nhiễu, tính toán hàm mất mát và cập nhật trọng số của mô hình qua nhiều vòng lặp.

Trong quá trình huấn luyện, mô hình sử dụng hàm mất mát dạng **MSE** giữa nhiều thực tế và nhiều dự đoán. Việc lựa chọn này xuất phát từ thực tế là mô hình sử dụng VAE đã được huấn luyện sẵn (pretrained) và cố định trọng số, do đó không tính thêm thành phần KL Divergence như trong công thức gốc của mô hình Latent Diffusion. Đồng thời, với độ phân giải ảnh thấp (32x32), hàm MSE đơn thuần vẫn cho kết quả học hiệu quả và giúp giảm chi phí tính toán. Phía dưới đây, chúng ta sẽ đi vào tìm hiểu công thức hàm mất mát đầy đủ của mô hình Stable Diffusion theo đúng lý thuyết gốc sau.

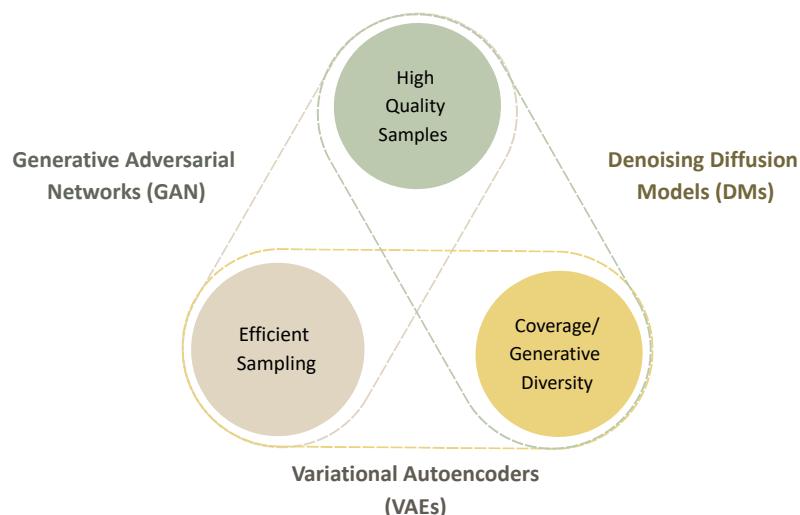
## 3. Model Deployment (Triển khai mô hình)

Sau khi huấn luyện xong, mô hình được chuẩn bị sẵn sàng để triển khai dưới dạng một ứng dụng đơn giản, có thể tương tác qua giao diện người dùng được xây dựng bằng thư viện Gradio. Người dùng chỉ cần nhập vào một mô tả emoji mong muốn, hệ thống sẽ tự động sinh ra hình ảnh emoji mới phù hợp với nội dung mô tả đó. Giao diện Gradio giúp quá trình sử dụng trở nên dễ dàng và trực quan, đồng thời hỗ trợ đánh giá chất lượng ảnh đầu ra một cách nhanh chóng.

### 54.1.3 Tổng quan về Stable Diffusion Model

#### Motivation

Trước khi đi vào chi tiết mô hình Stable Diffusion, hãy cùng nhìn lại lý do vì sao nó ra đời, dù trước đó đã có nhiều mô hình tạo sinh mạnh mẽ như GAN, VAE hay Diffusion Models.



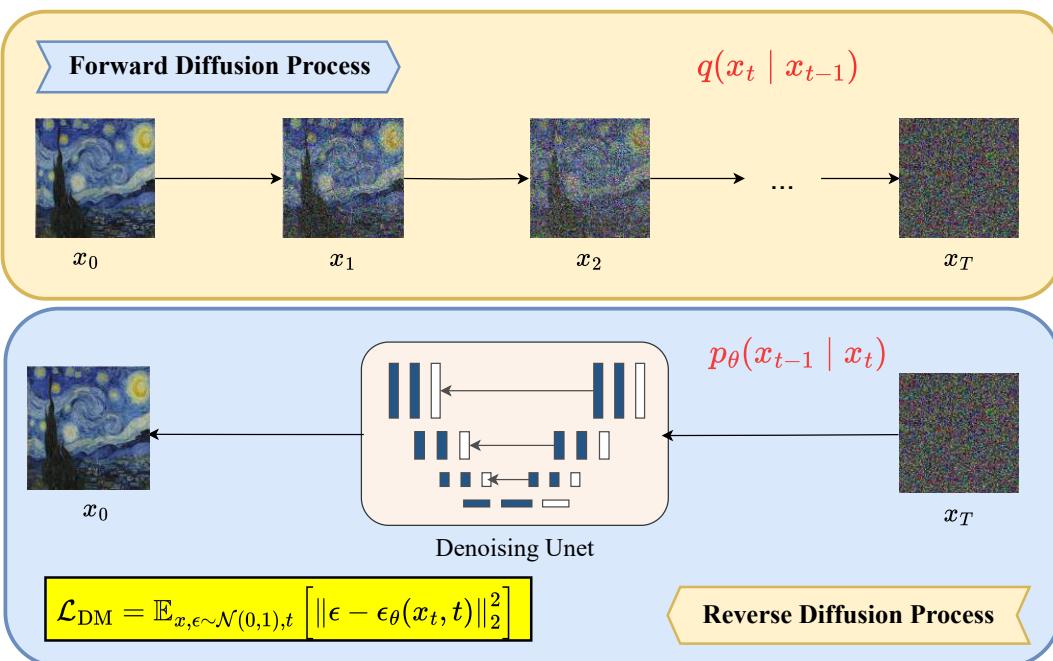
Hình 54.3: So sánh tổng quan các dòng mô hình sinh ảnh.

- **GANs** tạo ảnh sắc nét nhưng khó huấn luyện, dễ gặp lỗi mode collapse (mô hình chỉ sinh ra một số kiểu ảnh nhất định).
- **VAEs** ổn định hơn nhưng thường cho ảnh mờ, kém chân thực do giới hạn trong giả định phân phối tiềm ẩn.

- **Diffusion Models** nổi bật vì cho ảnh chất lượng cao, ổn định hơn GAN và mô hình hóa phân phối dữ liệu tốt hơn VAEs. Tuy nhiên, tốc độ sinh ảnh rất chậm.

Để hiểu nguyên nhân, ta cần nhắc lại cơ chế hoạt động của DMs (minh họa trong Hình 54.4), DMs tạo ảnh thông qua hai giai đoạn chính:

- **Forward Process (quá trình khuếch tán):** dần thêm nhiễu vào ảnh gốc  $x_0$  để thu được ảnh nhiễu hoàn toàn  $x_T$ .
- **Reverse Process (quá trình khử nhiễu):** sử dụng Denoising U-Net để tái tạo ảnh gốc từ  $x_T$  qua chuỗi bước  $x_T \rightarrow x_{T-1} \rightarrow \dots \rightarrow x_0$ .



Hình 54.4: Cấu trúc tổng quát và luồng xử lý của Diffusion Model truyền thống.

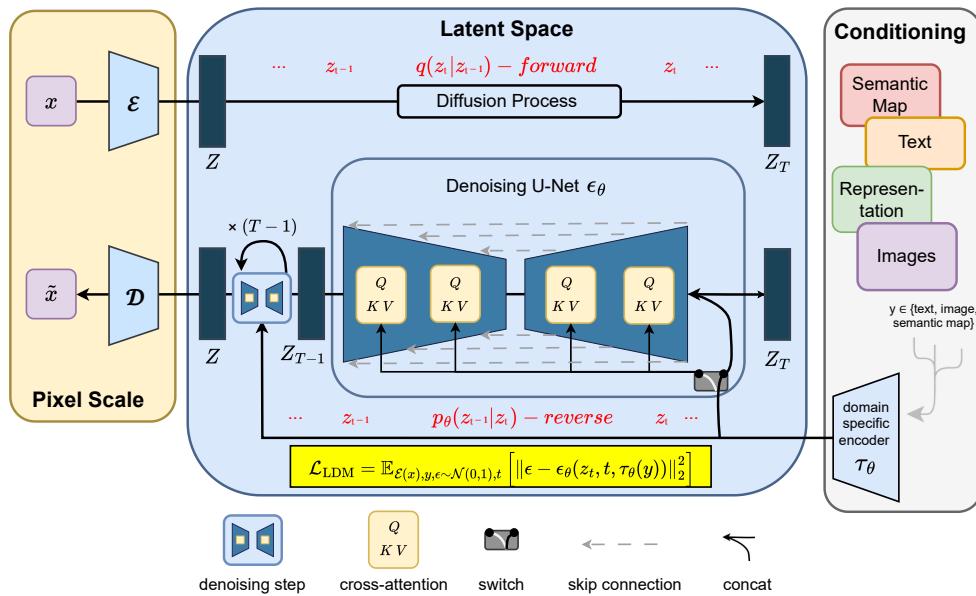
Ta thấy được vấn đề nằm ở việc toàn bộ quá trình khuếch tán và khử nhiễu diễn ra trên ảnh kích thước lớn, nên mỗi bước sampling đều rất tốn tài

nguyên. Hơn nữa, cần hàng trăm bước như vậy để khôi phục ảnh đầu ra. Đây là nguyên nhân chính khiến DMs trở nên chậm và khó ứng dụng thực tế, đặc biệt khi cần sinh ảnh theo thời gian thực.

Vì vậy, dù tạo ra ảnh có chất lượng tốt, DMs truyền thống vẫn gặp giới hạn nghiêm trọng về tốc độ xử lý và khả năng mở rộng. Chính hạn chế này đã thúc đẩy sự ra đời của **Stable Diffusion**.

### Stable Diffusion

Stable Diffusion thuộc nhóm **Latent Diffusion Models (LDMs)**, được giới thiệu bởi (Rombach et al., 2022). Ý tưởng chính của Stable Diffusion là, thay vì khuếch tán trực tiếp ảnh gốc  $x \in \mathbb{R}^{H \times W \times 3}$ , mô hình trước tiên sẽ mã hóa ảnh sang một vector đặc trưng  $z \in \mathbb{R}^d$  bằng encoder của VAE, sau đó thực hiện quá trình thêm nhiễu và khử nhiễu trong không gian này.



Hình 54.5: Kiến trúc của Stable Diffusion Model.

Kiến trúc trong Hình 54.5 cho ta thấy quá trình hoạt động của Stable Diffusion Model gồm ba bước chính:

1. **Encoding (mã hóa):** ảnh đầu vào  $x$  được encoder của VAE mã hóa thành vector tiềm ẩn  $z$ .
2. **Latent diffusion (khuếch tán trong không gian tiềm ẩn):** mô hình thêm nhiễu vào  $z$  để thu được  $z_t$ , sau đó dùng **U-Net** để dự đoán nhiễu  $\epsilon$  nhằm tái tạo lại vector gốc.
3. **Decoding (giải mã):** vector sạch  $z$  sau khi khử nhiễu được đưa qua decoder để tái tạo ảnh đầu ra.

Trong quá trình khử nhiễu, mô hình có thể nhận thêm **conditioning information (thông tin điều kiện)** như văn bản mô tả  $y$ , semantic map hoặc ảnh. Văn bản sẽ được mã hóa bởi một **text encoder** thành vector  $\tau_\theta(y)$ , rồi đưa vào U-Net thông qua **cross-attention** để ảnh sinh ra bám sát nội dung mô tả. Quá trình học được dẫn dắt bởi hàm mất mát có điều kiện:

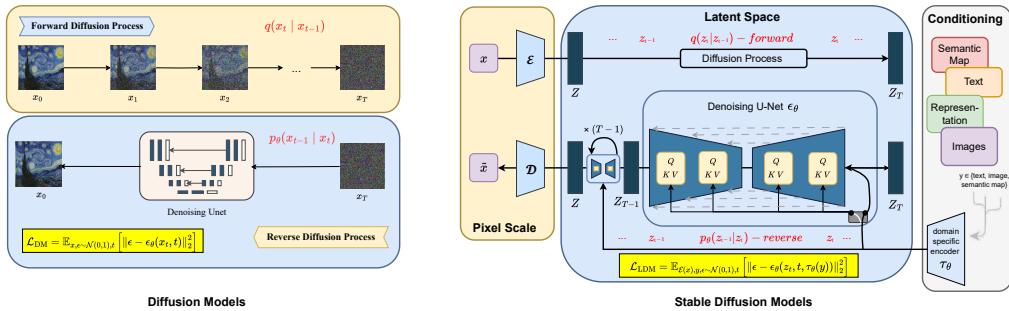
$$\mathcal{L}_{\text{LDM}} := \mathbb{E}_{\epsilon(x), y, \epsilon \sim \mathcal{N}(0, 1), t} \left[ \|\epsilon - \epsilon_\theta(z_t, t, \tau_\theta(y))\|_2^2 \right], \quad \text{trong đó:}$$

- $\epsilon \sim \mathcal{N}(0, 1)$ : nhiễu ngẫu nhiên được thêm vào.
- $z_t$ : vector latent sau khi bị thêm nhiễu tại bước  $t$ .
- $\epsilon_\theta$ : nhiễu được mô hình dự đoán.

Nhờ thực hiện toàn bộ quá trình khuếch tán trong **latent space**, kích thước đầu vào của U-Net giảm mạnh (thường 8–16 lần), từ đó tăng tốc huấn luyện và suy diễn (inference), giảm tiêu tốn tài nguyên mà vẫn giữ được chất lượng ảnh đầu ra ở mức cao.

### So sánh kiến trúc: Diffusion Models vs. Stable Diffusion Models

Điểm khác biệt cốt lõi giữa Diffusion Models (DMs) truyền thống và Stable Diffusion nằm ở không gian xử lý. Trong khi DMs thực hiện khuếch tán trực tiếp trên ảnh gốc có kích thước lớn  $x \in \mathbb{R}^{H \times W \times 3}$ , Stable Diffusion lựa chọn khuếch tán trong không gian tiềm ẩn (latent space)  $z \in \mathbb{R}^d$ , vốn đã được nén lại nhờ encoder của VAE. Điều này giúp giảm đáng kể chi phí tính toán, đồng thời tăng tốc độ suy diễn.



Hình 54.6: So sánh kiến trúc giữa Diffusion Model truyền thống và Stable Diffusion.

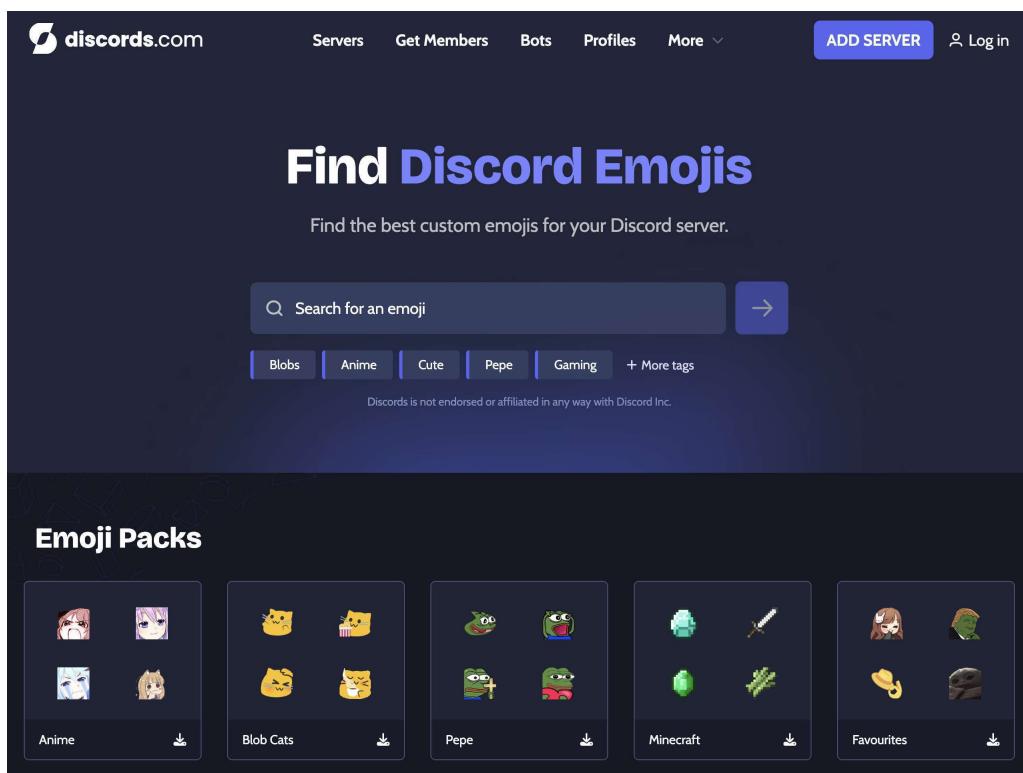
Bên cạnh đó, Stable Diffusion tích hợp thêm thông tin điều kiện (conditioning) như văn bản mô tả thông qua một text encoder huấn luyện sẵn (thường là CLIP). Vector hóa của prompt  $y$ , ký hiệu  $\tau_\theta(y)$ , được đưa vào mạng U-Net thông qua cơ chế cross-attention, cho phép mô hình kiểm soát nội dung ảnh sinh ra một cách hiệu quả.

## 54.2 Cài đặt chương trình

Trong phần này, chúng ta sẽ tìm hiểu về quá trình xây dựng toàn bộ chương trình Emoji Image Generation sử dụng mô hình Stable Diffusion (SD). Trong đó, bao gồm hai phần nội dung lớn là thu thập bộ dữ liệu để huấn luyện mô hình và xây dựng mô hình Stable Diffusion.

### 54.2.1 Thu thập bộ dữ liệu

Trong project này, chúng ta giả định chưa có sẵn bộ dữ liệu, vì vậy cần thực hiện thu thập dữ liệu. Dựa trên nội dung project, các file ảnh emoji sẽ được thu thập để huấn luyện mô hình SD. Nguồn dữ liệu từ internet là dễ tiếp cận nhất, và trong phạm vi project này, thư viện Selenium sẽ được sử dụng để thu thập dữ liệu từ trang web lưu trữ emoji, cụ thể là <https://discords.com/emoji-list>.



Hình 54.7: Trang web chứa các ảnh emoji có thể sử dụng trên ứng dụng Discord.

**Lưu ý:** Các bạn có thể tải trực tiếp bộ dữ liệu đã được thu thập sẵn ở phần 60.4 và bỏ qua phần này.

### Cài đặt thư viện Selenium

Chúng ta sẽ thực hiện thu thập dữ liệu trên Google Colab. Để sử dụng Selenium trên môi trường này, chúng ta cần thực thi đoạn code sau:

```

1 %%shell
2 # Ubuntu no longer distributes chromium-browser outside of snap
3 #
4 # Proposed solution: https://askubuntu.com/questions/1204571/how-to-install-chromium-without-snap
5
6 # Add debian buster
7 cat > /etc/apt/sources.list.d/debian.list << "EOF"

```

```
8 deb [arch=amd64 signed-by=/usr/share/keyrings/debian-buster.gpg] http://
 deb.debian.org/debian buster main
9 deb [arch=amd64 signed-by=/usr/share/keyrings/debian-buster-updates.gpg]
 http://deb.debian.org/debian buster-
 updates main
10 deb [arch=amd64 signed-by=/usr/share/keyrings/debian-security-buster.gpg]
 http://deb.debian.org/debian-security
 buster/updates main
11 EOF
12
13 # Add keys
14 apt-key adv --keyserver keyserver.ubuntu.com --recv-keys DCC9EFCBF77E11517
15 apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 648ACFD622F3D138
16 apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 112695A0E562B32A
17
18 apt-key export 77E11517 | gpg --dearmour -o /usr/share/keyrings/debian-
 buster.gpg
19 apt-key export 22F3D138 | gpg --dearmour -o /usr/share/keyrings/debian-
 buster-updates.gpg
20 apt-key export E562B32A | gpg --dearmour -o /usr/share/keyrings/debian-
 security-buster.gpg
21
22 # Prefer debian repo for chromium* packages only
23 # Note the double-blank lines between entries
24 cat > /etc/apt/preferences.d/chromium.pref << "EOF"
25 Package: *
26 Pin: release a=eoan
27 Pin-Priority: 500
28
29
30 Package: *
31 Pin: origin "deb.debian.org"
32 Pin-Priority: 300
33
34
35 Package: chromium*
36 Pin: origin "deb.debian.org"
37 Pin-Priority: 700
38 EOF
39
40 # Install chromium and chromium-driver
41 apt-get update
42 apt-get install chromium chromium-driver
43
44 # Install selenium
```

45 | pip install selenium

## Import các thư viện cần thiết

```
1 import os
2 import requests
3 import time
4 import pandas as pd
5 import random
6 import hashlib
7 import urllib.parse
8 from io import BytesIO
9 from PIL import Image
10
11 from tqdm import tqdm
12 from selenium import webdriver
13 from selenium.webdriver.chrome.service import Service
14 from selenium.webdriver.common.by import By
15 from selenium.webdriver.support.ui import WebDriverWait
16 from selenium.webdriver.support import expected_conditions as EC
```

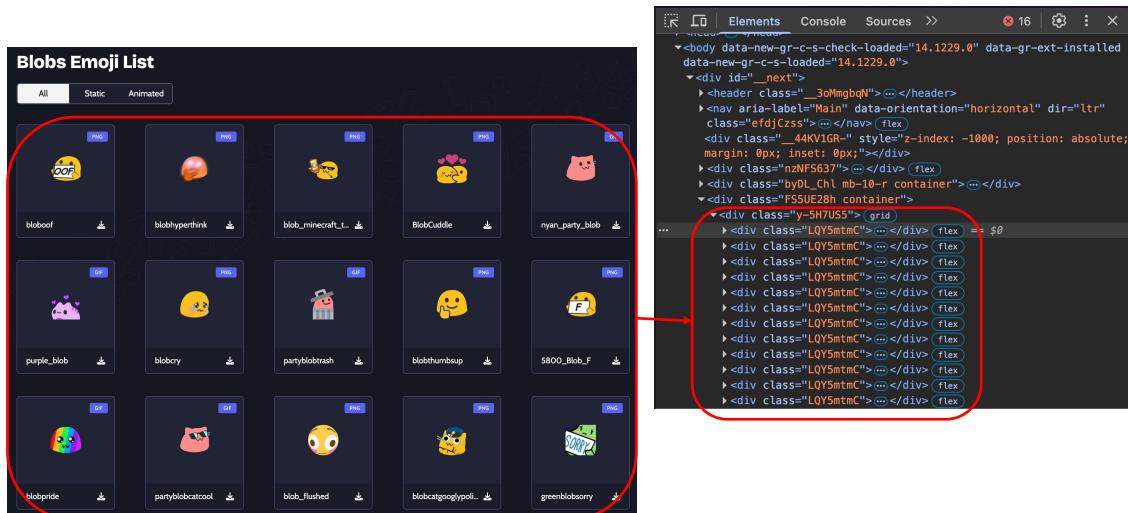
## Khởi tạo driver trình duyệt web

Bước đầu tiên với Selenium, ta cần khởi tạo một trình duyệt web để có thể sử dụng trong các đoạn code sau:

```
1 WEBDRIVER_DELAY_TIME_INT = 20
2 TIMEOUT_INT = 20
3 service = Service(executable_path=r"/usr/bin/chromedriver")
4 chrome_options = webdriver.ChromeOptions()
5 chrome_options.add_argument("--headless")
6 chrome_options.add_argument("--no-sandbox")
7 chrome_options.add_argument("--disable-dev-shm-usage")
8 chrome_options.add_argument("window-size=1920x1080")
9 chrome_options.headless = True
10 driver = webdriver.Chrome(service=service, options=chrome_options)
11 driver.implicitly_wait(TIMEOUT_INT)
12 wait = WebDriverWait(driver, WEBDRIVER_DELAY_TIME_INT)
```

## Xây dựng hàm trích xuất đường dẫn ảnh từ trang web

Xét danh sách các emoji thuộc thẻ (tag) "Blobs", ta có thể thấy danh sách các emoji được thể hiện trong giao diện người dùng như ở hình 54.8.



Hình 54.8: Danh sách các emojis có tag Blobs của một trang (page) được đối chiếu giữa giao diện web và cấu trúc HTML.

Các ô ảnh này đều được thể hiện bằng một thẻ div bên trong file HTML của trang web với cùng một class. Vì vậy, khi có được đường dẫn của một trang danh sách emoji, ta hoàn toàn có thể sử dụng code để trích xuất toàn bộ các đường dẫn ảnh cũng như một số thông tin khác về ảnh. Theo đó, ta sẽ có đoạn code trích xuất toàn bộ đường dẫn của một trang page như sau:

```

1 def get_image_links_from_page(page_url, driver):
2 driver.get(page_url)
3 try:
4 container = wait.until(EC.presence_of_element_located(
5 (By.CSS_SELECTOR, "div.FS5UE28h.container"))
6)
7 image_items = wait.until(EC.presence_of_all_elements_located(
8 (By.CSS_SELECTOR, "div.LQY5mtmC div.aLnnpRah.text-center")))
9)
10
11 image_links = []
12 for img_elem in image_items:

```

```

13 img_div = img_elem.find_element(By.CSS_SELECTOR, "div.Mw1EAtrx
14 img, img")
15
16 img_url = img_div.get_attribute("src")
17 img_title = img_div.get_attribute("title")
18 if img_url:
19 image_links.append((img_url, img_title))
20
21 return image_links
22 except Exception as e:
23 print(f"Error while trying to extract images: {e}")
24 return []

```

### Xây dựng hàm kiểm tra ảnh trùng

Đối với trang này, có một điểm cần lưu ý là ta có thể sẽ bắt gặp các ảnh emoji trùng khi thực hiện thu thập dữ liệu. Vì vậy, để tránh tình trạng lưu trữ các ảnh trùng, ta xây dựng một hàm kiểm tra ảnh trùng dựa vào đường dẫn ảnh với thư viện **hashlib** như sau:

```

1 def hash_image_content(url):
2 try:
3 response = requests.get(url, stream=True)
4 if response.status_code == 200:
5 return hashlib.md5(response.content).hexdigest()
6 else:
7 print(f"Error downloading image from {url}; status: {response.
8 status_code}")
8 return None
9 except requests.exceptions.RequestException as e:
10 print(f"Error with the image download for {url}: {e}")
11 return None

```

### Xây dựng hàm đổi định dạng ảnh

Khi quan sát đường dẫn ảnh trong trang web, ta có thể nhận thấy định dạng của các ảnh này thuộc đuôi .webp. Vì vậy, ta cần thực hiện chuyển đổi về định dạng ảnh quen thuộc là .jpg để có thể tương tác với ảnh dễ dàng hơn. Theo đó, ta triển khai hàm `convert_webp_to_jpg()` như sau:

```

1 def convert_webp_to_jpg(webp_data):

```

```

1 try:
2 img = Image.open(BytesIO(webp_data))
3 if img.format == 'WEBP':
4 if img.mode == 'RGBA':
5 img = img.convert('RGB')
6 buffer = BytesIO()
7 img.save(buffer, format="JPEG")
8 return buffer.getvalue()
9 else:
10 return webp_data
11 except Exception as e:
12 print(f"Error converting WebP to JPG: {e}")
13 return webp_data
14

```

## Xây dựng hàm tải ảnh

Dĩ nhiên, để tải được ảnh từ một đường dẫn chứa ảnh, ta sẽ xây dựng một hàm tải ảnh sử dụng thư viện `requests` như sau:

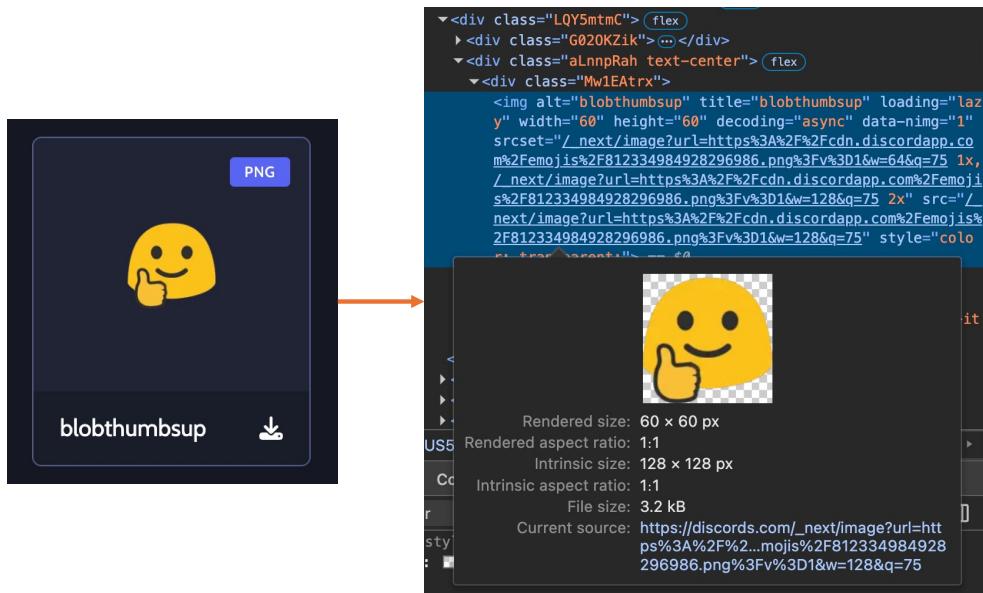
```

1 def download_image(img_url, img_name, folder_path):
2 try:
3 response = requests.get(img_url, stream=True)
4 if response.status_code == 200:
5 img_path = os.path.join(folder_path, f"{img_name}")
6 img_data = convert_webp_to_jpg(response.content)
7 with open(img_path, "wb") as f:
8 f.write(img_data)
9 else:
10 print(f"Error downloading image from {img_url}; status: {response.status_code}")
11 except requests.exceptions.RequestException as e:
12 print(f"Error with the image download for {img_url}: {e}")
13

```

## Xây dựng hàm thu thập thông tin của một ảnh

Tổng hợp tất cả các hàm đang có từ phía trên, ta xây dựng một hàm xử lý thông tin của một đường dẫn ảnh. Theo đó bao gồm việc kiểm tra ảnh trùng, thực hiện tải ảnh và lưu các thông tin ảnh dưới dạng dictionary, thuận tiện cho việc lưu thành file .csv ở các đoạn code tiếp theo. Như vậy, ta có code triển khai như sau:



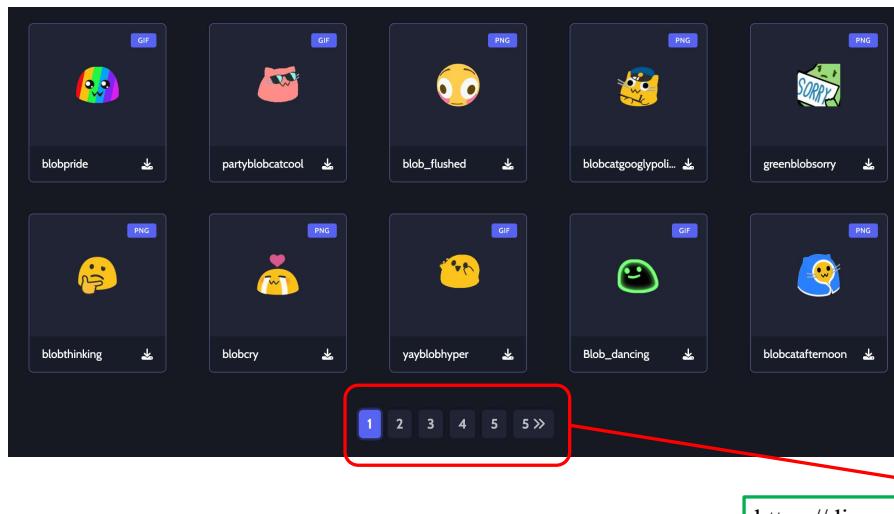
Hình 54.9: Một số thông tin của một ảnh emoji được thể hiện qua thẻ img trong file HTML.

```

1 def process_image_page(image_url, img_title, folder_path, idx, tag,
2 seen_hashes):
3 img_hash = hash_image_content(image_url)
4 if img_hash and img_hash not in seen_hashes:
5 seen_hashes.add(img_hash)
6 new_file_name = f"{tag}_{idx:07d}.jpg"
7 download_image(image_url, new_file_name, folder_path)
8 metadata = {
9 "file_name": new_file_name,
10 "image_url": image_url,
11 "image_title": img_title,
12 "tag": tag
13 }
14 return metadata
15 else:
16 return None

```

## Xây dựng hàm duyệt qua từng trang web



<https://discords.com/emoji-list/tag/Blobs?page=1>

Hình 54.10: Tận dụng tham số query bên trong đường dẫn trang chứa các emojis để có thể tự động duyệt qua các trang tiếp theo.

```

1 def loop_over_pages(base_url, tags, total_pages, driver, folder_path):
2 os.makedirs(folder_path, exist_ok=True)
3 all_metadata = []
4 seen_hashes = set()
5
6 for tag in tags:
7 all_images = []
8
9 for page in tqdm(range(1, total_pages + 1), desc=f"Extracting
10 Images for {tag}", unit="page"):
11 page_url = f"{base_url}/emoji-list/tag/{tag}?page={page}"
12 images = get_image_links_from_page(page_url, driver)
13 all_images.extend(images)
14
15 time.sleep(1)
16
17 metadata_list = []
18 for idx, (img_url, img_title) in enumerate(all_images, start=1):
19 metadata = process_image_page(img_url, img_title, folder_path,
20 idx, tag, seen_hashes)

```

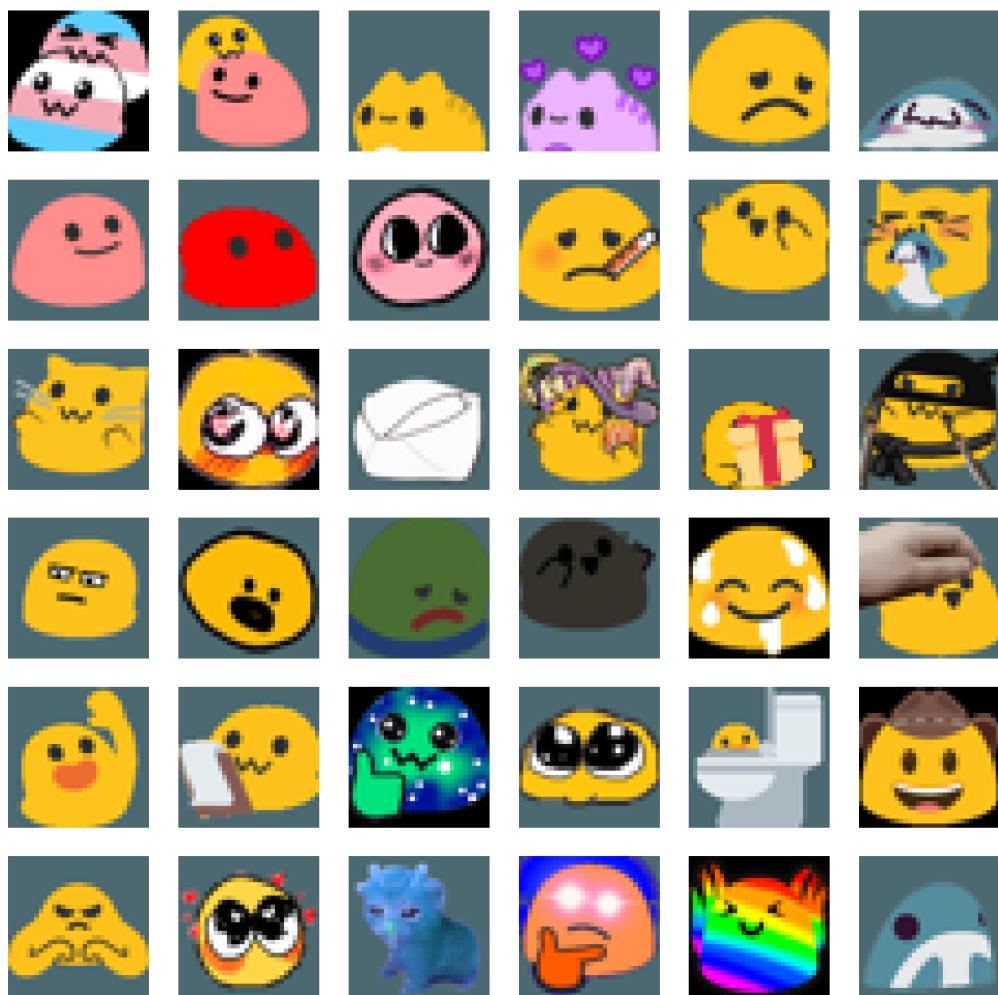
```
19 if metadata:
20 metadata_list.append(metadata)
21
22 all_metadata.extend(metadata_list)
23
24 return all_metadata
```

Với hàm trên, ta cho phép xử lý một danh sách các tag emoji khác nhau, song trong project này chúng ta sẽ chỉ tập trung vào tag **Blobs**. Với mỗi tag sẽ được thông qua hai vòng lặp, một dùng để trích xuất toàn bộ các thông tin ảnh và hai là thực hiện xử lý thông tin đó dựa trên các hàm đã xây dựng ở những bước trước. Khi kết thúc, ta trả về một list chứa metadata của các ảnh emoji.

### Xây dựng hàm lưu thông tin metadata

Với list các metadata trả về từ hàm `loop_over_pages()`, ta thực hiện lưu lại thành file .csv thông qua hàm sau:

```
1 def save_metadata(metadata_list, metadata_file):
2 df = pd.DataFrame(metadata_list)
3 df.to_csv(metadata_file, index=False, encoding="utf-8")
```



Hình 54.11: Trực quan hóa một số hình ảnh mà chúng ta đã thu thập được.

### Thực hiện thu thập dữ liệu

Cuối cùng, với toàn bộ các hàm trên, ta tổng hợp lại để tiến hành thực hiện thu thập dữ liệu ảnh emoji với code triển khai sau đây:

```

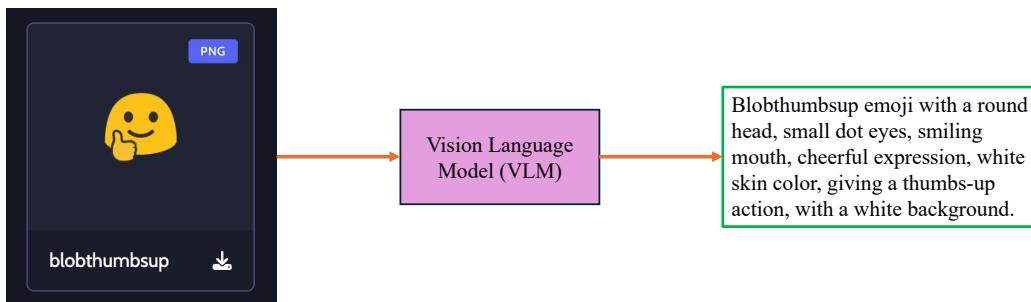
1 os.makedirs("crawled_data", exist_ok=True)
2 folder_path = os.path.join("crawled_data", "images")
3 metadata_file = os.path.join("crawled_data", "metadata.csv")
4
5 base_url = "https://discords.com"

```

```
6 tags = ["Blobs"]
7 total_pages = 1000
8
9 metadata_list = loop_over_pages(base_url, tags, total_pages, driver,
10 folder_path)
11 save_metadata(metadata_list, metadata_file)
12
13 print("Start downloading images...")
14 with tqdm(total=len(metadata_list), desc="Downloading Images", unit="image
15 ") as pbar:
16 for metadata in metadata_list:
17 img_url = metadata['image_url']
18 file_name = metadata['file_name']
19 download_image(img_url, file_name, folder_path)
20 pbar.update(1)
21
22 print("Download images completed.")
23 total_crawled_images = len(os.listdir(folder_path))
24 print(f"Total crawled images: {total_crawled_images}.")
25 driver.quit()
```

### Tạo mô tả cho ảnh emoji

Vì mô hình Stable Diffusion (SD) với input là một câu prompt mô tả ảnh muốn tạo sinh, trong khi dữ liệu ảnh hiện tại chưa có thông tin mô tả chi tiết. Một cách để chúng ta có thể giải quyết vấn đề này nhanh chóng đó là tận dụng một mô hình ngôn ngữ lớn để giúp chúng ta tạo nhanh các mô tả thông qua prompting.



Hình 54.12: Sử dụng mô hình ngôn ngữ thị giác lớn (VLM) để tạo mô tả cho các ảnh emoji.

Song, trong nội dung file mô tả của project, chúng ta sẽ không đề cập chi tiết về phần sử dụng mô hình ngôn ngữ thị giác lớn. Thay vào đó, bộ dữ liệu emoji với mô tả đính kèm sẽ được cung cấp sẵn.

### 54.2.2 Xây dựng mô hình Stable Diffusion

Trong phần này, ta sẽ thực hiện triển khai mô hình Stable Diffusion sử dụng thư viện PyTorch và Diffusers của HuggingFace để huấn luyện mô hình trên bộ dữ liệu emoji mà chúng ta đã chuẩn bị. Các bước thực hiện như sau:

#### Import các thư viện cần thiết

```
1 import os
2 import math
3 import torch
4 import random
5 import numpy as np
6 import pandas as pd
7 import matplotlib.pyplot as plt
8
9 from torch import nn
10 from PIL import Image
11 from tqdm import tqdm
12 from diffusers import AutoencoderKL
13 from torch.nn import functional as F
14 from torchvision import transforms
15 from torch.utils.data import Dataset, DataLoader
16 from transformers import CLIPTokenizer, CLIPTextModel
17 import torch.optim.lr_scheduler as lr_scheduler
18 from torch.amp import GradScaler, autocast
```

#### Cố định tham số ngẫu nhiên

Nhằm mục đích có thể tái tạo lại kết quả đã đạt được trong mỗi lần chạy lại chương trình, ta thực hiện cố định trạng thái ngẫu nhiên cho toàn bộ các hàm, module có liên quan đến các phép ngẫu nhiên như sau:

```
1 def set_seed(seed=42):
2 random.seed(seed)
3 np.random.seed(seed)
4 torch.manual_seed(seed)
5 torch.cuda.manual_seed(seed)
6 torch.cuda.manual_seed_all(seed)
7 torch.backends.cudnn.deterministic = True
8 torch.backends.cudnn.benchmark = False
9 os.environ["PYTHONHASHSEED"] = str(seed)
```

```

10 print(f"Seed set to {seed}")
11
12 set_seed()
13 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

```

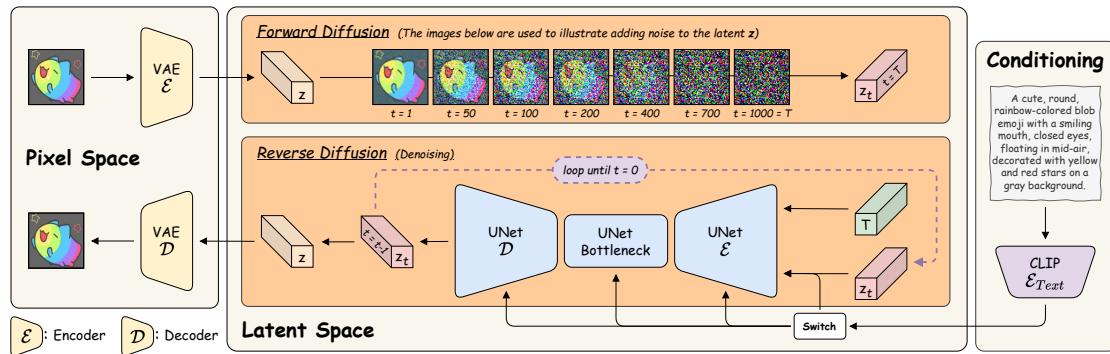
## Tải bộ dữ liệu

Chúng ta thực hiện tải bộ dữ liệu đã thu thập được ở phần trước vào notebook mới này thông qua lệnh sau:

```

1 # https://drive.google.com/file/d/15Z_F4Dwgb3NLqEGnVMUEJqyxXgW7Gx-h/view?
 usp=sharing
2 !gdown 15Z_F4Dwgb3NLqEGnVMUEJqyxXgW7Gx-h
3 !unzip blobs_crawled_data.zip

```



Hình 54.13: Minh họa tổng quát kiến trúc mạng Stable Diffusion cho bài toán tạo sinh hình emoji.

## Xây dựng các class Attention

Trong Stable Diffusion có bao gồm kiến trúc U-Net. Kiến trúc này có tận dụng các phép Attention nhằm cải thiện khả xử lý mối quan hệ không gian trong ảnh và vấn đề phụ thuộc dài hạn trong quá trình tạo sinh ảnh. Vì vậy, ở bước này, ta cần định nghĩa các lớp Attention bao gồm SelfAttention và CrossAttention như sau:

```

1 class SelfAttention(nn.Module):
2 def __init__(self, num_attn_heads, hidden_dim, in_proj_bias=True,
3 out_proj_bias=True):

```

```

3 super().__init__()
4 self.num_heads = num_attn_heads
5 self.head_size = hidden_dim // num_attn_heads
6
7 self.qkv_proj = nn.Linear(hidden_dim, 3 * hidden_dim, bias=
8 in_proj_bias)
9 self.output_proj = nn.Linear(hidden_dim, hidden_dim, bias=
10 out_proj_bias)
11
12
13 def forward(self, features, use_causal_mask=False):
14 b, s, d = features.shape
15
16 qkv_combined = self.qkv_proj(features)
17 q_mat, k_mat, v_mat = torch.chunk(qkv_combined, 3, dim=-1)
18
19 q_mat = q_mat.view(b, s, self.num_heads, self.head_size).permute(0
20 , 2, 1, 3)
21 k_mat = k_mat.view(b, s, self.num_heads, self.head_size).permute(0
22 , 2, 1, 3)
23 v_mat = v_mat.view(b, s, self.num_heads, self.head_size).permute(0
24 , 2, 1, 3)
25
26
27 qk = torch.matmul(q_mat, k_mat.transpose(-2, -1))
28 sqrt_qk = qk / math.sqrt(self.head_size)
29
30 if use_causal_mask:
31 causal_mask = torch.triu(torch.ones_like(sqrt_qk, dtype=torch.
32 bool), diagonal=1)
33 sqrt_qk = sqrt_qk.masked_fill(causal_mask, -torch.inf)
34
35
36 attn_weights = torch.softmax(sqrt_qk, dim=-1)
37 attn_values = torch.matmul(attn_weights, v_mat)
38
39
40 attn_values = attn_values.permute(0, 2, 1, 3).contiguous()
41 attn_values = attn_values.view(b, s, d)
42
43
44 final_output = self.output_proj(attn_values)
45 return final_output

```

```

1 class CrossAttention(nn.Module):
2 def __init__(self, num_attn_heads, query_dim, context_dim,
3 in_proj_bias=True, out_proj_bias=True):
4 super().__init__()
5 self.num_heads = num_attn_heads

```

```

5 self.head_size = query_dim // num_attn_heads
6
7 self.query_map = nn.Linear(query_dim, query_dim, bias=in_proj_bias
8)
8 self.key_map = nn.Linear(context_dim, query_dim, bias=in_proj_bias
9)
9 self.value_map = nn.Linear(context_dim, query_dim, bias=
10 in_proj_bias)
10
11 self.output_map = nn.Linear(query_dim, query_dim, bias=
12 out_proj_bias)
12
13 def forward(self, query_input, context_input):
14 b_q, s_q, d_q = query_input.shape
15 _, s_kv, _ = context_input.shape
16
17 q_mat = self.query_map(query_input)
18 k_mat = self.key_map(context_input)
19 v_mat = self.value_map(context_input)
20
21 q_mat = q_mat.view(b_q, s_q, self.num_heads, self.head_size).
22 permute(0, 2, 1, 3)
22 k_mat = k_mat.view(b_q, s_kv, self.num_heads, self.head_size).
23 permute(0, 2, 1, 3)
23 v_mat = v_mat.view(b_q, s_kv, self.num_heads, self.head_size).
24 permute(0, 2, 1, 3)
24
25 qk = torch.matmul(q_mat, k_mat.transpose(-2, -1))
26 sqrt_qk = qk / math.sqrt(self.head_size)
27 attn_weights = torch.softmax(sqrt_qk, dim=-1)
28
29 attn_values = torch.matmul(attn_weights, v_mat)
30 attn_values = attn_values.permute(0, 2, 1, 3).contiguous()
31 attn_values = attn_values.view(b_q, s_q, d_q)
32
33 final_output = self.output_map(attn_values)
34 return final_output

```

## Khai báo class DDPM

Một thành phần không thể thiếu đối với cài đặt của Stable Diffusion đó là DDPM (Denoising Diffusion Probabilistic Models). Để triển khai thành phần này, ta thực hiện như sau:

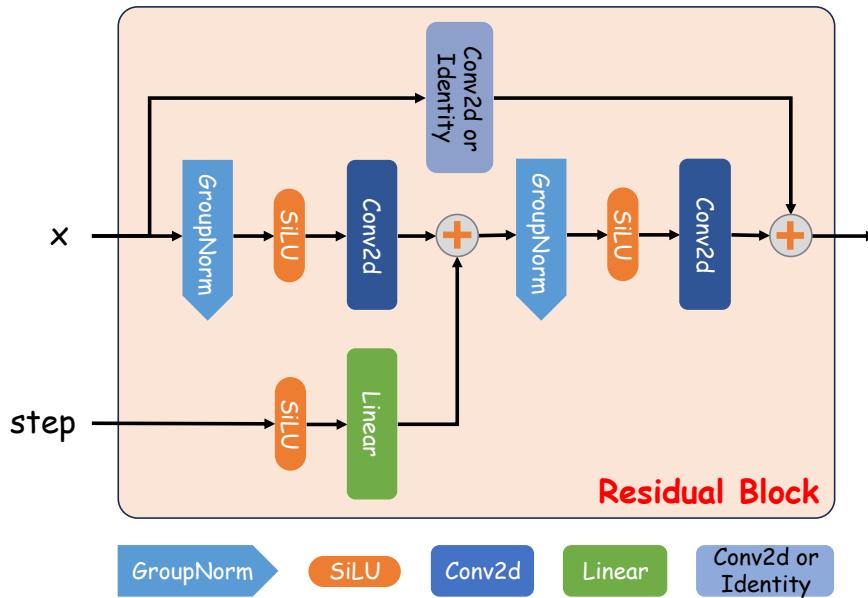
```
1 class DDPMscheduler:
2 def __init__(
3 self,
4 random_generator,
5 train_timesteps=1000,
6 diffusion_beta_start=0.00085,
7 diffusion_beta_end=0.012
8):
9
10 self.betas = torch.linspace(
11 diffusion_beta_start ** 0.5, diffusion_beta_end ** 0.5,
12 train_timesteps,
13 dtype=torch.float32) ** 2
14 self.alphas = 1.0 - self.betas
15 self.alphas_cumulative_product = torch.cumprod(self.alphas, dim=0)
16 self.one_val = torch.tensor(1.0)
17 self.prng_generator = random_generator
18 self.total_train_timesteps = train_timesteps
19 self.schedule_timesteps = torch.from_numpy(np.arange(0,
20 train_timesteps)[::-1].copy())
21
22 def set_steps(self, num_sampling_steps=50):
23 self.num_sampling_steps = num_sampling_steps
24 step_scaling_factor = self.total_train_timesteps // self.
25 num_sampling_steps
26 timesteps_for_sampling = (
27 np.arange(0, num_sampling_steps) * step_scaling_factor
28 .round()[:-1].copy().astype(np.int64)
29 self.schedule_timesteps = torch.from_numpy(timesteps_for_sampling)
30
31 def _get_prior_timestep(self, current_timestep):
32 previous_t = current_timestep - self.total_train_timesteps // self
33 .num_sampling_steps
34 return previous_t
35
36 def _calculate_variance(self, timestep):
37 prev_t = self._get_prior_timestep(timestep)
38 alpha_cumprod_t = self.alphas_cumulative_product[timestep]
39 alpha_cumprod_t_prev = self.alphas_cumulative_product[prev_t] if
40 prev_t >= 0 else self.one_val
41 beta_t_current = 1 - alpha_cumprod_t / alpha_cumprod_t_prev
42 variance_value = (1 - alpha_cumprod_t_prev) / (1 - alpha_cumprod_t
43) * beta_t_current
44 variance_value = torch.clamp(variance_value, min=1e-20)
45 return variance_value
```

```
40
41 def adjust_strength(self, strength_level=1):
42 initial_step_index = self.num_sampling_steps - int(self.
43 num_sampling_steps * strength_level)
44 self.schedule_timesteps = self.schedule_timesteps[
45 initial_step_index:]
46 self.start_sampling_step = initial_step_index
47
48 def step(self, current_t, current_latents, model_prediction):
49 t = current_t
50 prev_t = self._get_prior_timestep(t)
51
52 alpha_cumprod_t = self.alphas_cumulative_product[t]
53 alpha_cumprod_t_prev = self.alphas_cumulative_product[prev_t] if
54 prev_t >= 0 else self.one_val
55 beta_cumprod_t = 1 - alpha_cumprod_t
56 beta_cumprod_t_prev = 1 - alpha_cumprod_t_prev
57 alpha_t_current = alpha_cumprod_t / alpha_cumprod_t_prev
58 beta_t_current = 1 - alpha_t_current
59
60 predicted_original = (current_latents - beta_cumprod_t ** 0.5 *
61 model_prediction) / alpha_cumprod_t
62 ** 0.5
63
64 original_coeff = (alpha_cumprod_t_prev ** 0.5 * beta_t_current) /
65 beta_cumprod_t
66 current_coeff = alpha_t_current ** 0.5 * beta_cumprod_t_prev /
67 beta_cumprod_t
68
69 predicted_prior_mean = original_coeff * predicted_original +
70 current_coeff * current_latents
71
72 variance_term = 0
73 if t > 0:
74 target_device = model_prediction.device
75 noise_component = torch.randn(
76 model_prediction.shape,
77 generator=self.prng_generator,
78 device=target_device,
79 dtype=model_prediction.dtype
80)
81 variance_term = (self._calculate_variance(t) ** 0.5) *
82 noise_component
83
84 predicted_prior_sample = predicted_prior_mean + variance_term
```

```
76 return predicted_prior_sample
77
78 def add_noise(self, initial_samples, noise_timesteps):
79 alphas_cumprod = self.alphas_cumulative_product.to(
80 device=initial_samples.device,
81 dtype=initial_samples.dtype
82)
83 noise_timesteps = noise_timesteps.to(initial_samples.device)
84 sqrt_alpha_cumprod = alphas_cumprod=noise_timesteps] ** 0.5
85 sqrt_alpha_cumprod = sqrt_alpha_cumprod.view(
86 sqrt_alpha_cumprod.shape[0], *[1] * (initial_samples.ndim - 1
87))
88 sqrt_one_minus_alpha_cumprod = (1 - alphas_cumprod=noise_timesteps
89]) ** 0.5
90 sqrt_one_minus_alpha_cumprod = sqrt_one_minus_alpha_cumprod.view(
91 sqrt_one_minus_alpha_cumprod.shape[0], *[1] * (
92 initial_samples.ndim - 1))
93 random_noise = torch.randn(
94 initial_samples.shape, generator=self.prng_generator,
95 device=initial_samples.device, dtype=initial_samples.dtype
96)
97 noisy_result = sqrt_alpha_cumprod * initial_samples +
98 sqrt_one_minus_alpha_cumprod *
99 random_noise
100
101 return noisy_result, random_noise
```

## Khai báo kiến trúc U-Net

Với các class đã khai báo bên trên, ta tiến hành định nghĩa một số các thành phần liên quan đến kiến trúc U-Net được sử dụng trong Stable Diffusion như sau:



Hình 54.14: Minh họa kiến trúc của khối Residual trong Unet.

```

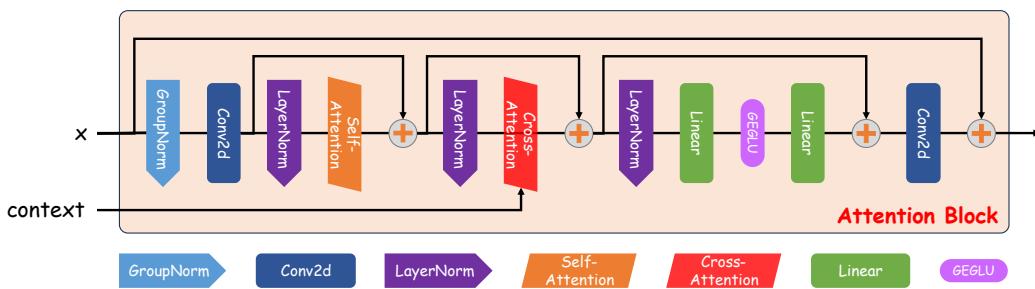
1 class UNET_ResidualBlock(nn.Module):
2 def __init__(self, in_channels, out_channels, time_dim=1280):
3 super().__init__()
4 self.gn_feature = nn.GroupNorm(32, in_channels)
5 self.conv_feature = nn.Conv2d(in_channels, out_channels,
6 kernel_size=3, padding=1)
6 self.time_embedding_proj = nn.Linear(time_dim, out_channels)
7
8 self.gn_merged = nn.GroupNorm(32, out_channels)
9 self.conv_merged = nn.Conv2d(out_channels, out_channels,
10 kernel_size=3, padding=1)
11
11 if in_channels == out_channels:
12 self.residual_connection = nn.Identity()
13 else:
14 self.residual_connection = nn.Conv2d(in_channels, out_channels,
15 kernel_size=1, padding=0)
15
16 def forward(self, input_feature, time_emb):
17 residual = input_feature
18
19 h = self.gn_feature(input_feature)
20 h = F.silu(h)

```

```

21 h = self.conv_feature(h)
22
23 time_emb_processed = F.silu(time_emb)
24 time_emb_projected = self.time_embedding_proj(time_emb_processed)
25 time_emb_projected = time_emb_projected.unsqueeze(-1).unsqueeze(-1)
26
27 merged_feature = h + time_emb_projected
28 merged_feature = self.gn_merged(merged_feature)
29 merged_feature = F.silu(merged_feature)
30 merged_feature = self.conv_merged(merged_feature)
31
32 output = merged_feature + self.residual_connection(residual)
33 return output

```



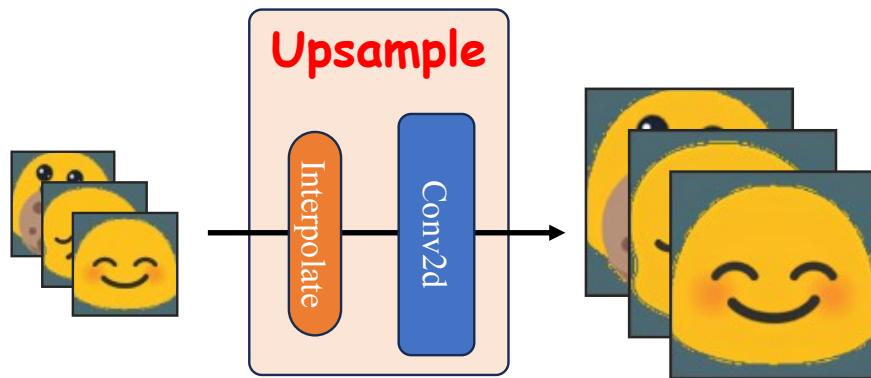
Hình 54.15: Minh họa kiến trúc của khối Attention trong Unet.

```

1 class UNET_AttentionBlock(nn.Module):
2 def __init__(self, num_heads, head_dim, context_dim=512):
3 super().__init__()
4 embed_dim = num_heads * head_dim
5
6 self.gn_in = nn.GroupNorm(32, embed_dim, eps=1e-6)
7 self.proj_in = nn.Conv2d(embed_dim, embed_dim, kernel_size=1,
8 padding=0)
9
10 self.ln_1 = nn.LayerNorm(embed_dim)
11 self.attn_1 = SelfAttention(num_heads, embed_dim, in_proj_bias=
12 False)
13 self.ln_2 = nn.LayerNorm(embed_dim)
14 self.attn_2 = CrossAttention(num_heads, embed_dim, context_dim,
15 in_proj_bias=False)
16 self.ln_3 = nn.LayerNorm(embed_dim)

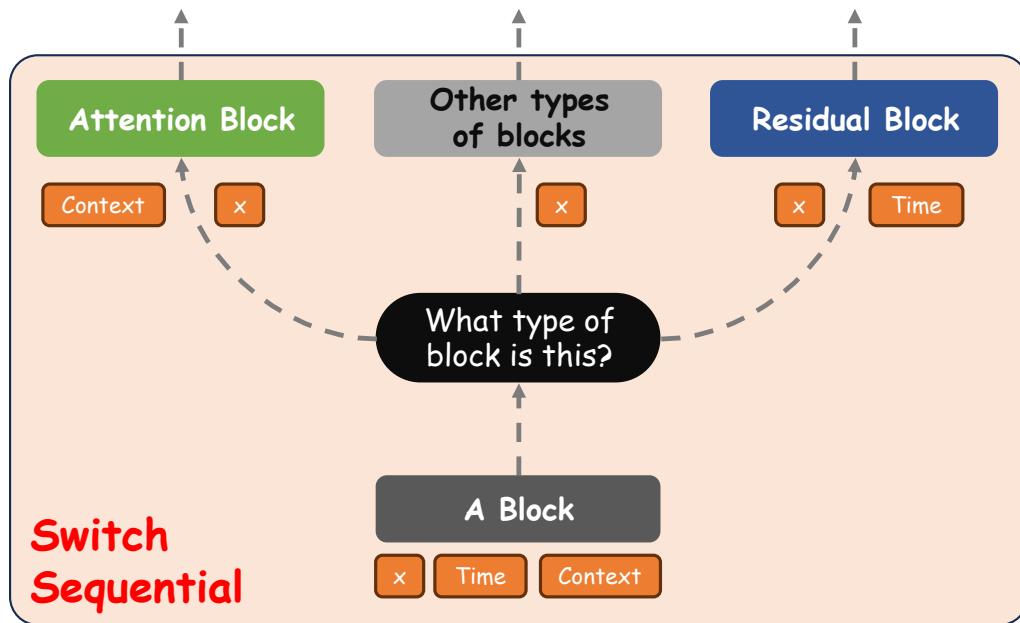
```

```
15 self.ffn_gelu = nn.Linear(embed_dim, 4 * embed_dim * 2)
16 self.ffn_out = nn.Linear(4 * embed_dim, embed_dim)
17 self.proj_out = nn.Conv2d(embed_dim, embed_dim, kernel_size=1,
18 padding=0)
19
20 def forward(self, input_tensor, context_tensor):
21 skip_connection = input_tensor
22
23 B, C, H, W = input_tensor.shape
24 HW = H * W
25
26 h = self.gn_in(input_tensor)
27 h = self.proj_in(h)
28 h = h.view(B, C, HW).transpose(-1, -2)
29
30 attn1_skip = h
31 h = self.ln_1(h)
32 h = self.attn_1(h)
33 h = h + attn1_skip
34
35 attn2_skip = h
36 h = self.ln_2(h)
37 h = self.attn_2(h, context_tensor)
38 h = h + attn2_skip
39
40 ffn_skip = h
41 h = self.ln_3(h)
42 intermediate, gate = self.ffn_gelu(h).chunk(2, dim=-1)
43 h = intermediate * F.gelu(gate)
44 h = self.ffn_out(h)
45 h = h + ffn_skip
46
47 h = h.transpose(-1, -2).view(B, C, H, W)
48 output = self.proj_out(h) + skip_connection
49 return output
```



Hình 54.16: Minh họa kiến trúc của khối Upsample trong Unet.

```
1 class Upsample(nn.Module):
2 def __init__(self, num_channels):
3 super().__init__()
4 self.conv = nn.Conv2d(num_channels, num_channels, kernel_size=3,
5 padding=1)
6
7 def forward(self, feature_map):
8 x = F.interpolate(feature_map, scale_factor=2, mode="nearest")
9 x = self.conv(x)
10 return x
```



Hình 54.17: Minh họa kiến trúc của khối Switch Sequential trong Unet.

```

1 class SwitchSequential(nn.Sequential):
2 def forward(self, x, guidance_context, time_embedding):
3 for module_instance in self:
4 if isinstance(module_instance, UNET_AttentionBlock):
5 x = module_instance(x, guidance_context)
6 elif isinstance(module_instance, UNET_ResidualBlock):
7 x = module_instance(x, time_embedding)
8 else:
9 x = module_instance(x)
10 return x
11
12
13 class TimeEmbedding(nn.Module):
14 def __init__(self, n_embd):
15 super().__init__()
16 self.proj1 = nn.Linear(n_embd, 4 * n_embd)
17 self.proj2 = nn.Linear(4 * n_embd, 4 * n_embd)
18
19 def forward(self, x):
20 x = self.proj1(x)
21 x = F.silu(x)
22 x = self.proj2(x)

```

23 | **return** x

## Xây dựng hàm mã hóa thông tin thời gian

Để đưa thông tin thời gian (timestep) và ngữ cảnh điều kiện vào quá trình Denoising, ta sẽ triển khai các hàm sau với mục tiêu mã hóa thông tin timestep thành vector:

```

1 def embed_a_timestep(timestep, embedding_dim=320):
2 half_dim = embedding_dim // 2
3 freqs = torch.exp(-math.log(10000) *
4 torch.arange(start=0, end=half_dim, dtype=torch.
5 float32) /
6 half_dim)
7 x = torch.tensor([timestep], dtype=torch.float32)[:, None] * freqs[
8 None]
9 return torch.cat([torch.cos(x), torch.sin(x)], dim=-1)
10
11 def embed_timesteps(timesteps, embedding_dim=320):
12 half_dim = embedding_dim // 2
13 freqs = torch.exp(-math.log(10000) *
14 torch.arange(half_dim, dtype=torch.float32) /
15 half_dim).to(device=timesteps.device)
16 args = timesteps[:, None].float() * freqs[None, :]
17 return torch.cat([torch.cos(args), torch.sin(args)], dim=-1)

```

## Khai báo Diffusion model

```

1 class Diffusion(nn.Module):
2 def __init__(self, h_dim=128, n_head=4):
3 super().__init__()
4 self.time_embedding = TimeEmbedding(320)
5 self.unet = UNET(h_dim, n_head)
6 self.unet_output = UNETOutputLayer(h_dim, 4)
7
8 @torch.autocast(
9 device_type="cuda", dtype=torch.float16,
10 enabled=True, cache_enabled=True
11)
12 def forward(self, latent, context, time):
13 time = self.time_embedding(time)
14 output = self.unet(latent, context, time)

```

```
15 output = self.unet_output(output)
16 return output
```

## Khai báo mô hình CLIP

Để mã hóa thông tin mô tả của một ảnh thành dạng vector, ta sử dụng mô hình CLIP. Code cài đặt như sau:

```
1 class CLIPTextEncoder(nn.Module):
2 def __init__(self):
3 super().__init__()
4 CLIP_id = "openai/clip-vit-base-patch32"
5 self.tokenizer = CLIPTokenizer.from_pretrained(CLIP_id)
6 self.text_encoder = CLIPTextModel.from_pretrained(CLIP_id)
7 self.device = "cuda" if torch.cuda.is_available() else "cpu"
8
9 for param in self.text_encoder.parameters():
10 param.requires_grad = False
11
12 self.text_encoder.eval()
13 self.text_encoder.to(self.device)
14
15 def forward(self, prompts):
16 inputs = self.tokenizer(
17 prompts,
18 padding="max_length",
19 truncation=True,
20 max_length=self.text_encoder.config.max_position_embeddings,
21 return_tensors="pt"
22)
23 input_ids = inputs.input_ids.to(self.device)
24 attention_mask = inputs.attention_mask.to(self.device)
25
26 with torch.no_grad():
27 text_encoder_output = self.text_encoder(
28 input_ids=input_ids,
29 attention_mask=attention_mask
30)
31 last_hidden_states = text_encoder_output.last_hidden_state
32
33 return last_hidden_states
```

## Khai báo mô hình pre-trained VAE

Để mã hóa ảnh đầu vào về dạng không gian tiềm ẩn và và giải mã không gian tiềm ẩn được tạo từ quá trình ngược của Diffusion thành ảnh, ta sẽ sử dụng mô hình Variational Autoencoder (VAE). Tại đây, ta tận dụng một pre-trained model có sẵn từ HuggingFace như sau:

```

1 VAE_id = "stabilityai/sd-vae-ft-mse"
2 vae = AutoencoderKL.from_pretrained(VAE_id)
3 vae.requires_grad_(False)
4 vae.eval()

```

## Khai báo hàm trực quan hóa ảnh và scale giá trị ảnh

```

1 def show_images(images, title="", titles=[]):
2 plt.figure(figsize=(8, 8))
3 for i in range(min(25, len(images))):
4 plt.subplot(5, 5, i+1)
5 img = images[i].permute(1, 2, 0).cpu().numpy()
6 plt.imshow(img)
7 if titles:
8 plt.title(titles[i])
9 plt.axis("off")
10 plt.suptitle(title)
11 plt.tight_layout()
12 plt.show()
13
14
15 def rescale(value, in_range, out_range, clamp=False):
16 in_min, in_max = in_range
17 out_min, out_max = out_range
18
19 in_span = in_max - in_min
20 out_span = out_max - out_min
21
22 scaled_value = (value - in_min) / (in_span + 1e-8)
23 rescaled_value = out_min + (scaled_value * out_span)
24
25 if clamp:
26 rescaled_value = torch.clamp(
27 rescaled_value,
28 out_min, out_max
29)

```

```

30
31 return rescaled_value

```

## Khai báo class PyTorch dataset

Ta xây dựng class PyTorch dataset cho bộ ảnh-mô tả về emoji đã thu thập được như sau:

```

1 WIDTH, HEIGHT = 32, 32
2 batch_size = 32
3
4 class EmojiDataset(Dataset):
5 def __init__(self, csv_files, image_folder, transform=None):
6 self.dataframe = pd.concat([pd.read_csv(csv_file) for csv_file in
7 csv_files])
8 self.images_folder = image_folder
9 self.dataframe["image_path"] = self.dataframe["file_name"].str.
10 replace("\\\", "/")
11 self.image_paths = self.dataframe["image_path"].tolist()
12 self.titles = self.dataframe["prompt"].tolist()
13 self.transform = transform
14
15 def __len__(self):
16 return len(self.dataframe)
17
18 def __getitem__(self, idx):
19 image_path = self.images_folder + "/" + self.image_paths[idx]
20 title = self.titles[idx]
21 title = title.replace('\"', "").replace('\'', "")
22 image = Image.open(image_path).convert('RGB')
23
24 if self.transform:
25 image = self.transform(image)
26
27 return image, title

```

Với class trên, ta khởi tạo DataLoader dùng để huấn luyện mô hình như sau (các bạn lưu ý thay đổi đường dẫn thư mục dataset cho phù hợp với máy của các bạn):

```

1 transform = transforms.Compose([
2 transforms.Resize(
3 (WIDTH, HEIGHT),
4 interpolation=transforms.InterpolationMode.BICUBIC

```

```

5),
6 transforms.ToTensor(),
7 transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
8])
9
10 csv_files = ["/content/blobs_crawled_data/metadata.csv"]
11 image_folder = "/content/blobs_crawled_data/images"
12
13 train_dataset = EmojiDataset(
14 csv_files=csv_files,
15 image_folder=image_folder,
16 transform=transform
17)
18 train_dataloader = DataLoader(
19 train_dataset,
20 batch_size=batch_size,
21 shuffle=True,
22 num_workers=2,
23 pin_memory=True,
24 persistent_workers=True
25)

```

## Thực hiện huấn luyện

Trước khi thực hiện huấn luyện, ta đóng gói phần cài đặt huấn luyện mô hình vào thành một hàm nhằm thuận tiện trong việc sử dụng code:

```

1 def train(diffusion, vae, text_encoder, scheduler,
2 optimizer, lr_scheduler, scaler,
3 criterion, dataloader, num_epochs, device="cuda"):
4 losses = []
5 for epoch in range(num_epochs):
6 diffusion.train()
7
8 epoch_loss = 0.0
9 progress_bar = tqdm(dataloader, desc=f"Epoch {epoch+1}/{num_epochs}"
10 , leave=False)
11
12 for batch_idx, (images, titles) in enumerate(progress_bar):
13 images = images.to(device)
14 image_titles = [f"A photo of {title}" for title in titles]
15 image_titles = [title if random.random() < 0.5 else "" for
title in image_titles]

```

```

16 with torch.no_grad():
17 latents = vae.encode(images).latent_dist.sample() * 0.
18 18215
19
20 timesteps = torch.randint(
21 0, scheduler.total_train_timesteps,
22 (latents.shape[0],), device=device
23)
24
25 noisy_latents, noise = scheduler.add_noise(latents, timesteps)
26 time_embeddings = embed_timesteps(timesteps).to(device)
27 text_embeddings = text_encoder(image_titles)
28
29 noise_pred = diffusion(noisy_latents, text_embeddings,
30 time_embeddings)
31
32 with autocast(device_type="cuda", dtype=torch.float16,
33 enabled=True, cache_enabled=True):
34 loss = criterion(noise_pred, noise)
35
36 optimizer.zero_grad()
37 scaler.scale(loss).backward()
38 scaler.step(optimizer)
39 scaler.update()
40
41 batch_loss = loss.item()
42 epoch_loss += batch_loss
43
44 progress_bar.set_postfix(loss=f"[batch_loss:.5f]",
45 lr=f"[optimizer.param_groups[0]['lr']:.6f]")
46
47 lr_scheduler.step()
48 avg_epoch_loss = epoch_loss / len(dataloader)
49 if (epoch + 1) % 10 == 0 or epoch == 0:
50 print(f"Epoch [{epoch+1}/{num_epochs}] - Avg Loss: {avg_epoch_loss:.5f}")
51 losses.append(avg_epoch_loss)
52
53 print("Training finished!")
54 return losses

```

Sau đó, ta khai báo các tham số, các mô hình thành phần và các hàm tham gia vào quá trình huấn luyện và thực hiện lời gọi hàm `train()` để tiến hành việc huấn luyện mô hình Stable Diffusion:

```

1 EPOCHS = 300
2
3 h_dim = 384
4 n_head = 8
5
6 vae = vae.to(device)
7 diffusion = Diffusion(h_dim, n_head).to(device)
8 clip = CLIPTextEncoder().to(device)
9
10 random_generator = torch.Generator(device="cuda")
11 noise_scheduler = DDPMscheduler(random_generator)
12
13 optimizer = torch.optim.AdamW(diffusion.parameters(), lr=1e-4)
14 criterion = torch.nn.MSELoss()
15
16 lrate_scheduler = lr_scheduler.CosineAnnealingLR(
17 optimizer, T_max=EPOCHS, eta_min=1e-5
18)
19 scaler = GradScaler()
20
21 def count_parameters(model):
22 return sum(p.numel() for p in model.parameters())
23
24 vae_params = count_parameters(vae)
25 diffusion_params = count_parameters(diffusion)
26 clip_params = count_parameters(clip)
27
28 print(f"VAE parameters: {vae_params}")
29 print(f"Diffusion parameters: {diffusion_params}")
30 print(f"CLIP parameters: {clip_params}")
31 print(f"Total parameters: {vae_params + diffusion_params + clip_params}")
32
33 losses = train(diffusion, vae, clip, noise_scheduler,
34 optimizer, lrate_scheduler, scaler,
35 criterion, train_dataloader, EPOCHS, device=device)

```

## Lưu trữ mô hình

Sau khi huấn luyện xong, việc lưu trữ mô hình cần được thực hiện để có thể tái sử dụng ở bất cứ đâu. Theo đó, đoạn code sau cần được thực thi:

```

1 torch.save(diffusion.state_dict(), "Emoji_SD.pth")
2 !zip -r Emoji_SD.zip Emoji_SD.pth

```

### 54.2.3 Triển khai mô hình

#### Xây dựng hàm tạo sinh ảnh

Với mô hình đã huấn luyện được ở phần trước, ta hoàn toàn có thể tạo một hàm thực hiện inference mô hình với một câu prompt bất kì nhằm lấy kết quả tạo sinh hình của mô hình như sau:

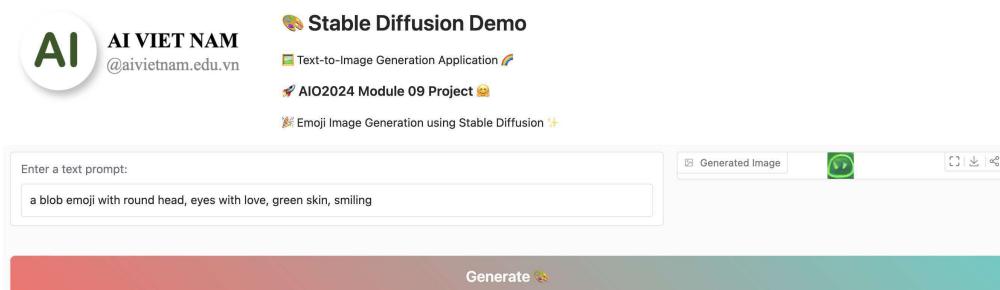
```
1 LATENTS_WIDTH = WIDTH // 8
2 LATENTS_HEIGHT = HEIGHT // 8
3
4 def generate_image(
5 prompt,
6 diffusion,
7 vae,
8 text_encoder,
9 scheduler,
10 num_inference_steps=100,
11 seed=None,
12 device="cuda" if torch.cuda.is_available() else "cpu",
13):
14 rng_generator = torch.Generator(device=device)
15 if seed is None:
16 rng_generator.seed()
17 else:
18 rng_generator.manual_seed(seed)
19
20 prompts = [prompt]
21 text_embeddings = text_encoder(prompts).to(device)
22
23 scheduler.set_steps(num_inference_steps)
24
25 latent_shape = (1, 4, LATENTS_HEIGHT, LATENTS_WIDTH)
26
27 noisy_latents = torch.randn(
28 latent_shape, generator=rng_generator, device=device)
29 timesteps = scheduler.schedule_timesteps
30
31 for t in tqdm(timesteps):
32 latent_model_input = noisy_latents
33
34 time_embedding = embed_a_timestep(t).to(device)
35
36 with torch.no_grad():
37 noise_pred = diffusion(
```

```

38 latent_model_input,
39 text_embeddings,
40 time_embedding
41)
42 noisy_latents = scheduler.step(
43 t,
44 noisy_latents,
45 noise_pred
46)
47
48 final_latents = noisy_latents / 0.18215
49
50 with torch.no_grad():
51 decoded_image_tensor = vae.decode(final_latents).sample
52
53 image_output = rescale(decoded_image_tensor, (-1, 1), (0, 255), clamp=
54 True)
55 image_output = image_output.permute(0, 2, 3, 1).to("cpu", torch.uint8)
56 .numpy()
57
58 return image_output[0]

```

Với hàm trên, ta hoàn toàn có thể ứng dụng vào một ứng dụng web demo về tạo sinh hình ảnh đơn giản để triển khai thành một chương trình hoàn chỉnh như ảnh sau (các bạn có thể trải nghiệm thử web demo tại phần 60.4).



Hình 54.18: Hình ảnh trang web demo cho ứng dụng sinh ảnh biểu tượng sử dụng Stable Diffusion.

### 54.3 Câu hỏi trắc nghiệm

1. Điểm yếu lớn nhất khiến Diffusion Models truyền thống khó ứng dụng trong thực tế là gì?
  - a) Chất lượng ảnh thấp.
  - b) Tốn kém chi phí lưu trữ.
  - c) Quá trình huấn luyện không ổn định.
  - d) Tốc độ sinh ảnh chậm.
2. Stable Diffusion thực hiện quá trình khuếch tán trong không gian nào để tăng hiệu quả tính toán?
  - a) Không gian ảnh RGB.
  - b) Không gian pixel grayscale.
  - c) Không gian tiềm ẩn (latent space).
  - d) Không gian vector hóa của prompt.
3. Trong hàm mất mát của Stable Diffusion, mô hình học để dự đoán:
  - a) Ảnh gốc  $x$ .
  - b) Vector tiềm ẩn  $z$ .
  - c) Điều kiện văn bản  $y$ .
  - d) Nhiều  $\epsilon$ .
4. Mô hình nào sau đây được dùng để mã hóa prompt văn bản trong Stable Diffusion?
  - a) BERT.
  - b) GPT-2.
  - c) CLIP Text Encoder.
  - d) ResNet.
5. Cross-attention trong UNet có vai trò:
  - a) Tăng khả năng mã hóa không gian ảnh.

- b) Truyền đặc trưng từ prompt văn bản vào quá trình sinh ảnh.
- c) Thay thế self-attention để tăng tốc độ.
- d) Giảm độ lệch thông tin giữa các lớp.
6. Lợi ích chính của việc sử dụng kiến trúc UNet trong quá trình khử nhiễu là gì?
- a) Kết hợp đặc trưng ở nhiều cấp độ khác nhau.
- b) Giảm số tham số mô hình.
- c) Tự động hóa quá trình huấn luyện.
- d) Truyền tải prompt qua latent vector.
7. Text embedding  $\tau_\theta(y)$  đóng vai trò gì trong quá trình sinh ảnh?
- a) Làm nhiều điều kiện để tăng đa dạng.
- b) Sinh ảnh trực tiếp không qua khử nhiễu.
- c) Điều kiện hoá quá trình sinh ảnh theo prompt.
- d) Loại bỏ thông tin nhiễu trong ảnh gốc.
8. Đoạn mã sau thực hiện chức năng gì trong pipeline Stable Diffusion?
- ```
1 text_input = tokenizer(prompt, padding="max_length", max_length=
                         tokenizer.model_max_length,
                         return_tensors="pt")
2 text_embeddings = text_encoder(text_input.input_ids.to(device))[0]
```
- a) Sinh ảnh đầu ra từ latent vector.
- b) Mã hóa văn bản thành vector đặc trưng.
- c) Tạo nhiễu khởi tạo.
- d) Tính toán tổn thất huấn luyện.
9. Trong đoạn mã dưới, vai trò của biến `latents` là gì?
- ```
1 latents = torch.randn(
2 (batch_size, unet.in_channels, height // 8, width // 8),
3 device=device
4)
```

- a) Biểu diễn ảnh RGB đầu ra.
  - b) Nhiều khởi tạo cho quá trình sinh ảnh.
  - c) Prompt đã mã hóa.
  - d) Kết quả đầu ra cuối cùng.
10. Tại sao diffusion trong latent space giúp giảm chi phí tính toán?
- a) Vì số bước sampling bị giảm.
  - b) Vì latent space là không gian nén, dữ liệu nhỏ hơn nhiều.
  - c) Vì VAE loại bỏ nhu cầu khử nhiễu.
  - d) Vì không cần huấn luyện lại từ đầu.

## 54.4 Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
2. **Hint:** Các file code gợi ý có thể được tải tại [đây](#).
3. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần project, ad mới copy các nội dung bài giải nêu trên vào đường dẫn).
4. **Demo:** Web demo và mã nguồn của ứng dụng có thể được truy cập tại [đây](#).
5. **Rubric:**

Mục	Kiến Thức	Đánh Giá
I.	<ul style="list-style-type: none"> <li>- Kiến thức về crawl data.</li> <li>- Kiến thức về thư viện Selenium để crawl dữ liệu.</li> </ul>	<ul style="list-style-type: none"> <li>- Nắm được cách sử dụng thư viện Selenium để lấy dữ liệu từ một trang web.</li> </ul>
II.	<ul style="list-style-type: none"> <li>- Các kiến thức cơ bản về bài toán Image Generation.</li> <li>- Các kiến thức cơ bản về mô hình Stable Diffusion.</li> <li>- Sử dụng thư viện PyTorch, Diffusers để triển khai mô hình Stable Diffusion và thực hiện huấn luyện trên bộ dữ liệu đã thu thập được.</li> </ul>	<ul style="list-style-type: none"> <li>- Hiểu được các khái niệm cơ bản về bài toán Image Generation.</li> <li>- Hiểu được các lý thuyết cơ bản trong mô hình Stable Diffusion.</li> <li>- Có khả năng sử dụng thư viện PyTorch và Diffusers để triển khai một mô hình Stable Diffusion nhằm thực hiện huấn luyện trên dữ liệu đã thu thập.</li> </ul>

- *Hết* -

# Chương 55

## Project 3: Sinh ảnh từ gợi ý (text) dùng Flow Matching

### 55.1 Giới thiệu



A photo of a chair swed in half



A photo of an open door

Hình 55.1: Text-Guided Image Generation.

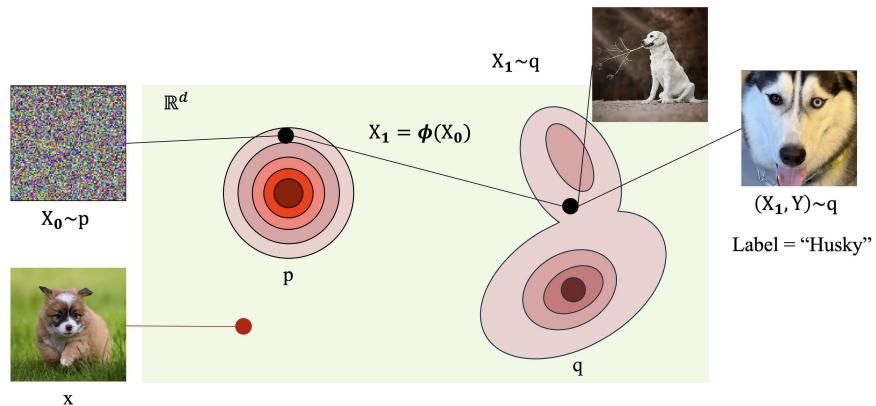
**Text-Guided Image Generation** là một biến thể nâng cao của bài toán Text-to-Image Generation, trong đó mô hình không chỉ tạo hình ảnh từ văn bản mà còn có khả năng chỉnh sửa hình ảnh có sẵn theo yêu cầu văn bản. Điều này có nghĩa là mô hình có thể hiểu và điều chỉnh hình ảnh dựa trên yêu cầu văn bản mà vẫn giữ nguyên những phần không bị ảnh hưởng.

Trong phần nội dung này, chúng ta sẽ ứng dụng mô hình Conditional Flow Matching vào giải quyết bài toán Text-to-Image Generation, sau đó mở rộng áp dụng cho bài toán Text-guided Image Generation.

Conditional Flow Matching (CFM) mở rộng Flow Matching với khả năng điều kiện hóa (conditioning). Điều này cho phép mô hình tạo ra dữ liệu có điều kiện dựa trên các thuộc tính hoặc thông tin bổ sung.

Trong mô hình có điều kiện, chúng ta làm việc với:

- $X_0 \sim p$ : dữ liệu thực
- $X_1 \sim q$ : noise

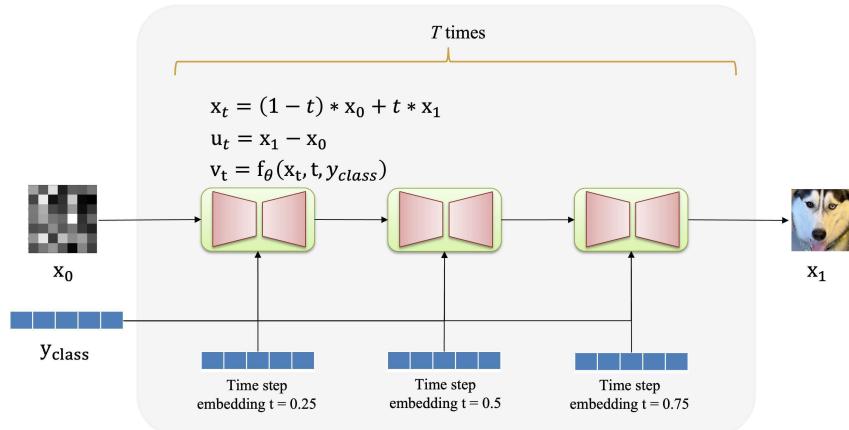


Hình 55.2: Conditional Flow Matching.

- $(X_1, Y) \sim q$ : cặp noise và thông tin điều kiện

Thông tin điều kiện  $Y$  có thể là các nhãn lớp, văn bản, hoặc các dữ liệu khác giúp hướng dẫn quá trình sinh.

### Huấn luyện Conditional Flow Matching



Hình 55.3: Flow Matching.

Quá trình huấn luyện cho Conditional Flow Matching tương tự như Flow Matching, nhưng bổ sung thông tin điều kiện vào mô hình học sâu để dự đoán các kết quả theo điều kiện:

- (a) Chọn ngẫu nhiên một mẫu có điều kiện  $(x_1, y_1) \sim q$
- (b) Chọn ngẫu nhiên một mẫu dữ liệu sạch  $x_0 \sim p_0$
- (c) Chọn ngẫu nhiên một thời điểm  $t \in [0, 1]$
- (d) Tính điểm nội suy  $x_t = (1 - t)x_0 + tx_1$
- (e) Tính vận tốc thực  $u_t = x_1 - x_0$
- (f) Dự đoán vận tốc bằng mô hình có điều kiện  $v_t = f_\theta(x_t, t, y_1)$
- (g) Tính hàm mất mát  $l_t = |v_t - u_t|^2$
- (h) Cập nhật tham số mô hình  $\theta$  bằng gradient descent

Hàm mất mát tổng quát trở thành:

$$\mathcal{L} = \mathbb{E}_{t, X_0, X_1, Y} |u_t^\theta(X_t, Y) - (X_1 - X_0)|^2$$

### Lấy mẫu từ Conditional Flow Matching

Quá trình lấy mẫu cũng tương tự nhưng sử dụng thông tin điều kiện:

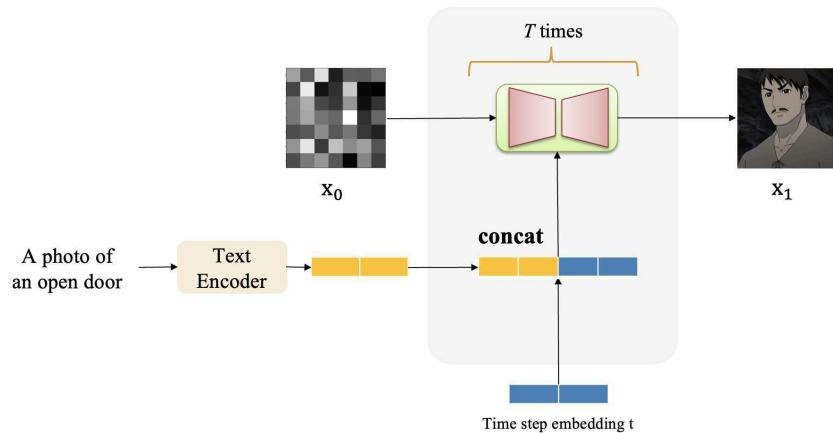
$$\frac{dy}{dt} = f_\theta(y, t, y_{class})$$

$$y(t + \Delta t) \approx y(t) + f_\theta(y, t, y_{class})\Delta t$$

Trong đó  $y_{class}$  là thông tin điều kiện (ví dụ: nhãn lớp) mà chúng ta muốn sử dụng để hướng dẫn quá trình sinh.

## 55.2 Text-to-Image using Conditional Flow Matching

Mô hình áp dụng CFM cho bài toán Text-to-Image được mô tả như hình sau:



Hình 55.4: Text-to-Image using CFM.

Quá trình huấn luyện mô hình như sau:

### 1. Dataset Preparing

Để huấn luyện mô hình, bộ dữ liệu được sử dụng cho các hình ảnh tương ứng và đoạn văn bản mô tả, bộ dữ liệu được sử dụng ở đây là: "Naruto Captions". Mô hình sentence transformers được sử dụng để mã hóa cho văn bản thành vector có 768 chiều.

```

1 # Install libs
2 !pip install -q datasets torchcfm
3
4 # Dataset
5 from datasets import load_dataset
6
7 ds = load_dataset("wanhin/naruto-captions", split="train")
8
9 # Text Encoder:
10 import torch

```

```
11 from sentence_transformers import SentenceTransformer
12
13 device = torch.device(
14 "cuda" if torch.cuda.is_available() else "cpu"
15)
16 text_encoder = SentenceTransformer("all-mpnet-base-v2").
 to(device)
```

## 2. Preprocessing

Xây dựng bộ dữ liệu cho quá trình huấn luyện mô hình, với dữ liệu đầu vào là text embedding, và đầu ra mô hình là biểu diễn của ảnh tương ứng.

```
1
2 from torchvision import transforms
3
4 transform = transforms.Compose([
5 transforms.Resize((64, 64)),
6 transforms.ToTensor()
7])
8
9 from torch.utils.data import Dataset, DataLoader
10
11 class CFMDataset(Dataset):
12 def __init__(self, dataset, transform, text_encoder,
13 device):
14 self.dataset = dataset
15 self.transform = transform
16 self.text_encoder = text_encoder
17 self.device = device
18 self.images = dataset["image"]
19 self.captions = dataset["text"]
20 self.embed_captions = text_encoder.encode(
21 self.captions, convert_to_tensor=True, device
22 =self.device
23)
24
25 def __len__(self):
26 return len(self.images)
27
28 def __getitem__(self, idx):
29 # get a image
30 image = self.images[idx]
31 image = self.transform(image)
```

```

31 # get a text
32 caption = self.captions[idx]
33 caption_embedding = self.embed_captions[idx]
34
35 return {
36 "image": image,
37 "caption": caption,
38 "caption_embedding": caption_embedding,
39 }
40
41 train_ds = CFMDataset(ds, transform, text_encoder, device
42)
42 train_loader = DataLoader(train_ds, batch_size=256,
43 shuffle=True)

```

### 3. Model

Mô hình kết hợp biểu diễn của văn bản vào với biểu diễn của time embedding. Tinh chỉnh UNet theo hướng dẫn sau:

```

1 import math
2 import torch.nn as nn
3 from torchcfm.models.unet import UNetModel
4
5 def timestep_embedding(timesteps, dim, max_period=10000):
6 """Create sinusoidal timestep embeddings.
7
8 :param timesteps: a 1-D Tensor of N indices, one per
9 batch element. These may be fractional.
10 :param dim: the dimension of the output.
11 :param max_period: controls the minimum frequency of
12 the embeddings.
13 :return: an [N x dim] Tensor of positional embeddings
14
15 """
16 half = dim // 2
17 freqs = torch.exp(
18 -math.log(max_period)
19 * torch.arange(start=0, end=half, dtype=torch.
20 float32, device=timesteps.device)
21 / half
22)
23 args = timesteps[:, None].float() * freqs[None]
24 embedding = torch.cat([torch.cos(args), torch.sin(
25 args)], dim=-1)
26 if dim % 2:
27

```

```
22 embedding = torch.cat([embedding, torch.
23 zeros_like(embedding[:, :, 1])], dim=-1)
24 return embedding
25
26 class UNetModelWithTextEmbedding(UNetModel):
27 def __init__(self, dim, num_channels, num_res_blocks,
28 embedding_dim, *args, **kwargs):
29 super().__init__(dim, num_channels,
30 num_res_blocks, *args, **kwargs)
31
32 self.embedding_layer = nn.Linear(embedding_dim,
33 num_channels*4)
34 self.fc = nn.Linear(num_channels*8, num_channels
35 *4)
36
37 def forward(self, t, x, text_embeddings=None):
38 """Apply the model to an input batch,
39 incorporating text embeddings."""
40 timesteps = t
41
42 while timesteps.dim() > 1:
43 timesteps = timesteps[:, 0]
44 if timesteps.dim() == 0:
45 timesteps = timesteps.repeat(x.shape[0])
46
47 hs = []
48 emb = self.time_embed(timestep_embedding(
49 timesteps, self.model_channels))
50
51 if text_embeddings is not None:
52 text_embedded = self.embedding_layer(
53 text_embeddings)
54 emb = torch.cat([emb, text_embedded], dim=1)
55 # 128*2
56 emb = self.fc(emb)
57
58 h = x.type(self.dtype)
59 for module in self.input_blocks:
60 h = module(h, emb)
61 hs.append(h)
62 h = self.middle_block(h, emb)
63 for module in self.output_blocks:
64 h = torch.cat([h, hs.pop()], dim=1)
65 h = module(h, emb)
66 h = h.type(x.dtype)
```

```
58 return self.out(h)
```

#### 4. Training

Huấn luyện mô hình dựa vào CFM dựa vào code sau:

```
1 from tqdm import tqdm
2 model = UNetModelWithTextEmbedding(
3 dim=(3, 64, 64), num_channels=32, num_res_blocks=1,
4 embedding_dim=768
5).to(device)
6 optimizer = torch.optim.Adam(model.parameters())
7
8 n_epochs = 20000
9
10 for epoch in tqdm(range(n_epochs)):
11 losses = []
12 for batch in train_loader:
13 optimizer.zero_grad()
14 x1 = batch["image"].to(device)
15 text_embeddings = batch["caption_embedding"].to(
16 device)
17 x0 = torch.randn_like(x1).to(device)
18 t = torch.rand(x0.shape[0], 1, 1, 1).to(device)
19 xt = t * x1 + (1 - t) * x0
20 ut = x1 - x0
21 t = t.squeeze()
22 vt = model(t, xt, text_embeddings=text_embeddings
23)
24 loss = torch.mean(((vt - ut) ** 2))
25 loss.backward()
26 optimizer.step()
27 losses.append(loss.item())
28
29 avg_loss = sum(losses) / len(losses)
30 if (epoch + 1) % 500 == 0:
31 print(f"Epoch [{epoch+1}/{n_epochs}], Loss: {avg_loss:.4f}")
```

#### 5. Inference

Sau khi huấn luyện, đoạn văn bản sau đây được sử dụng để giá mô hình: "A man with dark hair and brown eyes".

```
1 model.eval()
```

```
2 def euler_method(model, text_embedding, t_steps, dt,
3 noise):
4 y = noise
5 y_values = [y]
6 with torch.no_grad():
7 for t in t_steps[1:]:
8 t = t.reshape(-1,)
9 dy = model(t.to(device), y, text_embeddings=
text_embedding)
10 y = y + dy * dt
11 y_values.append(y)
12 return torch.stack(y_values)
13
14 # Initial random image and class (optional)
15 noise = torch.randn((3, 64, 64), device=device).unsqueeze
(0)
16 # A man with dark hair and brown eyes
17 text_embedding = text_embeddings[1].unsqueeze(0).to(
device)
18
19 # Time parameters
20 t_steps = torch.linspace(0, 1, 100, device=device)
21 dt = t_steps[1] - t_steps[0]
22
23 # Solve the ODE using Euler method
24 results = euler_method(model, text_embedding, t_steps, dt
, noise)
```

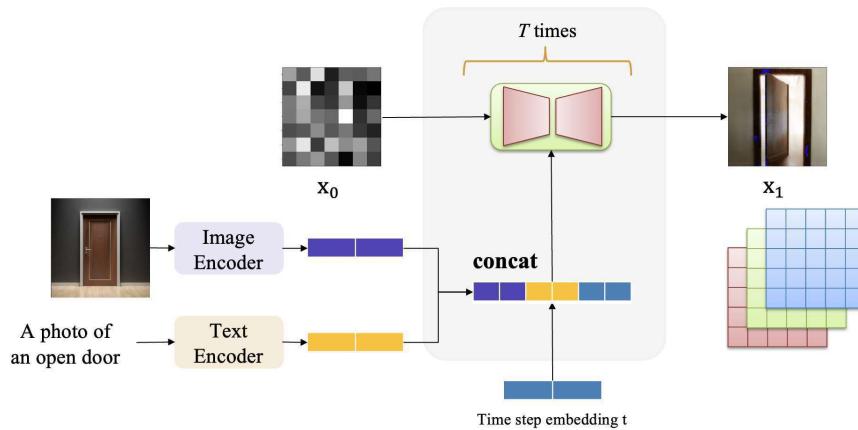
Kết quả sao sánh với ảnh thật:



Hình 55.5: Example of Text-to-Image Generation.

### 55.3 Text-Guided Image Generation using Conditional Flow Matching

Mô hình áp dụng CFM cho bài toán Text-Guided Image Generation được mô tả như hình sau:



Hình 55.6: Text-Guided Image Generation using CFM.

Quá trình huấn luyện mô hình như sau:

#### 1. Data Preparing

Để huấn luyện mô hình, bộ dữ liệu được sử dụng cho các hình ảnh tương ứng và đoạn văn bản mô tả, bộ dữ liệu được sử dụng ở đây là: "tedbench", mỗi mẫu dữ liệu gồm: ảnh gốc, đoạn văn bản chỉnh sửa và ảnh sau chỉnh sửa. Mô hình sentence transformers được sử dụng để mã hóa cho văn bản thành vector có 768 chiều.

```

1 # Install libs
2 !pip install -q datasets torchcfm
3
4 # Dataset
5 from datasets import load_dataset
6
7 ds = load_dataset("bahjat-kawar/tedbench", split="val")
8
9 # Text Encoder
10 import torch

```

```
11 from sentence_transformers import SentenceTransformer
12
13 device = torch.device("cuda" if torch.cuda.is_available()
14 else "cpu")
14 text_encoder = SentenceTransformer("all-mpnet-base-v2").
15 to(device)
```

## 2. Preprocessing

Xây dựng bộ dữ liệu cho quá trình huấn luyện mô hình, với dữ liệu đầu vào là text embedding và hình ảnh gốc, và đầu ra mô hình là biểu diễn của ảnh chỉnh sửa tương ứng.

```
1 from torchvision import transforms
2 from torch.utils.data import Dataset, DataLoader
3
4 class TextGuidedImageGenerationDataset(Dataset):
5 def __init__(self, dataset, transform, text_encoder,
6 device):
7 self.dataset = dataset
8 self.transform = transform
9 self.text_encoder = text_encoder
10 self.device = device
11 self.original_images = dataset["original_image"]
12 self.captions = dataset["caption"]
13 self.edited_images = dataset["edited_image"]
14 self.embed_captions = text_encoder.encode(
15 self.captions, convert_to_tensor=True, device
16 =self.device
17)
18
19 def __len__(self):
20 return len(self.captions)
21
22 def __getitem__(self, idx):
23 # get a image
24 original_image = self.original_images[idx]
25 original_image = self.transform(original_image)
26
27 edited_image = self.edited_images[idx]
28 edited_image = self.transform(edited_image)
29
30 # get a text
31 caption = self.captions[idx]
32 caption_embedding = self.embed_captions[idx]
```

```

32 return {
33 "original_image": original_image,
34 "edited_image": edited_image,
35 "caption": caption,
36 "caption_embedding": caption_embedding,
37 }
38
39 transform = transforms.Compose([
40 transforms.Resize((256, 256)),
41 transforms.ToTensor()
42])
43
44 train_ds = TextGuidedImageGenerationDataset(
45 ds, transform, text_encoder, device
46)
47 train_loader = DataLoader(train_ds, batch_size=256,
 shuffle=True)

```

### 3. Model

Mô hình kết hợp biểu diễn của văn bản và hình ảnh gốc vào với biểu diễn của time embedding. Tinh chỉnh UNet theo hướng dẫn sau:

```

1 import math
2 import torch.nn as nn
3 from torchcfm.models.unet import UNetModel
4
5 def timestep_embedding(timesteps, dim, max_period=10000):
6 """Create sinusoidal timestep embeddings.
7
8 :param timesteps: a 1-D Tensor of N indices, one per
9 batch element. These may be fractional.
10 :param dim: the dimension of the output.
11 :param max_period: controls the minimum frequency of
12 the embeddings.
13 :return: an [N x dim] Tensor of positional embeddings
14
15 """
16 half = dim // 2
17 freqs = torch.exp(
18 -math.log(max_period)
19 * torch.arange(start=0, end=half, dtype=torch.
20 float32, device=timesteps.device)
21 / half
22)
23 args = timesteps[:, None].float() * freqs[None]

```

```
20 embedding = torch.cat([torch.cos(args), torch.sin(
21 args)], dim=-1)
22 if dim % 2:
23 embedding = torch.cat([embedding, torch.
24 zeros_like(embedding[:, :, 1])], dim=-1)
25 return embedding
26
27 class UNetModelWithTextEmbedding(UNetModel):
28 def __init__(self, dim, num_channels, num_res_blocks,
29 embedding_dim, *args, **kwargs):
30 super().__init__(dim, num_channels,
31 num_res_blocks, *args, **kwargs)
32
33 self.image_encoder = nn.Sequential(
34 nn.Conv2d(3, num_channels, kernel_size=3,
35 stride=2, padding=1),
36 nn.ReLU(),
37 nn.Conv2d(num_channels, num_channels*2,
38 kernel_size=3, stride=2, padding=1),
39 nn.ReLU(),
40 nn.Conv2d(num_channels*2, num_channels*4,
41 kernel_size=3, stride=2, padding=1),
42 nn.ReLU(),
43 nn.AdaptiveAvgPool2d(1)
44)
45
46 self.embedding_layer = nn.Linear(embedding_dim,
47 num_channels*4)
48 self.fc = nn.Linear(num_channels*12, num_channels
49 *4)
50
51 def forward(self, t, x, text_embeddings=None,
52 original_image=None):
53 """Apply the model to an input batch,
54 incorporating text embeddings."""
55 timesteps = t
56
57 while timesteps.dim() > 1:
58 timesteps = timesteps[:, 0]
59 if timesteps.dim() == 0:
60 timesteps = timesteps.repeat(x.shape[0])
61
62 hs = []
63 emb = self.time_embed(timestep_embedding(
64 timesteps, self.model_channels))
```

```

53
54 if (text_embeddings is not None) and (
55 original_image is not None):
56 text_embedded = self.embedding_layer(
57 text_embeddings)
58 image_embedded = self.image_encoder(
59 original_image).squeeze(2, 3)
60 emb = torch.cat([emb, text_embedded,
61 image_embedded], dim=1)
62 emb = self.fc(emb)
63
64 h = x.type(self.dtype)
65 for module in self.input_blocks:
66 h = module(h, emb)
67 hs.append(h)
68 h = self.middle_block(h, emb)
69 for module in self.output_blocks:
70 h = torch.cat([h, hs.pop()], dim=1)
71 h = module(h, emb)
72 h = h.type(x.dtype)
73 return self.out(h)

```

#### 4. Training

Huấn luyện mô hình dựa vào CFM dựa vào code sau:

```

1 from tqdm import tqdm
2
3 model = UNetModelWithTextEmbedding(
4 dim=(3, 256, 256), num_channels=32, num_res_blocks=1,
5 embedding_dim=768
6).to(device)
7 optimizer = torch.optim.Adam(model.parameters())
8
9 n_epochs = 5000
10 for epoch in tqdm(range(n_epochs)):
11 losses = []
12 for batch in train_loader:
13 original_image = sample["original_image"].to(
14 device)
15 edited_image = sample["edited_image"].to(device)
16 caption_embedding = sample["caption_embedding"].
17 to(device)
18 optimizer.zero_grad()
19 x1 = edited_image
20 x0 = torch.randn_like(x1).to(device)

```

```

18 t = torch.rand(x0.shape[0], 1, 1, 1).to(device)
19 xt = t * x1 + (1 - t) * x0
20 ut = x1 - x0
21 t = t.squeeze()
22 vt = model(t, xt, text_embeddings=
23 caption_embedding, original_image=original_image)
24 loss = torch.mean(((vt - ut) ** 2))
25 loss.backward()
26 optimizer.step()
27
28 avg_loss = sum(losses)/len(losses)
29 if (epoch + 1) % 500 == 0:
30 print(f"Epoch [{epoch+1}/{n_epochs}], Loss: {avg_loss:.4f}")

```

## 5. Inference

Sau khi huấn luyện, đoạn văn bản sau đây được sử dụng để giá mô hình: "A photo of an open door" và hình ảnh gốc từ mẫu dữ liệu thứ 5 trong dữ liệu gốc.

```

1
2 model.eval()
3 def euler_method(model, text_embedding, t_steps, dt,
4 noise, original_image):
5 y = noise
6 y_values = [y]
7 with torch.no_grad():
8 for t in t_steps[1:]:
9 t = t.reshape(-1,)
10 dy = model(t.to(device), y, text_embeddings=
11 text_embedding, original_image=original_image)
12 y = y + dy * dt
13 y_values.append(y)
14 return torch.stack(y_values)
15
16 # Initial random image and class (optional)
17 # A photo of an open door.
18 sample = train_ds[5]
19 original_image = sample["original_image"].unsqueeze(0).to(
20 device)
21 edited_image = sample["edited_image"].unsqueeze(0).to(
22 device)
23 caption_embedding = sample["caption_embedding"].unsqueeze(
24 0).to(device)
25 noise = torch.randn_like(original_image, device=device)

```

```
21 text_embedding = caption_embedding
22
23 # Time parameters
24 t_steps = torch.linspace(0, 1, 150, device=device)
25 dt = t_steps[1] - t_steps[0]
26
27 # Solve the ODE using Euler method
28 results = euler_method(model, text_embedding, t_steps, dt
, noise, original_image)
```

## 6. Deployment

Triển khai mô hình trên streamlit. Tham khảo về [code](#) và [demo](#).

### Text-Guided Image Generation using Conditional Flow Matching

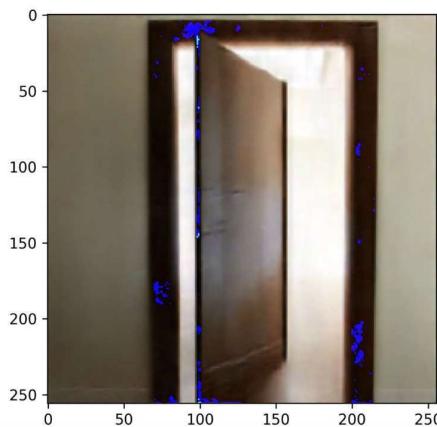
Model: Conditional Flow Matching. Dataset: Tedbench

Instruction:

A photo of an open door.

How would you like to give the input?

Run Example Image



Hình 55.7: Text-Guided Image Generation deployment on Streamlit.

## 55.4 Câu hỏi trắc nghiệm

**Câu hỏi 103** Phương pháp nào sau đây không được sử dụng trong Text-to-Image Generation?

- a) Flow Matching
- b) VAE
- c) K-means clustering
- d) Diffusion Models

**Câu hỏi 104** Phương pháp nào được sử dụng trong Text-to-Image để xử lý và mã hóa văn bản hiệu quả nhất?

- a) BERT
- b) LSTM
- c) CNN
- d) VAE

**Câu hỏi 105** Công thức nào sau đây mô tả sự thay đổi giữa dữ liệu nhiễu và dữ liệu sạch trong Flow Matching?

- a)  $v_t = x_1 - x_0$
- b)  $v_t = x_1 + x_0$
- c)  $v_t = x_1/x_0$
- d)  $v_t = \sigma(t) + x_0$

**Câu hỏi 106** Mỗi văn bản, đoạn text sẽ được biểu diễn thành vector bao nhiêu chiều?

- a) 512
- b) 768
- c) 256
- d) 128

**Câu hỏi 107** Đầu vào của mô hình UNet trong mô hình Text-to-Image Generation là bao nhiêu?

- a) 2
- b) 3
- c) 4
- d) 5

**Câu hỏi 108** Đầu vào của mô hình UNet trong mô hình Text-Guided Image Generation là bao nhiêu?

- a) 2
- b) 3
- c) 4
- d) 5

**Câu hỏi 109** Kích thước ảnh áp dụng cho mô hình Text-Guided Image Generation là bao nhiêu?

- a) 256x256
- b) 128x128
- c) 64x64
- d) 32x32

**Câu hỏi 110** Phương pháp biểu diễn cho time trong phần mô hình UNet là?

- a) RoPE
- b) Hàm sin cos
- c) ALiBi
- d) nn.Embedding

**Câu hỏi 111** Mô hình nào phù hợp để thay thế quá trình biểu diễn cho cả văn bản và hình ảnh gốc trong bài toán Text-Guided Image Generation?

- a) BERTs
- b) LSTM
- c) CLIP
- d) ResNet

**Câu hỏi 112** Công thức nào đúng trong phần code của Conditional Flow matching?

- a)  $y = y + dy/dt$
- b)  $y = y + dy * dt$
- c)  $y = y + dy + dt$
- d)  $y = y + dy - dt$

## 55.5 Phụ lục

1. **Hint:** Dựa vào file tải về [Text-Guided-Image-Generation-Conditional-Flow-Matching](#) để hoàn thiện các đoạn code.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, admin mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric:**

- *Hết* -

## **Phần XII**

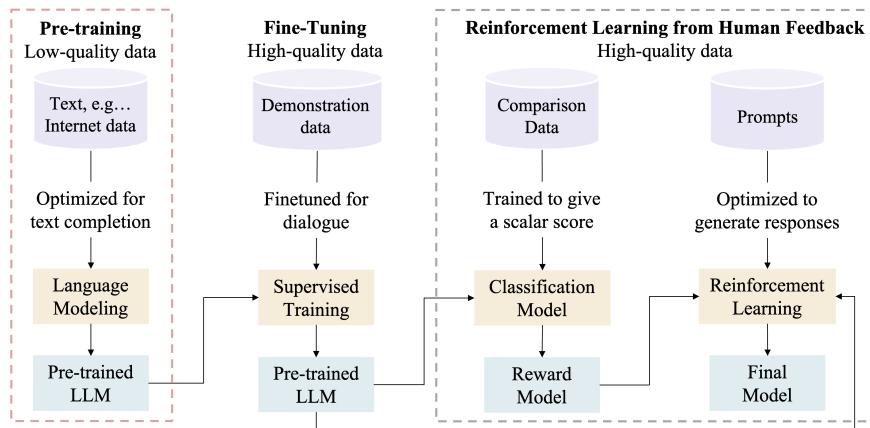
### **Module 12: LLMs và VLMs**

# Chương 56

## Pretraining GPT

### 56.1 Giới thiệu

Việc huấn luyện một mô hình ngôn ngữ lớn như GPT (điển hình là GPT-3, GPT-4, GPT-4o) thường bao gồm 3 giai đoạn chính:



Hình 56.1: GPT Training.

1. **Pre-training (Tiền huấn luyện):** - Giai đoạn mô hình học từ một lượng dữ liệu văn bản khổng lồ theo cách tự giám sát (self-supervised). - Mục tiêu: Dự đoán từ tiếp theo trong chuỗi văn bản.
2. **Supervised Fine-tuning (Tinh chỉnh có giám sát):** - Mô hình sau pre-training được tinh chỉnh thêm bằng cách sử dụng tập dữ liệu có nhãn với hướng dẫn rõ ràng (câu hỏi - câu trả lời). - Dữ liệu thường được thu thập và gắn nhãn bởi con người.
3. **Reinforcement Learning from Human Feedback (RLHF – Học tăng cường từ phản hồi con người):** - Mô hình được cải thiện thông qua phản hồi xếp hạng từ con người nhằm tối ưu hóa chất lượng phản hồi.

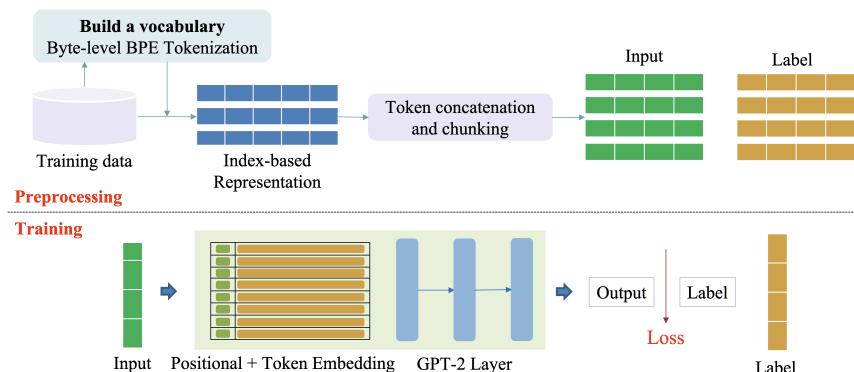
- Dùng thuật toán như Proximal Policy Optimization (PPO) để tinh chỉnh.

Để hiểu rõ về các quá trình huấn luyện mô hình, trong bài này, chúng ta sẽ tập trung phân tích nội dung ở bước đầu tiên **Pre-training**.

### Pre-training GPT-2

- Mục tiêu: Huấn luyện mô hình để học kiến thức ngôn ngữ nền tảng: từ vựng, cú pháp, ngữ nghĩa, logic và thậm chí cả thông tin thế giới.
- Dữ liệu: Nguồn dữ liệu lớn, không có nhãn: bao gồm sách, bài viết, mã nguồn, web crawl, v.v. Dữ liệu thường được xử lý qua các bước lọc: loại bỏ spam, văn bản lỗi, văn bản không tiếng Anh (nếu mô hình là English-based), v.v.
- Cách huấn luyện: Mô hình được huấn luyện bằng mục tiêu học tự giám sát. Mỗi mẫu đầu vào là một đoạn văn bản, và nhiệm vụ của mô hình là dự đoán từ tiếp theo.

Quá trình huấn luyện mô hình được mô tả như hình sau:



Hình 56.2: Pre-training GPT-2 Small.

Gồm 2 phần:

- Preprocessing (Tiền xử lý): Từ bộ dữ liệu như C4, xây dựng bộ từ điển dựa vào Byte-level BPE. Sau đó tiền hành biểu diễn sang vector chứa các giá trị index tương ứng trong bộ từ điển. Kết nối các vectors lại rồi phân chia thành các block để xây dựng giá trị I/O cho mô hình.

- (b) Training (Huấn luyện): Xây dựng mô hình GPT-2 Small. Huấn luyện mô hình dựa vào I/O và cập nhật trọng số mô hình thông qua hàm Cross-Entropy Loss

## 56.2 Câu hỏi trắc nghiệm

**Câu hỏi 113** GPT được huấn luyện bằng phương pháp học nào trong pre-training?

- a) Học có giám sát
- b) Học tăng cường
- c) Học không giám sát
- d) Học tự giám sát

**Câu hỏi 114** Mô hình GPT chỉ sử dụng kiến trúc nào?

- a) Encoder
- b) Decoder
- c) Encoder-Decoder
- d) RNN + Attention

**Câu hỏi 115** Byte-level BPE hoạt động trên đơn vị nào của văn bản?

- a) Từ (word)
- b) Ký tự (character)
- c) Byte
- d) Câu (sentence)

**Câu hỏi 116** Kỹ thuật Byte-level BPE giúp mô hình GPT xử lý tốt hơn điều gì?

- a) Cấu trúc ngữ pháp
- b) Dữ liệu hình ảnh
- c) Các từ chưa từng gặp (out-of-vocabulary)
- d) Phân loại ngữ nghĩa

**Câu hỏi 117** Mỗi văn bản, đoạn text sẽ được biểu diễn thành vector bao nhiêu chiều?

- a) 512
- b) 768
- c) 256
- d) 128

**Câu hỏi 118** Trong pretraining GPT, đầu vào chuỗi token sau khi mã hoá

sẽ đi qua thành phần nào đầu tiên?

- a) Linear layer
- b) Normalization
- c) Attention Head
- d) Embedding Layer

**Câu hỏi 119** Hàm mất mát chính được sử dụng trong pre-training của GPT là gì?

- a) Mean Squared Error
- b) Cross-entropy
- c) Hinge loss
- d) Contrastive loss

**Câu hỏi 120** Loại attention nào được sử dụng trong GPT?

- a) Bidirectional attention
- b) Encoder-Decoder attention
- c) Causal (masked) attention
- d) Cross-attention

**Câu hỏi 121** Mục tiêu của hàm loss trong pre-training GPT là gì?

- a) Tối đa hóa entropy
- b) Dự đoán token hiện tại
- c) Tối đa hóa xác suất token đúng
- d) Dự đoán vector nhúng tiếp theo

**Câu hỏi 122** Một batch training thường chứa các thành phần nào?

- a) Hình ảnh và mô tả
- b) Chuỗi token, attention mask, labels
- c) ID lớp và nhãn
- d) Attention mask và labels

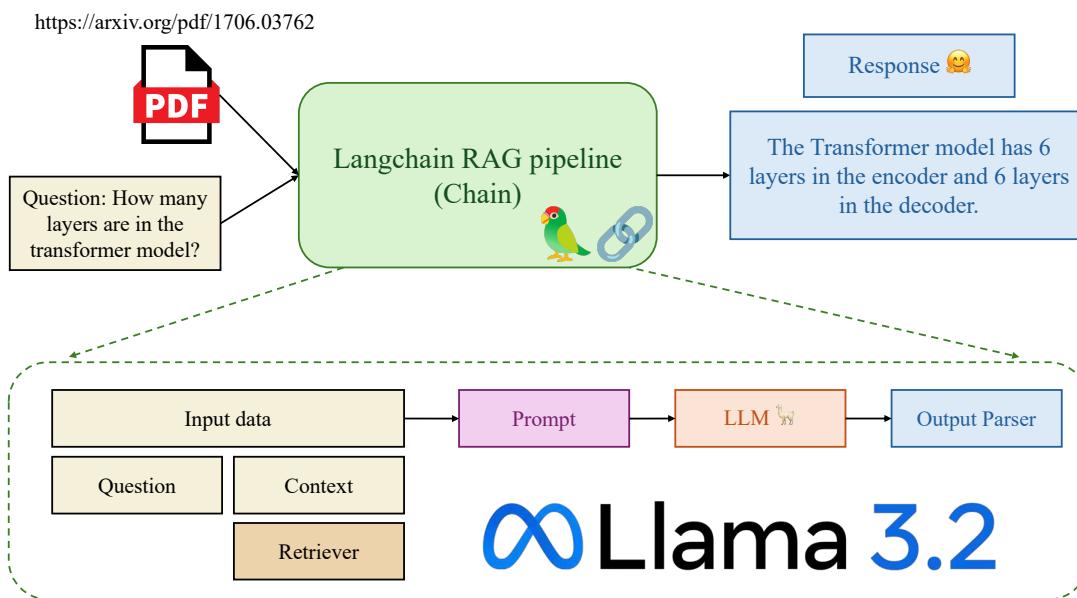
- *Hết* -

# Chương 57

## RAG, Prompting và LangChain

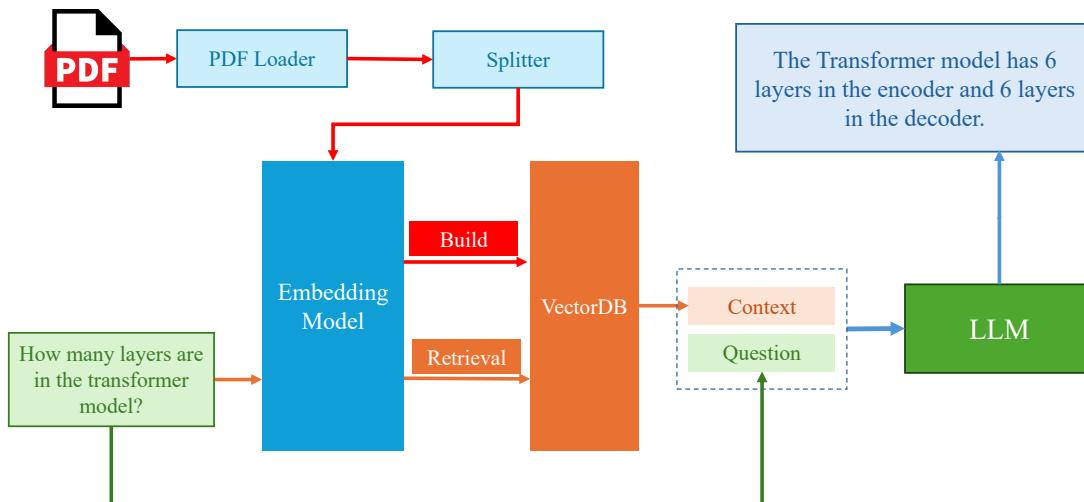
### 57.1 Giới thiệu

**LangChain** là một framework được thiết kế chuyên biệt cho việc triển khai LLMs trong các ứng dụng thực tế. LangChain hỗ trợ các công cụ và thư viện mạnh mẽ cho phép các nhà phát triển dễ dàng tích hợp các mô hình ngôn ngữ lớn với các ứng dụng của họ, từ các Chatbot thông minh cho đến các hệ thống phân tích dữ liệu phức tạp.



Hình 57.1: Minh họa ứng dụng hỏi đáp nội dung file pdf sử dụng LangChain.

Trong bài viết này, chúng ta sẽ xây dựng một ứng dụng về RAG (Retrieval Augmented Generation) trả lời các câu hỏi học thuật tận dụng nguồn tài liệu là các bài báo khoa học mà ta thu thập được (dưới dạng file pdf), sử dụng thư viện LangChain. Tổng quan, pipeline của project như sau:



Hình 57.2: Tổng quan về pipeline của project.

**Theo đó:**

1. Từ danh sách các bài báo khoa học, ta tách thành các văn bản nhỏ. Từ đó, xây dựng một hệ cơ sở dữ liệu vector với một embedding model.
2. Bên cạnh câu hỏi đầu vào (question), ta truy vấn các mẫu văn bản có liên quan đến câu hỏi, dùng làm ngữ cảnh (context) trong câu prompt. Đây là nguồn thông tin mà LLMs có thể dựa vào để trả lời câu hỏi.
3. Đưa câu prompt vào mô hình (question và context) để nhận câu trả lời từ mô hình.

## 57.2 Cài đặt chương trình

Trong phần này, chúng ta sẽ tiến hành cài đặt nội dung của project. Mã nguồn được xây dựng trên hệ điều hành Ubuntu với GPU 24GB. Các bước thực hiện như sau:

### 57.2.1 Tổ chức thư mục code

Để mã nguồn trở nên rõ ràng nhằm phục vụ cho mục đích đọc hiểu code, chúng ta sẽ tổ chức thư mục như sau:

```
rag_langchain/
 └── data_source/
 └── generative_ai/
 └── download.py

 └── src/
 └── base/
 └── llm_model.py

 └── rag/
 ├── file_loader.py
 ├── main.py
 ├── offline_rag.py
 ├── utils.py
 └── vectorstore.py

 └── app.py

 └── requirements.txt
```

Tổng quan, chúng ta sẽ có thư mục chứa mã nguồn có tên **rag\_langchain**

(các bạn hoàn toàn có thể sử dụng tên gọi khác). Bên trong sẽ có các thư mục con và các file với ý nghĩa như sau:

- **data\_source/**: Thư mục dùng để lưu trữ các tài liệu phục vụ cho việc xây dựng hệ cơ sở dữ liệu vector.
- **data\_source/generative\_ai/download.py**: File code dùng để tải tự động một số các bài báo khoa học dưới dạng file pdf.
- **src/base/llm\_model**: File code dùng để khai báo hàm khởi tạo mô hình ngôn ngữ lớn.
- **src/rag/**: Thư mục dùng để lưu trữ các code liên quan đến xây dựng RAG, bao gồm:
  1. **src/rag/file\_loader.py**: File code dùng để khai báo các hàm load file pdf (vì tài liệu của chúng ta thu thập thuộc file pdf).
  2. **src/rag/main.py**: File code dùng để khai báo hàm khởi tạo chains.
  3. **src/rag/offline\_rag.py**: File code dùng để khai báo PromptTemplate.
  4. **src/rag/utils.py**: File code dùng để khai báo hàm tách câu trả lời từ model.
  5. **src/rag/vectorstore.py**: File code dùng để khai báo hàm khởi tạo hệ cơ sở dữ liệu vector.
- **src/app.py**: File code dùng để khởi tạo API.
- **requirements.txt**: File code dùng để khai báo các thư viện cần thiết để sử dụng source code.

### 57.2.2 Cập nhật file requirements.txt

Để bắt đầu, chúng ta sẽ liệt kê các gói thư viện cần thiết để chạy được chương trình này. Các bạn hãy cập nhật file requirements.txt với nội dung sau:

```
1 torch==2.2.2
2 transformers==4.39.3
3 accelerate==0.28.0
4 bitsandbytes==0.42.0
5 huggingface-hub==0.22.2
```

```

6 langchain==0.1.14
7 langchain-core==0.1.43
8 langchain-community==0.0.31
9 pypdf==4.2.0
10 sentence-transformers==2.6.1
11 beautifulsoup4==4.12.3
12 langserve[all]
13 chromadb==0.4.24
14 langchain-chroma==0.1.0
15 faiss-cpu==1.8.0
16 rapidocr-onnxruntime==1.3.16
17 unstructured==0.13.2
18 fastapi==0.110.1
19 uvicorn==0.29.0

```

### 57.2.3 Cập nhật file `data_source/generative_ai/download.py`

Để tải một vài bài báo khoa học làm dữ liệu cho hệ cơ sở dữ liệu vector, chúng ta sẽ xây dựng một đoạn code tải tự động các bài báo. Nội dung như sau:

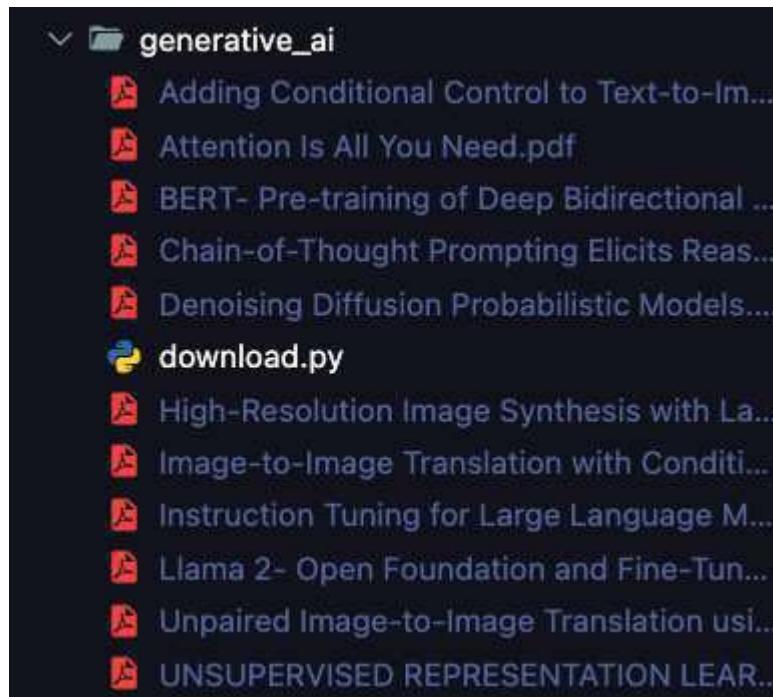
```

1 import os
2 import wget
3
4 file_links = [
5 {
6 "title": "Attention Is All You Need",
7 "url": "https://arxiv.org/pdf/1706.03762"
8 },
9 {
10 "title": "BERT- Pre-training of Deep Bidirectional Transformers
11 for Language Understanding",
12 "url": "https://arxiv.org/pdf/1810.04805"
13 },
14 {
15 "title": "Chain-of-Thought Prompting Elicits Reasoning in Large
16 Language Models",
17 "url": "https://arxiv.org/pdf/2201.11903"
18 },
19 {
20 "title": "Denoising Diffusion Probabilistic Models",
21 "url": "https://arxiv.org/pdf/2006.11239"
22 }
23]

```

```
20 },
21 {
22 "title": "Instruction Tuning for Large Language Models- A Survey",
23 "url": "https://arxiv.org/pdf/2308.10792"
24 },
25 {
26 "title": "Llama 2- Open Foundation and Fine-Tuned Chat Models",
27 "url": "https://arxiv.org/pdf/2307.09288"
28 }
29]
30
31 def is_exist(file_link):
32 return os.path.exists(f"./{file_link['title']}.pdf")
33
34 for file_link in file_links:
35 if not is_exist(file_link):
36 wget.download(file_link["url"], out=f"./{file_link['title']}.pdf")
```

Trong file code trên, chúng ta cung cấp một list các đường dẫn bài báo. Từ đó, sử dụng `wget` để tải về. Các bài báo sẽ được lưu ngay tại vị trí của file code. Vì mục đích demo, chúng ta sẽ chỉ tải một số lượng nhỏ các paper. Các bạn có thể tự thêm vào nhiều paper khác để test.



Hình 57.3: Minh họa danh sách các file bài báo khoa học sau khi được tải về.

#### 57.2.4 Cập nhật file `src/base/llm_model.py`

Tại file này, ta khai báo hàm `get_llm()`, dùng để thực hiện tải và gọi pre-trained LLM từ HuggingFace về máy. Đồng thời, ta áp dụng kỹ thuật quantization lên model để thực hiện inference trên GPU thấp. Nội dung file như sau:

```
1 import torch
2 from transformers import BitsAndBytesConfig
3 from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
4 from langchain.llms.huggingface_pipeline import HuggingFacePipeline
5
6
7 nf4_config = BitsAndBytesConfig(
8 load_in_4bit=True,
9 bnb_4bit_quant_type="nf4",
10 bnb_4bit_use_double_quant=True,
11 bnb_4bit_compute_dtype=torch.bfloat16
12)
13
```

```

14 def get_hf_llm(model_name: str = "meta-llama/Llama-3.2-3B-Instruct",
15 max_new_token = 1024,
16 **kwargs):
17
18 model = AutoModelForCausalLM.from_pretrained(
19 model_name,
20 quantization_config=nf4_config,
21 low_cpu_mem_usage=True
22)
23 tokenizer = AutoTokenizer.from_pretrained(model_name)
24
25 model_pipeline = pipeline(
26 "text-generation",
27 model=model,
28 tokenizer=tokenizer,
29 max_new_tokens=max_new_token,
30 pad_token_id=tokenizer.eos_token_id,
31 device_map="auto"
32)
33
34 llm = HuggingFacePipeline(
35 pipeline=model_pipeline,
36 model_kwargs=kwargs
37)
38
39 return llm

```

Trong project này, mô hình LLM mà chúng ta sử dụng là mô hình [Llama 3.2](#) 3B được huấn luyện trên dữ liệu instruction. Các bạn có thể thay thế bằng mô hình khác có cấu hình tương tự.

### 57.2.5 Cập nhật file `src/rag/file_loader.py`

```

1 from typing import Union, List, Literal
2 import glob
3 from tqdm import tqdm
4 import multiprocessing
5 from langchain_community.document_loaders import PyPDFLoader
6 from langchain_text_splitters import RecursiveCharacterTextSplitter
7
8 def remove_non_utf8_characters(text):
9 return ''.join(char for char in text if ord(char) < 128)
10
11 def load_pdf(pdf_file):

```

```
12 docs = PyPDFLoader(pdf_file, extract_images=True).load()
13 for doc in docs:
14 doc.page_content = remove_non_utf8_characters(doc.page_content)
15 return docs
16
17 def get_num_cpu():
18 return multiprocessing.cpu_count()
19
20 class BaseLoader:
21 def __init__(self) -> None:
22 self.num_processes = get_num_cpu()
23
24 def __call__(self, files: List[str], **kwargs):
25 pass
26
27 class PDFLoader(BaseLoader):
28 def __init__(self) -> None:
29 super().__init__()
30
31 def __call__(self, pdf_files: List[str], **kwargs):
32 num_processes = min(self.num_processes, kwargs["workers"])
33 with multiprocessing.Pool(processes=num_processes) as pool:
34 doc_loaded = []
35 total_files = len(pdf_files)
36 with tqdm(total=total_files, desc="Loading PDFs", unit="file")
37 as pbar:
38 for result in pool imap_unordered(load_pdf, pdf_files):
39 doc_loaded.extend(result)
40 pbar.update(1)
41 return doc_loaded
42
43 class TextSplitter:
44 def __init__(self,
45 separators: List[str] = ['\n\n', '\n', ' ', ''],
46 chunk_size: int = 300,
47 chunk_overlap: int = 0
48) -> None:
49
50 self.splitter = RecursiveCharacterTextSplitter(
51 separators=separators,
52 chunk_size=chunk_size,
53 chunk_overlap=chunk_overlap,
54)
55 def __call__(self, documents):
56 return self.splitter.split_documents(documents)
```

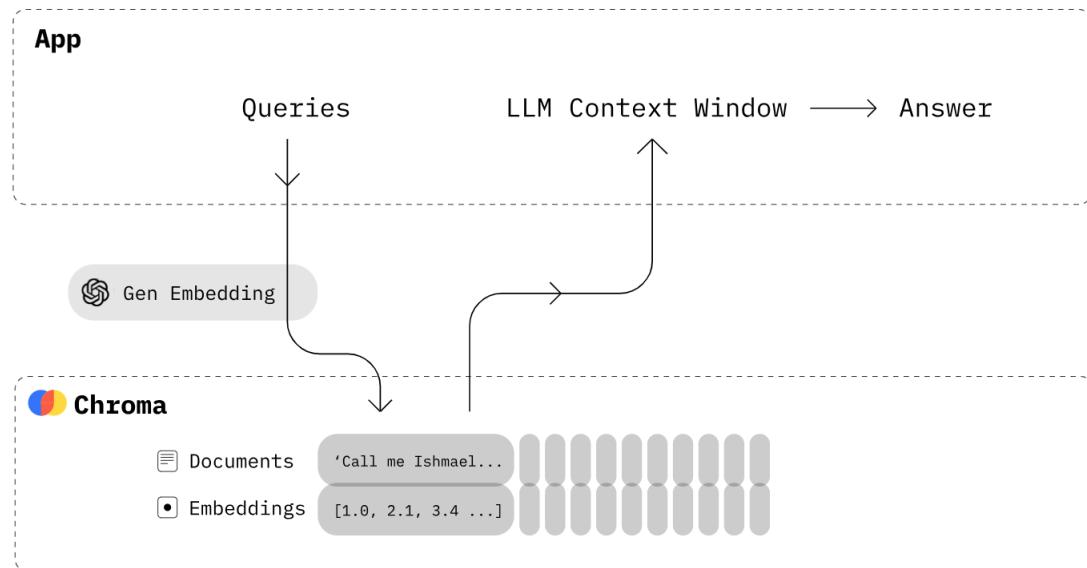
```

56
57 class Loader:
58 def __init__(self,
59 file_type: str = Literal["pdf"],
60 split_kwargs: dict = {
61 "chunk_size": 300,
62 "chunk_overlap": 0}
63) -> None:
64 assert file_type in ["pdf"], "file_type must be pdf"
65 self.file_type = file_type
66 if file_type == "pdf":
67 self.doc_loader = PDFLoader()
68 else:
69 raise ValueError("file_type must be pdf")
70
71 self.doc_splitter = TextSplitter(**split_kwargs)
72
73 def load(self, pdf_files: Union[str, List[str]], workers: int = 1):
74 if isinstance(pdf_files, str):
75 pdf_files = [pdf_files]
76 doc_loaded = self.doc_loader(pdf_files, workers=workers)
77 doc_split = self.doc_splitter(doc_loaded)
78 return doc_split
79
80 def load_dir(self, dir_path: str, workers: int = 1):
81 if self.file_type == "pdf":
82 files = glob.glob(f"{dir_path}/*.pdf")
83 assert len(files) > 0, f"No {self.file_type} files found in {dir_path}"
84 else:
85 raise ValueError("file_type must be pdf")
86 return self.load(files, workers)

```

### 57.2.6 Cập nhật file `src/rag/vectorstore.py`

Tại file này, ta định nghĩa một class để khởi tạo hệ cơ sở dữ liệu vector. Trong project này, chúng ta sẽ sử dụng Chroma. Về việc tìm kiếm tài liệu tương đồng, ta sử dụng FAISS. Như vậy, nội dung của file như sau:



Hình 57.4: Minh họa việc sử dụng vector database Chroma để truy vấn các tài liệu có liên quan làm context trong prompt. Ảnh: [Link](#).

```

1 from typing import Union
2 from langchain_chroma import Chroma
3 from langchain_community.vectorstores import FAISS
4 from langchain_community.embeddings import HuggingFaceEmbeddings
5
6 class VectorDB:
7 def __init__(self,
8 documents = None,
9 vector_db: Union[Chroma, FAISS] = Chroma,
10 embedding = HuggingFaceEmbeddings(),
11) -> None:
12
13 self.vector_db = vector_db
14 self.embedding = embedding
15 self.db = self._build_db(documents)
16
17 def _build_db(self, documents):
18 db = self.vector_db.from_documents(documents=documents,
19 embedding=self.embedding)
20
21 return db

```

```

22 def get_retriever(self,
23 search_type: str = "similarity",
24 search_kwargs: dict = {"k": 10}
25):
26 retriever = self.db.as_retriever(search_type=search_type,
27 search_kwargs=search_kwargs)
28 return retriever

```

### 57.2.7 Cập nhật file `src/rag/offline_rag.py`

Tại file này, ta khai báo class `Offline_RAG` để xây dựng một chain về RAG, bao gồm việc sử dụng retriever lấy context, xây dựng prompt và đưa vào model. Nội dung của file như sau:

```

1 import re
2 from langchain import hub
3 from langchain_core.runnables import RunnablePassthrough
4 from langchain_core.output_parsers import StrOutputParser
5
6 class Str_OutputParser(StrOutputParser):
7 def __init__(self) -> None:
8 super().__init__()
9
10 def parse(self, text: str) -> str:
11 return self.extract_answer(text)
12
13 def extract_answer(self,
14 text_response: str,
15 pattern: str = r"Answer:\s*(.*)"
16) -> str:
17
18 match = re.search(pattern, text_response, re.DOTALL)
19 if match:
20 answer_text = match.group(1).strip()
21 return answer_text
22 else:
23 return text_response
24
25 class Offline_RAG:
26 def __init__(self, llm) -> None:
27 self.llm = llm
28 self.prompt = hub.pull("rlm/rag-prompt")
29 self.str_parser = Str_OutputParser()
30

```

```

31 def get_chain(self, retriever):
32 input_data = {
33 "context": retriever | self.format_docs,
34 "question": RunnablePassthrough()
35 }
36 rag_chain = (
37 input_data
38 | self.prompt
39 | self.llm
40 | self.str_parser
41)
42 return rag_chain
43
44 def format_docs(self, docs):
45 return "\n\n".join(doc.page_content for doc in docs)

```

### 57.2.8 Cập nhật file `src/rag/utils.py`

Tại file này, ta khai báo hàm tách phần trả lời của model từ câu prompt (phần bắt đầu từ “Answer:”):

```

1 import re
2
3 def extract_answer(text_response: str,
4 pattern: str = r"Answer:\s*(.*)"
5) -> str:
6
7 match = re.search(pattern, text_response)
8 if match:
9 answer_text = match.group(1).strip()
10 return answer_text
11 else:
12 return "Answer not found."

```

### 57.2.9 Cập nhật file `src/rag/main.py`

Tại file này, ta khởi tạo toàn bộ các instance của các class, các hàm mà ta đã khai báo trước đó và kết nối chúng vào trong một hàm duy nhất gọi là `build_rag_chain()`:

```

1 from pydantic import BaseModel, Field
2

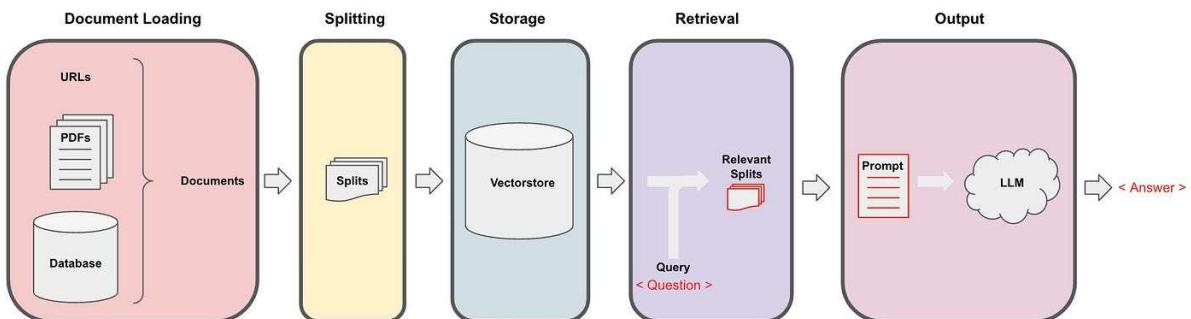
```

```

3 from src.rag.file_loader import Loader
4 from src.rag.vectorstore import VectorDB
5 from src.rag.offline_rag import Offline_RAG
6
7 class InputQA(BaseModel):
8 question: str = Field(..., title="Question to ask the model")
9
10 class OutputQA(BaseModel):
11 answer: str = Field(..., title="Answer from the model")
12
13 def build_rag_chain(llm, data_dir, data_type):
14 doc_loaded = Loader(file_type=data_type).load_dir(data_dir, workers=2)
15 retriever = VectorDB(documents = doc_loaded).get_retriever()
16 rag_chain = Offline_RAG(llm).get_chain(retriever)
17
18 return rag_chain

```

Như vậy, ta đã hoàn thiện toàn bộ các code cần thiết để xây dựng một ứng dụng về RAG. Để tổng quát hóa toàn bộ quy trình, chúng ta có thể tham khảo qua ảnh sau:



Hình 57.5: Minh họa chuỗi (chain) các bước xây dựng RAG trong LangChain.  
Ảnh: [Link](#).

### 57.2.10 Cập nhật file `src/app.py`

Cuối cùng, ta tạo file dùng để khai báo API với LangServe để triển khai ứng dụng RAG. Đối với LangServe, cách sử dụng gần như tương tự với việc sử dụng FastAPI. Nội dung file code như sau:

```

1 import os

```

```
2 os.environ["TOKENIZERS_PARALLELISM"] = "false"
3
4 from fastapi import FastAPI
5 from fastapi.middleware.cors import CORSMiddleware
6
7 from langserve import add_routes
8
9 from src.base.llm_model import get_hf_llm
10 from src.rag.main import build_rag_chain, InputQA, OutputQA
11
12 llm = get_hf_llm(temperature=0.9)
13 genai_docs = "./data_source/generative_ai"
14
15 # ----- Chains-----
16
17 genai_chain = build_rag_chain(llm, data_dir=genai_docs, data_type="pdf")
18
19 # ----- App - FastAPI -----
20
21 app = FastAPI(
22 title="LangChain Server",
23 version="1.0",
24 description="A simple api server using Langchain's Runnable interfaces",
25)
26
27 app.add_middleware(
28 CORSMiddleware,
29 allow_origins=["*"],
30 allow_credentials=True,
31 allow_methods=["*"],
32 allow_headers=["*"],
33 expose_headers=["*"],
34)
35
36 # ----- Routes - FastAPI -----
37
38 @app.get("/check")
39 async def check():
40 return {"status": "ok"}
41
42 @app.post("/generative_ai", response_model=OutputQA)
43 async def generative_ai(inputs: InputQA):
44 answer = genai_chain.invoke(inputs.question)
45 return {"answer": answer}
```

```

46
47 # ----- Langserve Routes - Playground -----
48 add_routes(app,
49 genai_chain,
50 playground_type="default",
51 path="/generative_ai")

```

Để khởi động API, chúng ta duy chuyển đến thư mục root của source code trong terminal (trong trường hợp của bài viết sẽ là thư mục `rag_langchain/`), sử dụng lệnh sau (sau khi đã cài đặt các thư viện cần thiết cũng như vector database). Lưu ý, nếu bị lỗi do port đã được sử dụng trong máy của bạn thì có thể thay đổi sang một port khác:

```
1 uvicorn src.app:app --host "0.0.0.0" --port 5000 --reload
```

The screenshot shows the LangChain Server API documentation. At the top, it says "LangChain Server 1.0 OAS 3.1 /openapi.json". Below that, it says "A simple api server using Langchain's Runnable interfaces".

**generative\_ai** (Using pydantic 2.7.0. OpenAPI docs for `invoke`, `batch`, `stream`, `stream_log` endpoints will not be generated. API endpoints and playground should work as expected. If you need to see the docs, you can downgrade to pydantic 1. For example, `pip install pydantic==1.10.13` See <https://github.com/liaogolo/fastapi/issues/10360> for details.)

- GET /generative\_ai/input\_schema Generative Ai Input Schema
- GET /generative\_ai/output\_schema Generative Ai Output Schema
- GET /generative\_ai/config\_schema Generative Ai Config Schema

**generative\_ai/config** Endpoints with a default configuration set by `config_hash` path parameter. Used in conjunction with share links generated using the LangServe UI playground. The hash is an LZString compressed JSON string.

- GET /generative\_ai/c/{config\_hash}/input\_schema Generative Ai Input Schema With Config
- GET /generative\_ai/c/{config\_hash}/output\_schema Generative Ai Output Schema With Config
- GET /generative\_ai/c/{config\_hash}/config\_schema Generative Ai Config Schema With Config

**default**

- GET /check Check
- POST /generative\_ai Generative Ai
- POST /generative\_ai/token\_feedback Create Feedback From Token

Hình 57.6: Minh họa API sau khi ta triển khai thành công.

```
Curl
curl -X 'POST' \
 'http://0.0.0:5050/generative_ai' \
 -H 'accept: application/json' \
 -H 'Content-Type: application/json' \
 -d '{
 "question": "What is instruction tuning in LLMs?"
}'
Request URL
http://0.0.0:5050/generative_ai
Server response
Code Details
200 Response body
{
 "answer": "Instruction tuning is a technique used to enhance the capabilities and controllability of large language models (LLMs) by fine-tuning them on a dataset consisting of instruction-output pairs in a supervised manner. This process bridges the gap between the next-word prediction objective of LLMs and the user's objective of instruction following. It allows for more controllable and predictable model behavior and provides a channel for humans to intervene with the model's behaviors. Instruction tuning is also computationally efficient and can help LLMs rapidly adapt to a specific domain without extensive retraining or architectural changes. However, crafting high-quality instructions that properly cover the desired target behaviors is a challenge."
}
 Download
```

Hình 57.7: Minh họa một kết quả của model thông qua API mà chúng ta đã xây dựng.

### 57.3 Câu hỏi trắc nghiệm

1. LangChain được sử dụng nhằm mục đích gì?
  - a) Web Scraping.
  - b) Model Quantization.
  - c) Building language model-powered applications.
  - d) Database Management.
2. Nội dung nào dưới đây là một thành phần cốt lõi của LangChain?
  - a) Transformers.
  - b) Agents.
  - c) Callbacks.
  - d) Hooks.
3. Trong LangChain, mục đích trong việc sử dụng PromptTemplate là?
  - a) Tạo các trường thông tin trong hệ cơ sở dữ liệu lưu thông tin người dùng.
  - b) Định nghĩa các tính năng trong giao diện của người dùng.
  - c) Tối ưu tốc độ xử lý của mô hình.
  - d) Chuẩn hóa một cấu trúc phản hồi nhất quán từ mô hình.
4. Xét đoạn code dưới đây:

```
1 from langchain_openai import ChatOpenAI
2 from langchain_openai import OpenAI
3
4 llm = OpenAI()
5 chat_model = ChatOpenAI(model="gpt-3.5-turbo-0125")
```

Ý nghĩa của đoạn code trên là?

- a) Khởi tạo model GPT 3.5 Turbo-0125.
- b) Tải pre-trained model GPT 3.5 Turbo-0125.
- c) Kiểm tra tốc độ đường truyền với ChatGPT API.

- d) Các đáp án trên đều sai.
5. Ý nghĩa của phương thức `from_template()` trong class `PromptTemplate` là?
- a) Để khởi tạo prompt template từ một file.
  - b) Để khởi tạo prompt template từ một string.
  - c) Để khởi tạo prompt template từ một danh sách các tin nhắn.
  - d) Để khởi tạo prompt template từ một prompt template có sẵn.
6. Trong LangChain, loại OutputParser nào dưới đây có thể được sử dụng để trả về kết quả của mô hình dưới dạng JSON?
- a) PydanticOutputParser.
  - b) RegexOutputParser.
  - c) JsonOutputParser.
  - d) YamlOutputParser.
7. Xét đoạn code dưới đây:

```
1 from langchain import HuggingFaceHub
2 from langchain import PromptTemplate
3
4 template = """Question: {question}
5
6 Answer: """
7 prompt = PromptTemplate(
8 template=template,
9 input_variables=['question']
10)
11
12 hub_llm = HuggingFaceHub(
13 repo_id="google/flan-t5-xl"
14)
15
16 llm_chain = prompt | hub_llm
17
18 print(llm_chain.run("What year was the World Cup first held?"))
```

Ý nghĩa của các dòng code 16 là gì?

- a) Khai báo hệ cơ sở dữ liệu vector.
- b) Khởi tạo LLMChain với LLM và Prompt.
- c) Cài đặt ủy quyền và bảo mật cho người dùng.
- d) Phân tích và trực quan hóa dữ liệu.

8. Xét đoạn code dưới đây:

```
1 from langchain_community.document_loaders import PyPDFLoader
2
3 pdf_loader = PyPDFLoader(url, extract_images=True)
4
5 docs = pdf_loader.load()
```

Tham số `extract_images` tại dòng code 3 có chức năng gì?

- a) Trả về tất cả ảnh từ file pdf.
- b) Bỏ qua ảnh, chỉ load text.
- c) Phân tích ảnh thành vector.
- d) Chuyển đổi ảnh trong file pdf thành text.

9. Tại sao chúng ta cần phải chia nhỏ các tài liệu đầu vào thành các tài liệu ngắn hơn? Chọn câu trả lời **SAI**.

- a) Giúp LLM tập trung tạo ra câu trả lời chỉ dựa trên các thông tin có liên quan.
- b) Tiết kiệm bộ nhớ cho phần cứng.
- c) Chỉ dựa vào một phần nhỏ tài liệu thì mô hình vẫn trả lời chính xác.
- d) Giúp mô hình LLM chạy nhanh hơn.

10. Xét đoạn code dưới đây:

```
1 from langchain_community.document_loaders import PyPDFLoader
2 from langchain_text_splitters import RecursiveCharacterTextSplitter
3 from langchain_community.embeddings import HuggingFaceEmbeddings
4 from langchain_chroma import Chroma
5
6 pdf_url = "https://arxiv.org/pdf/2401.18059v1.pdf"
7
```

```
8 # PDF loader
9 pdf_loader = PyPDFLoader(pdf_url, extract_images=True)
10 pdf_pages = pdf_loader.load()
11
12 # Splitter
13 splitter = RecursiveCharacterTextSplitter(
14 chunk_size=300,
15 chunk_overlap=0,
16)
17 docs = splitter.split_documents(pdf_pages)
18
19 # Embedding model
20 embedding_model = HuggingFaceEmbeddings()
21
22 # vector store
23 chroma_db = Chroma.from_documents(docs, embedding=embedding_model)
```

Nhiệm vụ của `embedding_model` là gì?

- a) Dùng biến đổi chuỗi đầu vào thành các vector cho cơ sở dữ liệu vector.
- b) Dùng để lập chỉ mục cho cơ sở dữ liệu.
- c) Dùng để tìm kiếm tài liệu.
- d) Dùng để tính toán độ tương đồng.

## 57.4 Phụ lục

1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 3 khi hết deadline phần nội dung này, ad mới copy các tài liệu bài giải nêu trên vào đường dẫn).
3. **Demo:** Web demo của ứng dụng có thể được truy cập tại [đây](#).
4. **Rubric:**

Mục	Kiến Thức	Đánh Giá
I.	<ul style="list-style-type: none"> <li>- Kiến thức về mô hình ngôn ngữ lớn (LLMs).</li> <li>- Kiến thức bài toán Retrieval Augmented Generation (RAG).</li> </ul>	<ul style="list-style-type: none"> <li>- Hiểu được các nội dung cơ bản về LLMs và RAG. Input Output của bài toán RAG và luồng xử lý cơ bản.</li> </ul>
II.	<ul style="list-style-type: none"> <li>- Các kiến thức cơ bản về thư viện LangChain.</li> <li>- Tổng quan các cách bước cơ bản trong việc sử dụng LangChain.</li> <li>- Chức năng một số hàm cơ bản trong LangChain nhằm phục cho cài đặt ứng dụng RAG.</li> <li>- Khái niệm về API và luồng triển khai API cơ bản.</li> </ul>	<ul style="list-style-type: none"> <li>- Nắm được các nội dung và chức năng cơ bản của thư viện LangChain.</li> <li>- Có thể sử dụng thư viện LangChain để cài đặt một ứng dụng RAG sử dụng LLMs.</li> </ul>

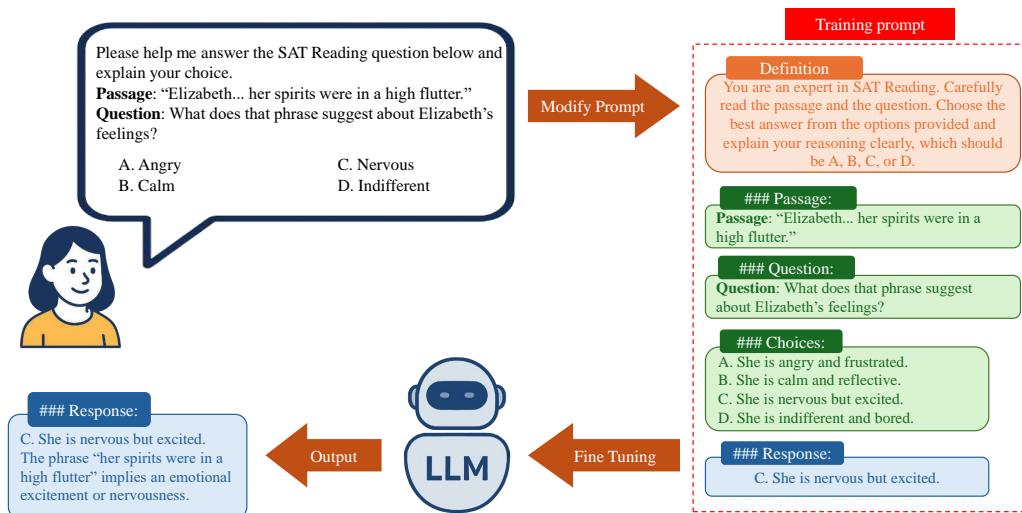
- *Hết* -

# Chương 58

## Instruction Tuning

### 58.1 Giới thiệu

**Instruction Tuning (IT)** là một trong những kỹ thuật training mô hình ngôn ngữ lớn (LLMs) rất quan trọng. Trong đó, IT giúp cải thiện khả năng của mô hình cũng như kiểm soát kết quả đầu ra. Là kiểu huấn luyện mô hình có giám sát từ bộ dữ liệu theo cặp (instruction-output), từ đó giúp mô hình thu hẹp khoảng cách giữa từ kế tiếp được sinh ra và sự chỉ dẫn của con người.



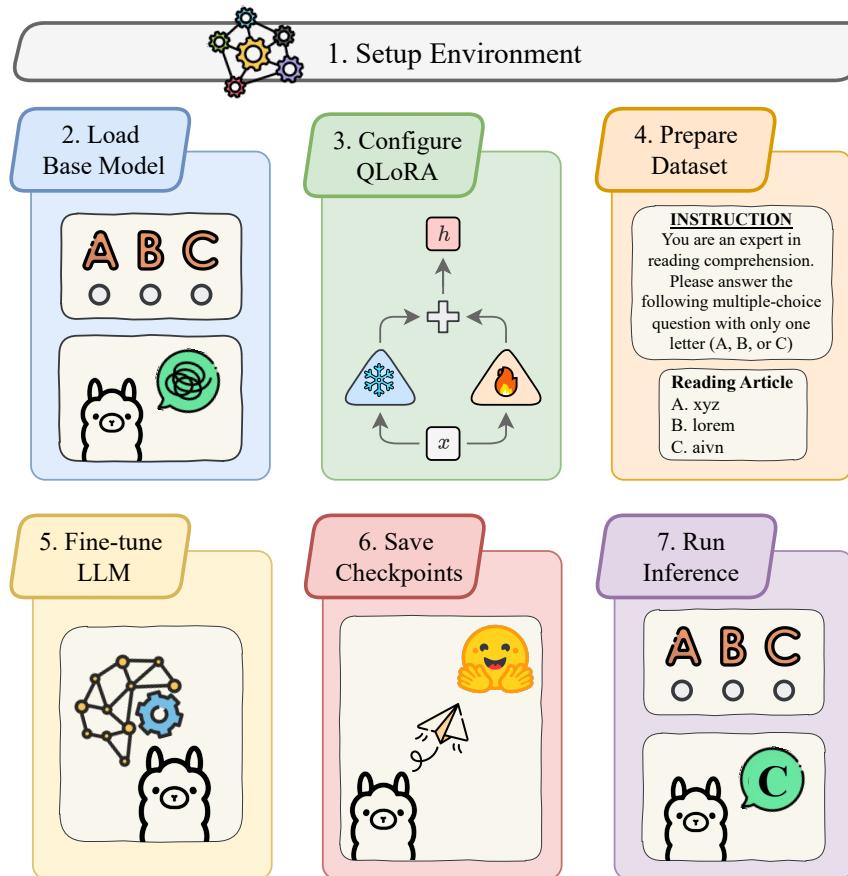
Hình 58.1: Instruction Fine-tuning mô hình ngôn ngữ lớn với dữ liệu trắc nghiệm.

Bài viết trình bày quá trình huấn luyện một mô hình ngôn ngữ lớn (LLM) với dữ liệu instruction để thực hiện các câu hỏi trắc nghiệm trong bài đọc hiểu thuộc kỳ thi SAT. Mục tiêu là xây dựng một mô hình có khả năng cải thiện độ chính xác trong các bài kiểm tra đọc hiểu SAT, đồng thời hỗ trợ

nâng cao kỹ năng đọc hiểu tiếng Anh một cách tổng quát. Bài toán được định nghĩa với đầu vào và đầu ra như sau:

- **Input:** Một câu prompt với lời hướng dẫn (instruction) để LLM thực hiện một bài bài đọc hiểu SAT và lựa chọn phương án đúng.
- **Output:** Lời phản hồi từ mô hình, trong trường hợp này là một trong các phương án trắc nghiệm đã được mô tả trong prompt.

Dựa vào nội dung mô tả trên, pipeline các bước xử lý trong bài này được minh họa theo hình dưới đây:



Hình 58.2: Fine-tuning mô hình ngôn ngữ lớn với dữ liệu instruction làm bài đọc hiểu SAT.

Trong đó:

1. **Setup Environment:** Thiết lập môi trường làm việc, bao gồm cài đặt thư viện và cấu hình phần cứng phù hợp cho huấn luyện mô hình.
2. **Load Base Model:** Tải mô hình ngôn ngữ lớn (LLM) gốc từ thư viện như Hugging Face làm nền tảng để tinh chỉnh.
3. **Configure QLoRA:** Cấu hình QLoRA để giảm tài nguyên huấn luyện bằng cách khóa các tầng gốc và chỉ tinh chỉnh các adapter nhẹ.
4. **Prepare Dataset:** Chuẩn bị dữ liệu instruction tuning bao gồm đoạn văn, câu hỏi trắc nghiệm và đáp án đúng theo định dạng đầu vào/đầu ra rõ ràng.
5. **Fine-tune LLM:** Huấn luyện mô hình trên tập dữ liệu đọc hiểu đã chuẩn bị bằng kỹ thuật QLoRA nhằm tối ưu hiệu suất.
6. **Save Checkpoints:** Lưu lại các checkpoint của mô hình trong quá trình huấn luyện để theo dõi tiến độ và phục hồi nếu cần.
7. **Run Inference:** Thực hiện suy luận (inference) trên các câu hỏi trắc nghiệm mới để đánh giá khả năng đọc hiểu của mô hình sau tinh chỉnh.

Qua pipeline trên, ta sẽ có một mô hình ngôn ngữ lớn đã được tinh chỉnh với chi phí thấp, có khả năng hiểu và trả lời hiệu quả các câu hỏi trắc nghiệm trong các bài đọc hiểu tiếng Anh như SAT.

## 58.2 Cài đặt chương trình

### 58.2.1 Cài đặt và import các thư viện cần thiết

Đầu tiên, một số thư viện sau cần được cài đặt để có thể chạy được các mô hình ngôn ngữ lớn từ thư viện HuggingFace:

```
1 !pip install -q -U bitsandbytes
2 !pip install -q -U datasets
3 !pip install -q -U git+https://github.com/huggingface/transformers.git
4 !pip install -q -U git+https://github.com/huggingface/peft.git
5 !pip install -q -U git+https://github.com/huggingface/accelerate.git
6 !pip install -q -U loralib
7 !pip install -q -U einops
```

Sau khi cài đặt hoàn tất, bước tiếp theo là import các thư viện đã tải cũng như một số thư viện khác để phục vụ cho chương trình:

```
1 import json
2 import os
3 import bitsandbytes as bnb
4 import torch
5 import torch.nn as nn
6 import transformers
7
8 from pprint import pprint
9 from tqdm import tqdm
10 from datasets import load_dataset, Dataset
11
12 from peft import (
13 LoraConfig,
14 PeftConfig,
15 PeftModel,
16 get_peft_model,
17 prepare_model_for_kbit_training
18)
19 from transformers import (
20 AutoConfig,
21 AutoModelForCausalLM,
22 AutoTokenizer,
23 BitsAndBytesConfig
24)
```

### 58.2.2 Tải pre-trained LLM

Mô hình ngôn ngữ lớn được sử dụng trong bài này là Llama đến từ Meta (Facebook), mô hình là phiên bản 3.1-Instruct với tổng cộng 8 tỷ tham số. Tuy nhiên, mô hình này cần xin quyền truy cập từ Meta. Để làm điều đó, trước tiên hãy vào [đường link này](#), hãy đăng nhập và xin cấp quyền sau khi điền đủ các thông tin được yêu cầu. Thông thường, Meta sẽ duyệt trong vài phút kể từ lúc gửi. Sau khi được cấp quyền, hãy lấy **User Access Tokens** bằng cách nhấn vào [link này](#) (đã đăng nhập trước), chọn “Create New Token”, chọn “Read”, đặt tên và nhấn “Create token”, và thay Access Tokens này vào đoạn code sau để đăng nhập HuggingFace trên notebook:

```
1 from huggingface_hub import login
2
3 login(token="hf_xxxxxxxxxxxxxxxx") # your access token
```

Khi đã đăng nhập thành công, tiến hành khởi tạo mô hình Llama như sau:

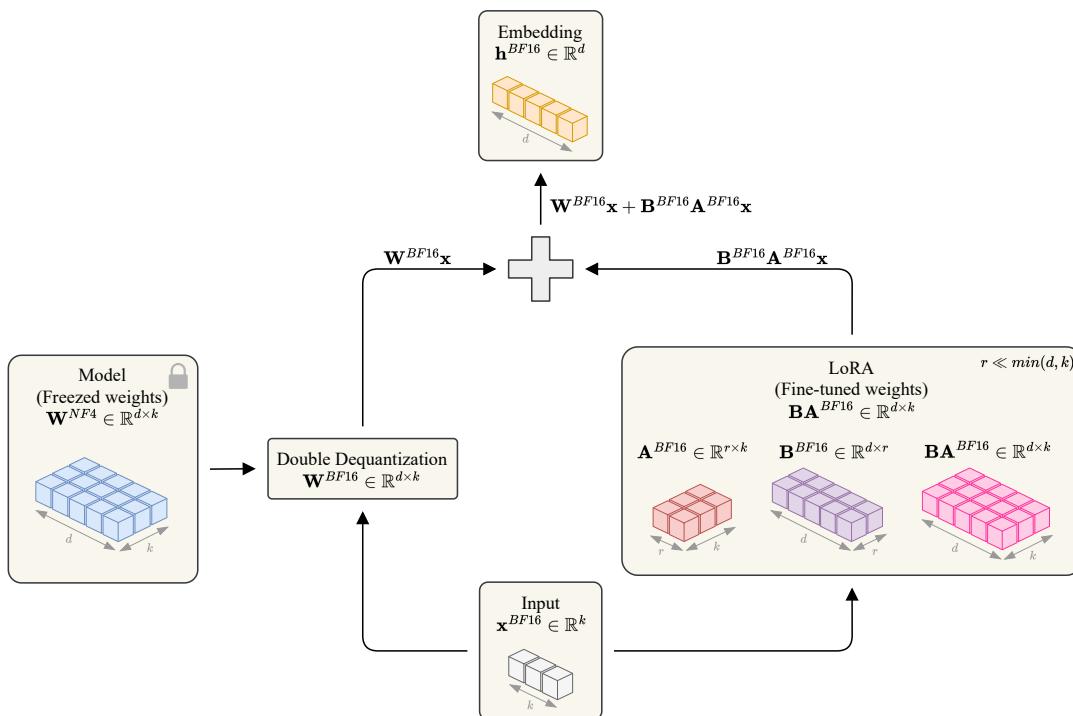
```
1 MODEL_NAME = "meta-llama/Llama-3.1-8B-Instruct"
2
3 bnb_config = BitsAndBytesConfig(
4 load_in_4bit=True,
5 bnb_4bit_quant_type="nf4",
6 bnb_4bit_compute_dtype=torch.bfloat16,
7 bnb_4bit_use_double_quant=True
8)
9
10 model = AutoModelForCausalLM.from_pretrained(
11 MODEL_NAME,
12 quantization_config=bnb_config,
13 device_map="auto",
14 trust_remote_code=True
15)
16
17 tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
18 tokenizer.pad_token = tokenizer.eos_token
19
20 model.gradient_checkpointing_enable()
```

Trong đoạn code trên, mô hình được tải với kỹ thuật lượng tử hóa 4-bit (NF4) thông qua thư viện BitsAndBytes, giúp giảm dung lượng và tối ưu hóa hiệu suất tính toán trên GPU. Tokenizer phù hợp với mô hình cũng được tải về và thiết lập token padding bằng ký tự kết thúc chuỗi (EOS). Tiếp theo, gradient

checkpointing được kích hoạt để tiết kiệm bộ nhớ khi huấn luyện. Sau đó, hàm `prepare_model_for_kbit_training()` được áp dụng nhằm điều chỉnh mô hình sẵn sàng cho quá trình fine-tuning. D

### 58.2.3 Cài đặt QLoRA

Kể đến, kỹ thuật LoRA được áp dụng lên các module attention (`q_proj` và `v_proj`) để fine-tune mô hình nhanh chóng, nhẹ nhàng, nhưng vẫn hiệu quả.



Hình 58.3: Trực quan về kỹ thuật QLoRA.

```

1 model = prepare_model_for_kbit_training(model)
2
3 peft_config = LoraConfig(
4 r=8,
5 lora_alpha=16,
6 target_modules=[
7 "q_proj",
8 "v_proj",

```

```

9],
10 lora_dropout=0.05,
11 bias="none",
12 task_type="CAUSAL_LM"
13)
14
15 model = get_peft_model(model, peft_config)

```

#### 58.2.4 Chuẩn bị bộ dữ liệu huấn luyện

Để thực hiện nhiệm vụ “đọc bài đọc và trả lời câu hỏi trắc nghiệm”, mô hình LLaMA được fine-tune trên tập dữ liệu [emozilla/sat-reading](#) từ Hugging Face. Bộ dữ liệu này bao gồm các đoạn văn và câu hỏi trích từ phần đọc hiểu của mươi bài thi thử SAT được công bố công khai. Mỗi mục dữ liệu chứa đoạn văn, câu hỏi, bốn lựa chọn trả lời, và đáp án đúng. Một số câu hỏi yêu cầu tham chiếu đến dòng cụ thể trong văn bản, được đánh dấu bằng trường boolean `requires_line`. Các mục liên quan đến biểu đồ và bảng biểu đã bị loại bỏ nhằm đảm bảo tính nhất quán của tập dữ liệu. Dữ liệu được chia thành ba phần: tập huấn luyện (298 mục), tập kiểm định (39 mục), và tập kiểm tra (38 mục), tổng cộng 375 mục. Việc tải và đọc dữ liệu được thực hiện như sau:

```
1 data = load_dataset("emozilla/sat-reading")
```

Dưới đây là một vài ví dụ điển hình trong tập dữ liệu:

Tuy nhiên, để đảm bảo cấu trúc rõ ràng cho các mẫu huấn luyện, hai hàm được định nghĩa nhằm trích xuất thông tin từ văn bản gốc, gồm:

- `extract_sections()` thực hiện phân tách nội dung đầu vào thành các phần rõ ràng như đoạn văn, câu hỏi, các lựa chọn, và đáp án dưới dạng ký tự (A, B, C, D).
- `map_answer()` được dùng để ánh xạ ký tự đáp án sang nội dung đầy đủ tương ứng trong danh sách lựa chọn.

```

1 def extract_sections(text):
2 sections = {
3 "passage": "",
4 "question": "",

```

```

5 "choices": [],
6 "answer_letter": ""
7 }
8
9 answer_part = text.split("Answer:")[-1].strip()
10 sections["answer_letter"] = answer_part[0] if answer_part else ""
11
12 content = text.split("SAT READING COMPREHENSION TEST")[-1].split(
13 Answer:) [0]
14 blocks = [b.strip() for b in content.split("\n\n") if b.strip()]
15
16 passage_lines = []
17 for line in blocks:
18 if line.startswith("Question"):
19 break
20 passage_lines.append(line)
21 sections["passage"] = "\n".join(passage_lines).strip()
22
23 for block in blocks:
24 if block.startswith("Question"):
25 q_part = block.split(", 1) if ")" in block else (block, "")
26 sections["question"] = q_part[-1].split("\n") [0].strip()
27 sections["choices"] = [line.strip() for line in block.split("\
28 n") [1:]]
29 if line.startswith(("A)", "B)", "C)", "D")
30
31 def map_answer(text, letter):
32 sections = extract_sections(text)
33 for choice in sections["choices"]:
34 if choice.startswith(f"{letter}")):
35 return choice
36 return letter

```

Sau đó, dùng hai hàm vừa định nghĩa để xây dựng một hàm tạo prompt với cấu trúc như mong muốn bằng cách sau:

```

1 LLAMA3_SYSTEM_PROMPT = """You are a helpful AI assistant developed by Meta
2 . Respond safely and accurately."""
3
4 def generate_prompt(text, answer_letter):
5 sections = extract_sections(text)
6 choices_text = "\n".join(sections['choices'])

```

```

6
7 return [
8 {
9 "role": "system",
10 "content": LLAMA3_SYSTEM_PROMPT
11 },
12 {
13 "role": "user",
14 "content": f"""Read the passage and answer the question.
15
16 ### Passage:
17 {sections["passage"]}
18
19 ### Question:
20 {sections["question"]}
21
22 ### Choices:
23 {choices_text}
24
25 Respond with ONLY the letter and full text of the correct answer."""
26 },
27 {
28 "role": "assistant",
29 "content": map_answer(text, answer_letter)
30 }
31]

```

Tiếp tục thực hiện hàm tokenize để chuẩn bị đầu vào phù hợp cho mô hình. Cụ thể là:

- Hàm `generate_and_tokenize_prompt()` nhận đầu vào là một cặp (`user_input, answer`) và sinh ra prompt hoàn chỉnh thông qua hàm `generate_prompt()`.
- Prompt sau đó được xử lý định dạng bằng `tokenizer.apply_chat_template()` để đảm bảo phù hợp với định dạng hội thoại mà tokenizer yêu cầu.
- Hàm `tokenizer()` được dùng để chuyển prompt thành tensor `input_ids`, cùng với `attention_mask` phục vụ cho quá trình huấn luyện.
- Cuối cùng, nhãn huấn luyện (labels) được gán bằng chính `input_ids` để mô hình học theo dạng tự hồi tiếp (causal language modeling).

```
1 def generate_and_tokenize_prompt(user_input, answer):
2 try:
3 full_prompt = generate_prompt(user_input, answer)
4
5 prompt_str = tokenizer.apply_chat_template(
6 full_prompt,
7 tokenize=False,
8 add_generation_prompt=False
9)
10
11 tokenized = tokenizer(
12 prompt_str,
13 padding="max_length",
14 truncation=True,
15 max_length=1024,
16 return_tensors="pt"
17)
18
19 input_ids = tokenized["input_ids"][0]
20 labels = input_ids.clone()
21
22 return {
23 "input_ids": input_ids,
24 "attention_mask": tokenized["attention_mask"][0],
25 "labels": labels
26 }
27
28 except Exception as e:
29 print(f"Error processing sample: {e}")
30 return None
```

Với các hàm đã được chuẩn bị, bước cuối cùng trong phần này là xây dựng bộ dữ liệu huấn luyện theo các bước sau:

- Duyệt qua từng mẫu dữ liệu trong tập huấn luyện gốc.
- Thực hiện tiền xử lý văn bản bằng cách loại bỏ tiêu đề thừa và chuẩn hóa đáp án bằng hàm `map_answer()`.
- Áp dụng hàm `generate_and_tokenize_prompt()` để sinh ra đầu vào `tokenized` cho từng mẫu.
- Cuối cùng, tập dữ liệu được chia thành hai phần: tập huấn luyện (`train_dataset`) và tập đánh giá (`eval_dataset`) theo tỷ lệ 90/10.

```
1 training_samples = []
2 for sample in tqdm(data["train"]):
3 try:
4 processed_text = sample["text"].replace("SAT READING COMPREHENSION
5 TEST", "").strip()
6 processed_answer = map_answer(sample["text"], sample["answer"]).
7 strip())
8
9 tokenized_sample = generate_and_tokenize_prompt(processed_text,
10 processed_answer)
11 if tokenized_sample is not None:
12 training_samples.append(tokenized_sample)
13 except Exception as e:
14 print(f"Skipping invalid sample: {e}")
15
16 training_samples = [s for s in training_samples if s is not None]
17 train_samples, val_samples = train_test_split(training_samples, test_size=
18 0.1, random_state=42)
19 train_dataset = Dataset.from_list(train_samples)
20 eval_dataset = Dataset.from_list(val_samples)
```

<|begin\_of\_text|><|start\_header\_id|>system<|end\_header\_id|>

Cutting Knowledge Date: December 2023  
Today Date: 26 Jul 2024

You are a helpful AI assistant developed by Meta. Respond safely and accurately.

<|eot\_id|><|start\_header\_id|>user<|end\_header\_id|>

Read the passage and answer the question.

### Passage:

This passage is adapted from Elizabeth Cady Stanton's address to the 1869 Woman Suffrage Convention in Washington, DC. I urge a sixteenth amendment, because

. . .

and not with her consent would one drop of blood ever be shed, one life sacrificed in vain.

### Question:

Stanton claims that which of the following was a relatively recent historical development?

### Choices:

- A) The control of society by men
- B) The spread of war and injustice
- C) The domination of domestic life by men
- D) The acknowledgment of women's true character

Respond with ONLY the letter and full text of the correct answer.<|eot\_id|>

<|start\_header\_id|>assistant<|end\_header\_id|>

D) The acknowledgment of women's true character<|eot\_id|>

Hình 58.4: Trực quan nội dung của một prompt sau khi được tổ chức lại.

### 58.2.5 Thực hiện instruction tuning LLM

Một khi việc chuẩn bị dữ liệu đã hoàn tất, quá trình huấn luyện mô hình được thực hiện theo các bước sau:

- Lớp **LogLossCallback** được định nghĩa nhằm ghi lại giá trị hàm mất mát (loss) sau mỗi bước logging, hỗ trợ theo dõi tiến trình huấn luyện theo thời gian thực.

- Các tham số huấn luyện được thiết lập thông qua **TrainingArguments**, bao gồm:
  - Batch size nhỏ (1), kết hợp với **gradient accumulation** để phù hợp với mô hình lớn.
  - Sử dụng optimizer 8-bit (**paged\_adamw\_8bit**) nhằm tiết kiệm bộ nhớ và tăng hiệu suất.
  - Scheduler dạng cosine với tỉ lệ warmup ban đầu.
  - Dánh giá và lưu mô hình mỗi 50 bước thông qua **eval\_steps** và **save\_steps**.
  - Cấu hình tự động tải lại checkpoint có hiệu suất tốt nhất theo metric loss.
- Hàm **DataCollatorForLanguageModeling()** được sử dụng để xử lý đầu vào theo chuẩn causal language modeling (không áp dụng masking).
- Trước khi huấn luyện, gradient cho đầu vào được kích hoạt và mô hình được biên dịch bằng **torch.compile()** để tối ưu hiệu suất trên phần cứng hiện đại.
- Quá trình huấn luyện được thực hiện thông qua lệnh gọi **Trainer.-train()**.

```
1 class LogLossCallback(TrainerCallback):
2 def on_log(self, args, state, control, logs=None, **kwargs):
3 if logs is not None and "loss" in logs:
4 print(f"Step {state.global_step} - Loss: {logs['loss']:.4f}")
5
6 training_args = TrainingArguments(
7 per_device_train_batch_size=1,
8 gradient_accumulation_steps=2,
9 num_train_epochs=2,
10 learning_rate=2e-4,
11 fp16=True,
12 save_total_limit=3,
13 logging_steps=10,
14 output_dir="llama3-8b-sat-reading",
15 optim="paged_adamw_8bit",
16 lr_scheduler_type="cosine",
17 warmup_ratio=0.05,
18 eval_strategy="steps",
```

```
19 eval_steps=50,
20 save_strategy="steps",
21 save_steps=50,
22 load_best_model_at_end=True,
23 metric_for_best_model="loss",
24 greater_is_better=False,
25 report_to="none",
26 remove_unused_columns=False
27)
28
29 data_collator = DataCollatorForLanguageModeling(
30 tokenizer=tokenizer,
31 mlm=False,
32 pad_to_multiple_of=8
33)
34
35 trainer = Trainer(
36 model=model,
37 train_dataset=train_dataset,
38 eval_dataset=eval_dataset,
39 args=training_args,
40 data_collator=data_collator,
41 callbacks=[LogLossCallback()]
42)
43
44 model.config.use_cache = False
45 model.enable_input_require_grads()
46 model = torch.compile(model)
47
48 trainer.train()
```

### 58.2.6 Save Checkpoints

Khi quá trình huấn luyện hoàn tất, mô hình được lưu để sử dụng hoặc chia sẻ sau này. Cụ thể:

- Mô hình sau huấn luyện được lưu cục bộ vào thư mục "**trained--model**" bằng hàm `save_pretrained()`.
- Sau đó, mô hình được đẩy lên Hugging Face Hub thông qua hàm `push_to_hub()`, giúp dễ dàng chia sẻ và tái sử dụng trong các dự án khác.

- Để thực hiện bước này, người dùng cần có tài khoản trên Hugging Face và phải cung cấp `use_auth_token=True` để xác thực quyền truy cập.

```

1 model.save_pretrained("trained-model")
2 PEFT_MODEL = "your_huggingface_user_name/instructionTuning-llama-3-1-8B-
 SAT-reading-solver"
3 model.push_to_hub(PEFT_MODEL, use_auth_token=True)

```

### 58.2.7 Run Inference

Cuối cùng, để đánh giá mô hình sau huấn luyện, quá trình tải mô hình và thực hiện suy luận (inference) trên tập kiểm tra được tiến hành như sau:

- Mô hình được tải từ Hugging Face Hub thông qua ID đã định nghĩa trước trong biến `PEFT_MODEL`
- Tiếp theo là định nghĩa lại các thành phần liên quan đến mô hình tương tự như các đoạn code đã thực hiện bên trên.
- Sau khi tải mô hình và tokenizer, mô hình được gắn adapter PEFT thông qua hàm `PeftModel.from_pretrained()` để khôi phục các tham số đã fine-tuned.

```

1 PEFT_MODEL = "your_huggingface_user_name/instructionTuning-llama-3-1-8B-
 SAT-reading-solver"
2
3 config = PeftConfig.from_pretrained(PEFT_MODEL)
4
5 bnb_config = BitsAndBytesConfig(
6 load_in_4bit=True,
7 bnb_4bit_use_double_quant=True,
8 bnb_4bit_quant_type="nf4",
9 bnb_4bit_compute_dtype=torch.bfloat16
10)
11
12 model = AutoModelForCausalLM.from_pretrained(
13 config.base_model_name_or_path,
14 quantization_config=bnb_config,
15 device_map="auto",
16 trust_remote_code=True
17)

```

```

18
19 tokenizer = AutoTokenizer.from_pretrained(config.base_model_name_or_path)
20 tokenizer.pad_token = tokenizer.eos_token
21
22 model = PeftModel.from_pretrained(model, PEFT_MODEL)

```

Khi đã khôi phục mô hình, quá trình định dạng prompt và sinh câu trả lời được thực hiện để đánh giá khả năng suy luận của mô hình trên các mẫu kiểm tra, chi tiết như sau:

- Trong hàm `predict()`, prompt đầu vào được định dạng lại theo chuẩn hội thoại bằng `tokenizer.apply_chat_template`, sau đó thực hiện sinh câu trả lời bằng hàm `model.generate()` với các tham số cấu hình sinh (ví dụ: không sampling, temperature = 0.0, giới hạn số token đầu ra).
- Hàm `extract_answer()` được dùng để trích xuất câu trả lời đầy đủ từ đầu ra của mô hình, và được so sánh với đáp án gốc sau khi ánh xạ bằng `map_answer()`.
- Vòng lặp đánh giá thực hiện suy luận trên 10 mẫu đầu tiên của tập kiểm tra, in ra nội dung câu hỏi, các lựa chọn, dự đoán từ mô hình, đáp án đúng và kết quả đúng/sai.

```

1 generation_config = GenerationConfig(
2 max_new_tokens=64,
3 temperature=0.0,
4 top_p=1.0,
5 do_sample=False,
6 repetition_penalty=1.0,
7 eos_token_id=tokenizer.eos_token_id,
8 pad_token_id=tokenizer.eos_token_id
9)
10
11 def predict(text):
12 messages = format_test_prompt(text)
13
14 prompt_text = tokenizer.apply_chat_template(
15 messages,
16 add_generation_prompt=True,
17 tokenize=False
18)

```

```
19 inputs = tokenizer(prompt_text, return_tensors="pt").to(model.device)
20
21 with torch.no_grad():
22 outputs = model.generate(
23 input_ids=inputs["input_ids"],
24 attention_mask=inputs["attention_mask"],
25 generation_config=generation_config
26)
27
28
29 output_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
30 return extract_answer(output_text)
31
32 for i in range(10):
33 print("=". * 100)
34 sample = data["test"][i]
35 input_text = sample["text"]
36 true_answer = sample["answer"].strip()
37
38 predicted_answer = predict(input_text)
39
40 true_answer_full = map_answer(input_text, true_answer)
41
42 pred_choice = extract_choice_letter(predicted_answer)
43 true_choice = extract_choice_letter(true_answer_full)
44
45 print(f"### Sample {i+1}")
46 print(f"[Question]\n{extract_sections(input_text)[\"question\"]}")
47 print(f"[Choices]\n{extract_sections(input_text)[\"choices\"]}")
48 print(f"\n[Model Prediction]\n{predicted_answer}")
49 print(f"\n[Ground Truth]\n{true_answer_full}")
50 print(f"""\nResult: {"CORRECT" if pred_choice == true_choice else "
51 INCORRECT"}""")
52 print("=". * 100 + "\n")
```

### 58.3 Câu hỏi trắc nghiệm

1. Trong ngữ cảnh về mô hình ngôn ngữ lớn (LLMs), Instruction Tuning được hiểu như thế nào?
  - a) Huấn luyện mô hình trên task mới mà không cần mẫu dữ liệu nào.
  - b) Điều chỉnh kết quả của mô hình trong quá trình deploy.
  - c) Huấn luyện mô hình để tuân theo các yêu cầu cụ thể (instruction).
  - d) Giảm kích thước của mô hình để tăng độ hiệu quả.
2. Instruction Tuning là một dạng của kiểu học gì?
  - a) Supervised Learning.
  - b) Self-supervised Learning.
  - c) Unsupervised Learning.
  - d) Reinforcement Learning.
3. Khi thực hiện Instruction Tuning, hàm loss nào sau đây có thể được sử dụng?
  - a) Mean Squared Error (MSE).
  - b) Hinge Loss.
  - c) Kullback-Leibler Divergence.
  - d) Cross-Entropy Loss.
4. Mệnh đề sau đúng hay sai: “Ta luôn luôn nên áp dụng Instruction Tuning để giảm thiểu chi phí tính toán trong quá trình training”?
  - a) Đúng.
  - b) Sai.
5. Trong LLMs, khái niệm prompt được hiểu như thế nào?
  - a) Một phần mềm hỗ trợ giúp tối ưu hóa hiệu suất của mô hình.
  - b) Một kỹ thuật mã hóa thông tin riêng tư trong quá trình đào tạo mô hình.

- c) Một câu hỏi hoặc yêu cầu mà người dùng đưa ra cho mô hình.
- d) Một thuật toán đặc biệt để phân loại dữ liệu đầu vào.
6. Trong LLMs, ta nên áp dụng kỹ thuật nào sau đây để cải thiện khả năng thực hiện một task cụ thể nào đó của mô hình mà không cần training?
- a) Sử dụng kỹ thuật Transfer Learning.
  - b) Ứng dụng khả năng Zero-shot Learning của mô hình.
  - c) Lập trình thủ công tại bước hậu xử lý cho mỗi task.
  - d) Mở rộng bộ dữ liệu training.
7. Kỹ thuật prompting nào dưới đây cung cấp cho mô hình chỉ một ví dụ về task cần làm?
- a) One-shot Learning.
  - b) Few-shot Learning.
  - c) Continuous Learning.
  - d) Transfer Learning.
8. Câu nào sau đây mô tả đúng về kỹ thuật Parameter Efficient Fine-tuning (PEFT)?
- a) Một kỹ thuật dùng để huấn luyện mô hình trên bộ dữ liệu cực lớn.
  - b) Một kỹ thuật liên quan đến việc cập nhật một phần nhỏ tham số của mô hình khi huấn luyện.
  - c) Một kỹ thuật huấn luyện dành riêng cho các mô hình có kích thước nhỏ (dưới 1 tỷ tham số).
  - d) Một kỹ thuật để tăng chi phí tính toán của mô hình.
9. Mệnh đề sau đúng hay sai: “Low-Rank Adaptation (LoRA) là một kỹ thuật về PEFT”?
- a) Đúng.
  - b) Sai.

10. So với LoRA, QLoRA có điểm gì khác biệt gì trong việc huấn luyện LLMs?
- a) QLoRA lượng tử hóa (quantize) tham số mô hình; LoRA thì không.
  - b) QLoRA sử dụng nhiều tham số mô hình; LoRA ít hơn.
  - c) QLoRA tối ưu khả năng tổng quát của mô hình; LoRA tối ưu trên một task cụ thể.
  - d) QLoRA giảm kích thước mô hình; LoRA tăng lên.

## 58.4 Phụ lục

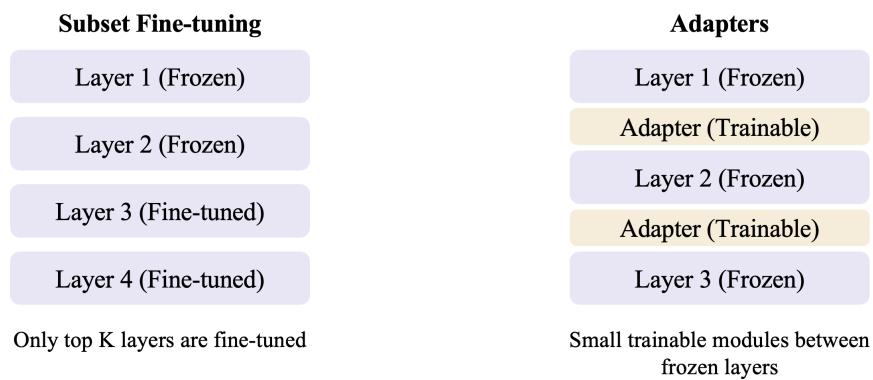
- (a) **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
- (b) **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng thứ 6 khi hết deadline phần nội dung này, ad mới copy các tài liệu bài giải nêu trên vào đường dẫn).
- (c) **Demo:** Web demo và mã nguồn của ứng dụng có thể được truy cập tại [đây](#).
- (d) **Rubric:**

- *Hết* -

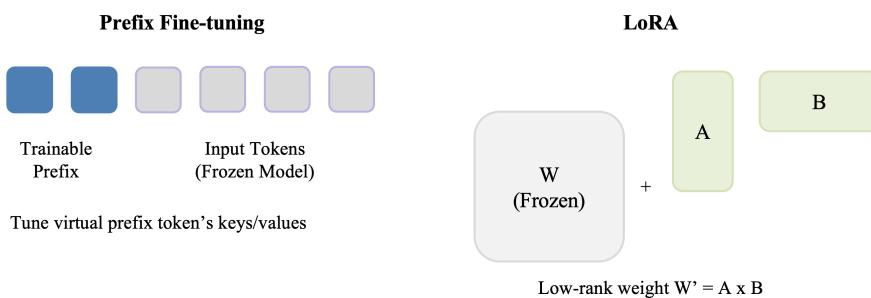
# Chương 59

## Project 1: Áp dụng PEFT cho bài toán hỏi đáp các câu hỏi trắc nghiệm

### 59.1 Giới thiệu



Hình 59.1: Subset Fine-tuning and Adapters Tuning.



Hình 59.2: Pre-fix Tuning and LoRA.

Trong bối cảnh các mô hình ngôn ngữ lớn (Large Language Models – LLMs) ngày càng phát triển cả về quy mô và sức mạnh, việc tinh chỉnh

(fine-tune) toàn bộ mô hình trở nên tốn kém cả về bộ nhớ lẫn chi phí tính toán. **Parameter-Efficient Fine-Tuning (PEFT)** là một nhóm các kỹ thuật nhằm giảm số lượng tham số cần cập nhật trong quá trình fine-tune, giữ nguyên phần lớn trọng số gốc của mô hình, từ đó giúp tăng hiệu quả huấn luyện và khả năng mở rộng.

Mục tiêu chính của PEFT:

- Tận dụng sức mạnh của mô hình tiền huấn luyện (pre-trained).
- Giảm số lượng tham số cần cập nhật trong quá trình tinh chỉnh.
- Dễ dàng chia sẻ, lưu trữ và tái sử dụng các mô hình đã tinh chỉnh.

## 1. Subset Fine-tuning

Subset Fine-tuning là một phương pháp tinh chỉnh hiệu quả tham số bằng cách chỉ tinh chỉnh một phần (subset) các tầng của mô hình, giữ nguyên (freeze) các tầng còn lại. Cơ chế hoạt động: Chỉ top-K tầng cuối cùng (thường là những tầng gần output) được fine-tune, các tầng trước đó được giữ nguyên.

Ưu điểm:

- Giảm chi phí huấn luyện vì số lượng tham số cập nhật ít hơn.
- Dễ triển khai trên hầu hết các kiến trúc Transformer hiện có.

Hạn chế:

- Không tận dụng toàn bộ cấu trúc mô hình để thích nghi với dữ liệu mới.
- Dễ bị overfit nếu fine-tune quá ít tầng hoặc underfit nếu chọn tầng không phù hợp.

## 2. Adapter-tuning

Adapters là các module nhỏ được chèn vào giữa các tầng của mô hình gốc. Mô hình chính được giữ nguyên, chỉ các module Adapter là được huấn luyện.

Kiến trúc Adapter phổ biến: Down-projection → Non-linearity → Up-projection (thường với bottleneck rank nhỏ, ví dụ: 16, 32)

Ưu điểm:

- Có thể dễ dàng huấn luyện nhiều task khác nhau bằng cách thay Adapter.
- Thích hợp cho multi-task learning hoặc deployment nhiều mô hình nhẹ.
- Cập nhật ít tham số nhưng vẫn đạt hiệu quả cao.

Hạn chế:

- Cần chèn các module vào mô hình, đòi hỏi can thiệp code.
- Tăng độ trễ nhẹ trong quá trình inference.

### 3. Prefix Tuning

Prefix Tuning là kỹ thuật tinh chỉnh bằng cách thêm các vector học được (prefix vectors) vào phần đầu của chuỗi đầu vào trong mỗi layer, đồng thời giữ nguyên toàn bộ mô hình gốc.

Cách hoạt động:

- Huấn luyện các prefix embedding, thường biểu diễn attention key/value bổ sung.
- Không can thiệp trực tiếp vào tham số gốc của mô hình.

Ưu điểm:

- Số lượng tham số cần huấn luyện cực kỳ nhỏ (thường <1%).
- Dễ mở rộng sang nhiều tác vụ khác nhau.

Hạn chế:

- Phụ thuộc vào khả năng của mô hình trong việc xử lý prefix (không phải mô hình nào cũng hỗ trợ).
- Cần số lượng prefix tương ứng với mỗi tầng.

### 4. Low-rank Adaptation

LoRA là kỹ thuật tinh chỉnh bằng cách chèn ma trận low-rank để mô phỏng sự thay đổi tham số trong các tầng Attention.

Cách hoạt động:

- Các ma trận trọng số W được đóng băng.
- Thay vào đó, huấn luyện hai ma trận A và B có rank thấp sao cho:

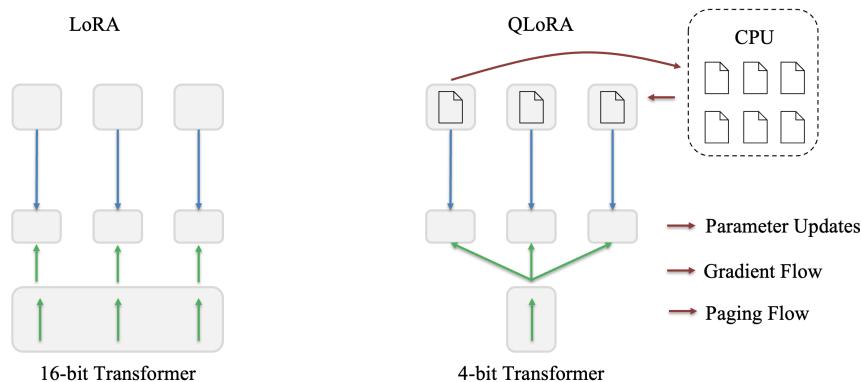
Ưu điểm:

- Tối ưu về bộ nhớ: số tham số học được giảm mạnh (rank thấp).
- Hiệu quả tương đương hoặc tốt hơn full fine-tuning trong nhiều task.
- Không cần can thiệp quá sâu vào mô hình, dễ áp dụng.

Hạn chế:

- Cần chọn rank hợp lý (quá thấp → thiếu năng lực học, quá cao → tốn tài nguyên).
- Dễ overfit nếu áp dụng không đúng cấu hình.

## 5. QLoRA



Hình 59.3: LoRA and QLoRA.

QLoRA là viết tắt của Quantized Low-Rank Adapter, là một phương pháp fine-tuning cực kỳ tiết kiệm tài nguyên, cho phép huấn luyện các mô hình ngôn ngữ cực lớn (tới 65B tham số) ngay cả trên GPU 24 hoặc 48 GB (consumer GPUs).

Mục tiêu chính của QLoRA:

- Kết hợp sức mạnh của LoRA với mô hình được lượng tử hóa (quantized).
- Giảm tối đa dung lượng bộ nhớ và chi phí tính toán khi fine-tune.
- Giữ hiệu năng tương đương hoặc tốt hơn so với full fine-tuning trong nhiều tác vụ.

QLoRA kết hợp 3 ý tưởng chính:

- 4-bit Quantization của mô hình gốc

Trước tiên, mô hình gốc (pre-trained model) được lượng tử hóa về 4-bit bằng kỹ thuật NF4 (Normalized Float 4-bit) – một định dạng mới giữ lại tốt hơn phân phối gốc của trọng số.

Điều này giúp giảm đáng kể bộ nhớ RAM/GPU, cho phép chạy mô hình lớn hơn rất nhiều.

- Low-Rank Adapter

Mô hình gốc được giữ nguyên (sau khi lượng tử hóa).

Các ma trận Low-Rank ( $A \times B$ ) như trong LoRA sẽ được huấn luyện thêm, nhưng vẫn ở độ chính xác float32 hoặc bfloat16.

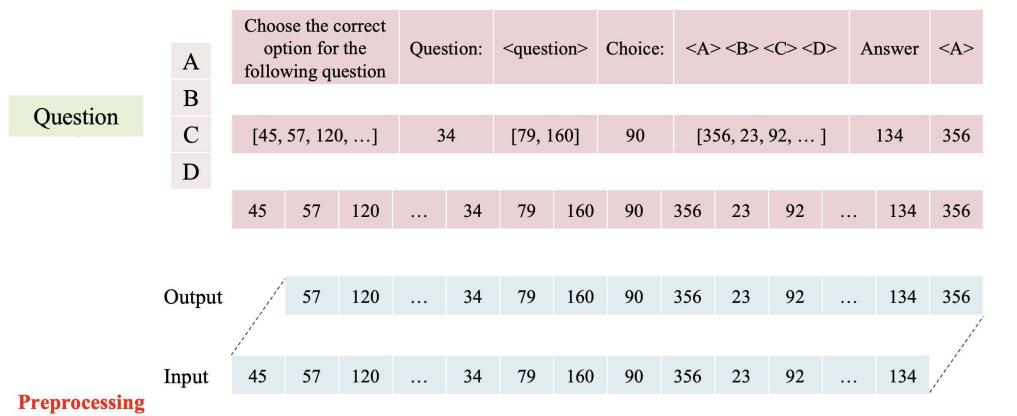
Như vậy, chỉ có rất ít tham số cần học (low-rank), và không cần cập nhật mô hình gốc.

- Double Quantization

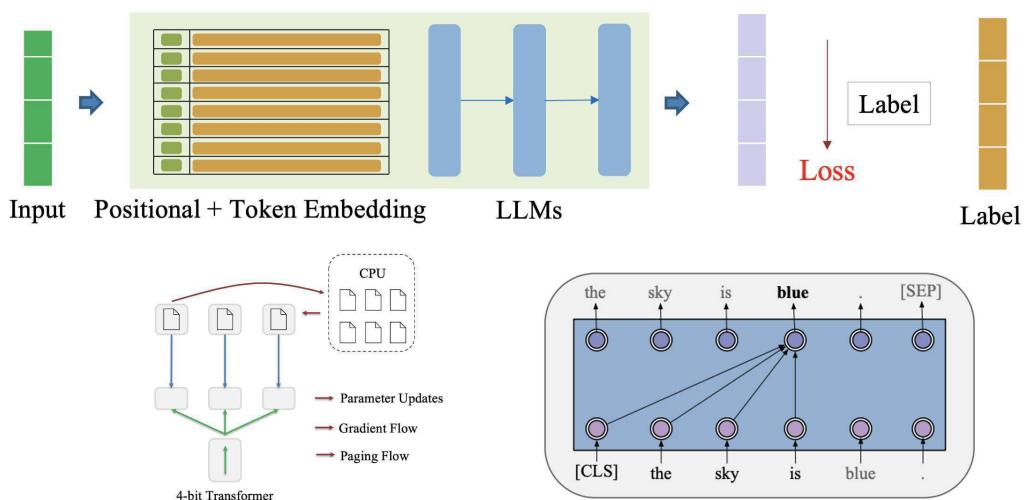
Tăng cường nén bằng cách tiếp tục lượng tử hóa các giá trị lượng tử hóa – giúp giảm thêm nhu cầu lưu trữ mà không làm mất thông tin.

## 59.2 PEFT for Multiple-choice QA

Mô hình áp dụng các phương pháp trong PEFT áp dụng cho bài toán Multiple-choice QA được thực hiện với 2 phần như sau:



Hình 59.4: Preprocessing Stage.



Hình 59.5: Training Stage.

Quá trình huấn luyện mô hình như sau:

## 1. Cài đặt thư viện và tải về bộ dữ liệu

Đầu tiên là cài đặt một số thư viện để có thể chạy được các mô hình ngôn ngữ lớn từ thư viện Unslot. Và bộ dữ liệu được sử dụng là MedMCQA, với mỗi câu hỏi sẽ có 4 đáp án và một đáp án đúng để làm nhãn.

```

1 # Install libs
2 !pip install -q unsloth==2025.3.15 datasets==3.5.0
3 !pip install transformers bitsandbytes
4
5 from datasets import load_dataset
6
7 ds = load_dataset("openlifescienceai/medmcqa")
8 del ds["test"]

```

## 2. Modeling

Mô hình LLaMA-3.2-1B được tải từ thư viện unsloth với kỹ thuật lượng tử hóa 4-bit, giúp giảm dung lượng và tối ưu hóa hiệu suất tính toán trên GPU.

Tiếp theo, kỹ thuật LoRA được áp dụng lên nhiều thành phần của mô hình, bao gồm các modules attention (q\_proj, k\_proj, v\_proj, o\_proj), các modules feed-forward network (up\_proj, down\_proj), và module gating (gate\_proj).

Bên cạnh đó, gradient checkpointing được kích hoạt giúp tiết kiệm bộ nhớ trong quá trình huấn luyện mà không làm giảm hiệu suất.

Đặc biệt, ReLoRA cũng được kích hoạt với use\_reloRA=True, cho phép mô hình định kỳ hợp nhất các tham số LoRA vào mô hình gốc và tái khởi tạo ma trận LoRA. Kỹ thuật này không chỉ giúp cải thiện hiệu quả huấn luyện mà còn duy trì bộ nhớ thấp.

```

1 from unsloth import FastLanguageModel
2
3 max_seq_length = 2048
4 model, tokenizer = FastLanguageModel.from_pretrained(
5 model_name="unsloth/Llama-3.2-1B-bnb-4bit",
6 max_seq_length=max_seq_length,
7 load_in_4bit=True,
8 dtype=None,

```

```

9)
10
11 model = FastLanguageModel.get_peft_model(
12 model,
13 r=16,
14 lora_alpha=16,
15 lora_dropout=0,
16 target_modules=[
17 "q_proj", "k_proj", "v_proj", "up_proj",
18 "down_proj", "o_proj", "gate_proj"],
19 use_rslora=True,
20 use_gradient_checkpointing="unslot",
21 random_state = 42,
22 loftq_config = None,
23)
24 print(model.print_trainable_parameters())

```

### 3. Preprocessing

Mã hoá văn bản thành các vector và kết hợp thiết kế prompt xây dựng đầu vào mô hình qua các bước:

- Lấy dữ liệu từ examples: câu hỏi (question), danh sách các lựa chọn (opa, opb, opc, opd), đáp án (cop).
- Ánh xạ đáp án (cop) vào nhãn id2label: vì đáp án cop có định dạng số nguyên (0, 1, 2, 3), ta ánh xạ chúng vào các nhãn tương ứng (0 -> A, 1 -> B, 2 -> C, 3 -> D).
- Sử dụng .format() để kết hợp câu hỏi và các lựa chọn đáp án thành một chuỗi hoàn chỉnh, theo định dạng yêu cầu.
- Cuối cùng, áp dụng hàm map() lên toàn bộ dataset để chuẩn hóa dữ liệu, giúp dữ liệu được biến đổi theo quy trình đã định, sẵn sàng cho huấn luyện.

Cấu trúc của prompt:

- Phần hướng dẫn: "Choose the correct option for the following question".
- Phần câu hỏi: "Question: <question>"(nội dung câu hỏi cụ thể thay cho <question>).
- Phần lựa chọn: "Choice: <A> <B> <C> <D>"(các phương án lựa chọn A, B, C, D).
- Phần đáp án: "Answer: <A>"(đáp án đúng, trong ví dụ này là A).

```
1 data_prompt = """Choose the correct option for the
2 following question.
3
4 ### Question:
5 {}
6
7 ### Choice:
8 {}
9
10 ### Answer:
11 """
12
13 id2label = {
14 0: 'A',
15 1: 'B',
16 2: 'C',
17 3: 'D'
18 }
19
20 def formatting_prompt(examples):
21 questions = examples["question"]
22 opas = examples["opa"]
23 opbs = examples["opb"]
24 opcs = examples["opc"]
25 opds = examples["opd"]
26 cops = examples["cop"]
27
28 texts = []
29 for idx in range(len(questions)):
30 question = questions[idx]
31 opa = opas[idx]
32 opb = opbs[idx]
33 opc = opcs[idx]
34 opd = opds[idx]
35 answer = id2label[cops[idx]]
36 if answer == "A":
37 answer = answer + " " + opa
38 elif answer == "B":
39 answer = answer + " " + opb
40 elif answer == "C":
41 answer = answer + " " + opc
42 elif answer == "D":
43 answer = answer + " " + opd
44
45 choices = f"A. {opa}. B. {opb}. C. {opc}. D. {opd}
```

```

45 }
46 text = data_prompt.format(question, choices)
47 texts.append(text)
48 return {"text": texts,}
49 process_ds = ds.map(formatting_prompt, batched=True)

```

#### 4. Training

Sau khi dữ liệu huấn luyện được chuẩn bị, quá trình tùy chỉnh các tham số huấn luyện được thiết lập thông qua TrainingArguments, bao gồm:

- Giá trị `per_device_train_batch_size` kết hợp với `gradient_accumulation_steps` để đạt số lượng mẫu mà mô hình sẽ xử lý trước khi cập nhật trọng số phù hợp với tài nguyên GPU và số bước huấn luyện bạn mong muốn.
- Sử dụng optimizer 8-bit (`adamw_8bit`) nhằm tiết kiệm bộ nhớ và tăng hiệu suất.
- Scheduler dạng linear kết hợp với `warmup_steps=10` để thực hiện warm-up trước khi học với tốc độ đầy đủ.
- Đánh giá và lưu mô hình mỗi 50 bước thông qua `eval_steps` và `save_steps`.
- Cấu hình tự động tải lại mô hình có hiệu suất tốt nhất vào cuối quá trình huấn luyện bằng `load_best_model_at_end=True`.

Khi các tham số huấn luyện đã được cấu hình, mô hình huấn luyện được bắt đầu với SFTTrainer. Các tham số huấn luyện được truyền vào qua đối tượng `args`, và các bộ dữ liệu huấn luyện và kiểm tra được chỉ định qua `train_dataset` và `eval_dataset`.

Thêm vào đó, tham số `dataset_text_field="text"` xác định trường chứa văn bản đầu vào trong dataset.

Cuối cùng, quá trình huấn luyện được thực hiện thông qua lệnh `trainer.train()`.

```

1 from trl import SFTTrainer
2 from transformers import TrainingArguments
3 from unslloth import is_bfloat16_supported
4
5 args = TrainingArguments(
6 output_dir="med-mcqa-llama-3.2-1B-4bit-lora",

```

```
7 logging_dir="logs",
8 learning_rate=3e-4,
9 lr_scheduler_type="linear",
10 per_device_train_batch_size=64,
11 gradient_accumulation_steps=16,
12 num_train_epochs=2,
13 eval_strategy="steps",
14 save_strategy="steps",
15 logging_strategy="steps",
16 eval_steps=50,
17 save_steps=50,
18 logging_steps=50,
19 save_total_limit=1,
20 load_best_model_at_end=True,
21 fp16=not is_bfloat16_supported(),
22 bf16=is_bfloat16_supported(),
23 optim="adamw_8bit",
24 weight_decay=0.01,
25 warmup_steps=10,
26 seed=0,
27),
28
29 trainer=SFTTrainer(
30 model=model,
31 tokenizer=tokenizer,
32 args=args,
33 train_dataset=process_ds["train"],
34 eval_dataset=process_ds["validation"],
35 dataset_text_field="text",
36)
37
38 trainer.train()
```

## 5. Save Checkpoints

Khi quá trình huấn luyện hoàn tất, mô hình được lưu để sử dụng hoặc chia sẻ sau này. Cụ thể:

- Mô hình sau huấn luyện được lưu cục bộ vào thư mục "unslloth-llama-traned" bằng hàm `save_pretrained()`.
- Sau đó, mô hình được đẩy lên Hugging Face Hub thông qua hàm `push_to_hub()`, giúp dễ dàng chia sẻ và tái sử dụng trong các dự án khác.

- Để thực hiện bước này, người dùng cần có tài khoản trên Hugging Face và phải cung cấp `use_auth_token=True` để xác thực quyền truy cập.

```

1 from huggingface_hub import login
2
3 login(token="hfxxxxxxxxxxxxxx") # your access token
4
5 model.save_pretrained("unslloth-llama-traned")
6 PEFT_MODEL = "your_huggingface_user_name/Llama-3.2-1B-bnb
 -4bit-MedMCQA"
7 model.push_to_hub(PEFT_MODEL, use_auth_token=True)

```

## 6. Inference

Sau khi huấn luyện, sử dụng mô hình để thực hiện suy luận (inference) trên một ví dụ như sau:

- Option 1:**
  - Sử dụng `FastLanguageModel.from_pretrained()` để tải mô hình đã huấn luyện với các tham số cấu hình như `max_seq_length` và `load_in_4bit`.
  - `FastLanguageModel.for_inference(model)` giúp tối ưu hóa mô hình cho quá trình suy luận (inference).
  - Dữ liệu đầu vào được tokenized bằng `tokenizer`, sau đó mô hình sinh kết quả dựa trên `generate()`.
  - Kết quả được giải mã bằng `tokenizer.decode()`.
- Option 2:**
  - Sử dụng `pipeline("text-generation")` trong thư viện `transformers` để tạo ra một pipeline sẵn có cho tác vụ sinh văn bản.
  - Dữ liệu đầu vào được xử lý bằng pipeline và mô hình sẽ trả về kết quả dựa trên `max_new_tokens`.
  - Kết quả trả về dưới dạng chuỗi văn bản sinh ra từ mô hình.

```

1 from transformers import pipeline
2
3 #Option 1
4 max_seq_length = 2048
5 model,tokenizer = FastLanguageModel.from_pretrained(

```

```
6 model_name = PEFT_MODEL,
7 max_seq_length = max_seq_length,
8 load_in_4bit = True,
9 dtype = None,
10)
11 FastLanguageModel.for_inference(model)
12 model.to("cuda")
13
14 input_text = process_ds["validation"][0]["text"]
15
16 inputs = tokenizer(input_text, return_tensors="pt").to("cuda")
17
18 output = model.generate(inputs["input_ids"], max_length=128)
19
20 generated_text = tokenizer.decode(output[0], skip_special_tokens=True)
21 print(f"Answer: {generated_text}")
22
23 #Answer: D Local anesthesia is effective only when the
24 nerve is not covered by myelin sheath\n
25
26 #Option 2
27 generator = pipeline("text-generation", model="thainq107/
28 med-mcqa-llama-3.2-1B-4bit-lora")
29
30 output = generator([process_ds["validation"][0]["text"]], max_new_tokens=128, return_full_text=False)[0]
31
32 # [{"generated_text": "D Local anesthesia is effective
33 only when the nerve is not covered by myelin sheath\n
34 "}]
```

## 7. Deployment

Triển khai mô hình trên streamlit. Tham khảo về [code](#) và [demo](#).

# Multiple-choice Question Answering

**Model: LLaMA-3.2-1B. Dataset: MedMCQA**

Prompt:

"Choose the correct option for the following question. ### Question: Which of the following is not true?

"Choose the correct option for the following question.

### Question:

Which of the following is not true for myelinated nerve fibers:

### Choice:

- A. Impulse through myelinated fibers is slower than non-myelinated fibers.
- B. Membrane currents are generated at nodes of Ranvier.
- C. Saltatory conduction of impulses is seen.
- D. Local anesthesia is effective only when the nerve is not covered by myelin sheath

### Answer:

D Local anesthesia is effective only when the nerve is not covered by myelin sheath

Hình 59.6: Multiple-choice QA deployment on streamlit.

### 59.3 Câu hỏi trắc nghiệm

**Câu hỏi 123** Trong kiến trúc Llama 3.2-1B, chọn nhóm tầng nào để gắn LoRA adapters vừa can thiệp trực tiếp vào Attention và Feed-Forward nhưng vẫn đạt hiệu quả cao về tham số?

- a) embed\_tokens và lm\_head
- b) q\_proj, k\_proj, v\_proj, o\_proj, gate\_proj, up\_proj, down\_proj
- c) Toàn bộ LayerNorm
- d) Chỉ gate\_proj và up\_proj

**Câu hỏi 124** Nếu không đổi token PAD trong labels thành -100 trước khi tính loss, điều gì xảy ra?

- a) Mô hình bị ép học dự đoán PAD, loss tăng và chất lượng giảm
- b) Trainer tự động bỏ qua PAD
- c) Quá trình train nhanh hơn vì ít mask
- d) Gây lỗi shape mismatch trong CrossEntropyLoss

**Câu hỏi 125** Đặt max\_seq\_length = 128. Khi prompt + answer dài hơn giới hạn và bị truncation, rủi ro nghiêm trọng nhất đối với bài MCQA là gì?

- a) Thiếu <eos> nên mô hình không dừng sinh
- b) Mất <bos> làm lệch embedding
- c) Bị cắt mất một hoặc nhiều lựa chọn đáp án làm mô hình thiếu dữ liệu suy luận
- d) Tăng bộ nhớ vì padding

**Câu hỏi 126** Trong bốn cấu hình fine-tune sau, cấu hình nào tiết kiệm VRAM nhất?

- a) Full-finetune fp16 + ZeRO-1
- b) LoRA bf16
- c) LoRA 8-bit + gradient checkpointing
- d) QLoRA 4-bit + FlashAttention-v2 + gradient checkpointing

**Câu hỏi 127** Với DataCollatorForSeq2Seq, lý do phải truyền model=model là gì?

- a) Lấy config.bos\_token\_id
- b) Tự sinh <eos> khi cần

- c) Suy luận label\_pad\_token\_id từ model.config.pad\_token\_id nếu người dùng không cung cấp
- d) Không chức năng gì, chỉ giữ API thông nhất

**Câu hỏi 128** Đóng băng toàn bộ bias và LayerNorm, chỉ LoRA-hoá các linear nhằm mục đích chính nào?

- a) Giảm FLOPs mà không ảnh hưởng hội tụ
- b) Tránh “catastrophic forgetting” trong LayerNorm
- c) Bias và LayerNorm chứa quá ít tham số; cập nhật chúng dễ làm lệch phân phối đặc trưng
- d) LoRA không hỗ trợ LayerNorm

**Câu hỏi 129** Khi đã fine-tune bằng LoRA và chuyển sang inference, phương thức model.merge\_and\_unload() chủ yếu giúp gì?

- a) Loại bỏ hoàn toàn adapter, trả mô hình về trạng thái ban đầu
- b) Gộp trọng số LoRA vào mô hình gốc, giảm overhead khi tính forward
- c) Khởi tạo lại gradient để có thể tiếp tục fine-tune
- d) Tạo checkpoint mới cho các adapter

**Câu hỏi 130** Khi đóng băng trọng số gốc và áp dụng LoRA, ưu điểm lý thuyết cốt lõi đối với không gian tối ưu hóa là gì?

- a) Phẳng hóa loss để gradient dễ đi hơn
- b) Giới hạn cập nhật vào tiêu không gian hạng thấp, giảm overfitting và catastrophic forgetting
- c) Tăng rank ma trận, cho phép biểu diễn phức tạp hơn
- d) Cho phép tính Hessian trực tiếp với chi phí thấp

**Câu hỏi 131** Phương pháp nào không thuộc vào PEFT?

- a) Adapter Tuning
- b) LoRA
- c) QLoRA
- d) Clustering

**Câu hỏi 132** Bộ dữ liệu nào sử dụng cho Multiple-choice QA là?

- a) MedMCQA
- b) IMDB

- c) C4
- d) ROOTS

## 59.4 Phụ lục

1. **Hint:** Dựa vào file tải về PEFT for Multiple-choice Question Answering để hoàn thiện các đoạn code.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng thứ 3 khi hết deadline phần project, admin mới copy các nội dung bài giải nêu trên vào đường dẫn).

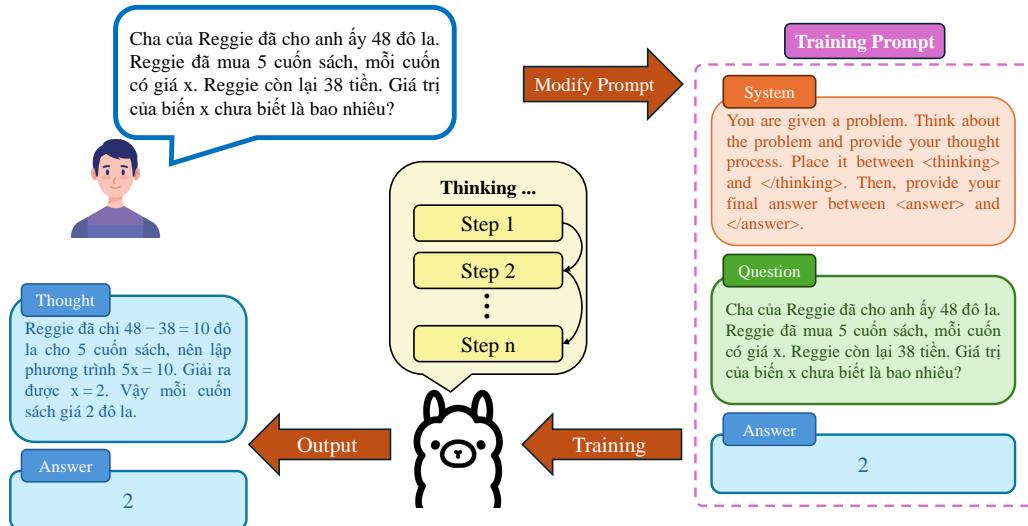
- *Hết* -

# Chương 60

## LLM Reasoning

### 60.1 Giới thiệu

**LLM Reasoning** (Tạm dịch: **LLM Suy Luận**) là khả năng của các mô hình ngôn ngữ lớn trong việc mô phỏng quy trình tư duy logic và giải quyết vấn đề theo từng bước rõ ràng, từ việc trích xuất, liên kết các luận điểm đến diễn giải chi tiết trước khi đưa ra kết quả cuối cùng. Các mô hình như [OpenAI o1](#) hay [DeepSeek-R1](#) đã chứng tỏ thành công ấn tượng khi triển khai khả năng suy luận của mô hình lớn vào nhiều bài toán phức tạp, nâng cao cả độ chính xác lẫn khả năng giải thích của kết quả.



Hình 60.1: Huấn luyện mô hình ngôn ngữ lớn có khả năng suy luận với dữ liệu giải toán Tiếng Việt.

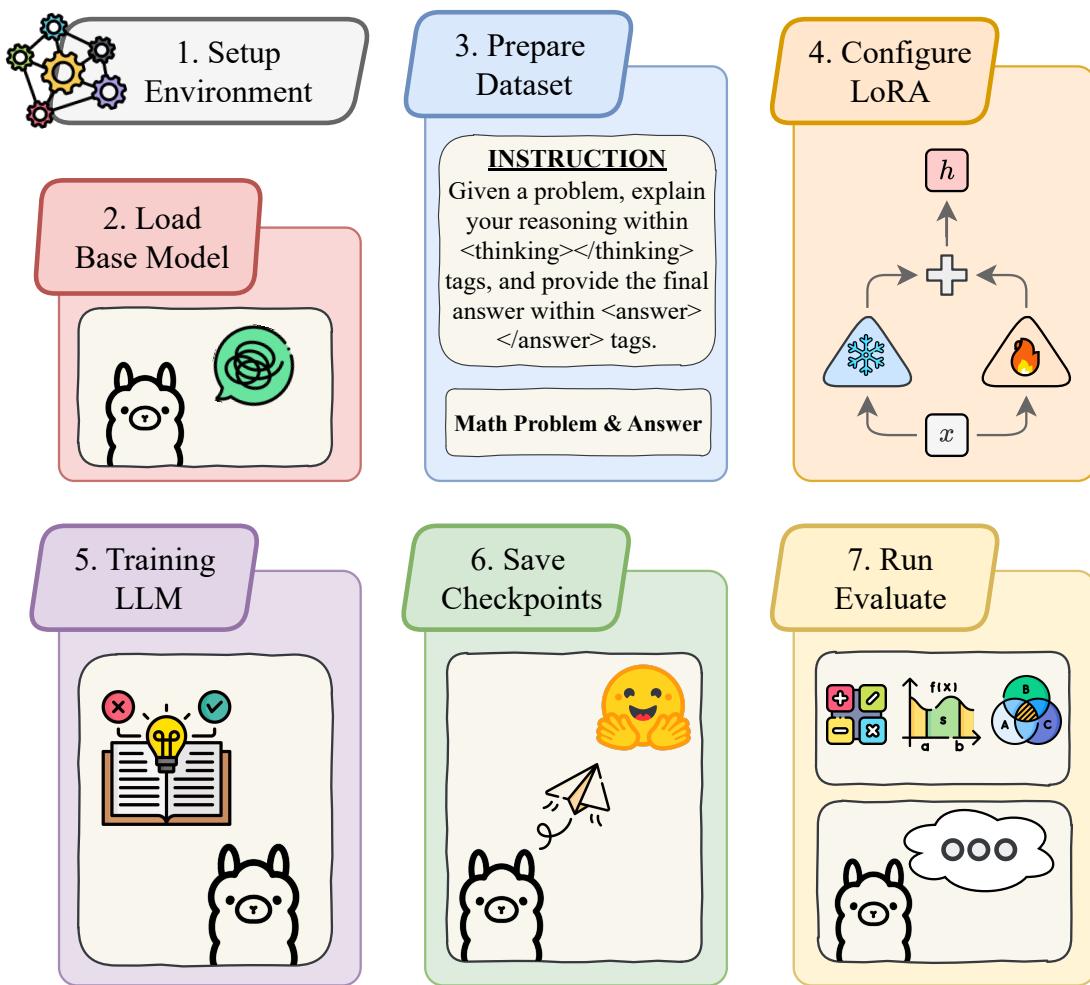
Thay vì chỉ sinh văn bản mạch lạc dựa trên dữ liệu đã học, LLM Reasoning đòi hỏi mô hình “suy nghĩ từng bước” để đưa ra lời giải chính xác và giải thích chi tiết quá trình tư duy, đồng thời cải thiện tính minh bạch và độ

tin cậy trên các nhiệm vụ như giải toán trắc nghiệm, lập luận khoa học hay phân tích ngữ nghĩa.

Trong bài viết này, chúng ta sẽ tìm hiểu cách cài đặt quá trình huấn luyện nhằm tăng cường khả năng suy luận cho một mô hình ngôn ngữ lớn thông qua dữ liệu các câu hỏi toán học. Mục tiêu là xây dựng một mô hình ngôn ngữ lớn chuyên cho việc giải toán trắc nghiệm tiếng Việt, không những có thể đưa ra đáp án đúng mà còn trình bày được chuỗi suy nghĩ logic dẫn đến đáp án đó (dẫn đến việc cải thiện hiệu suất mô hình). Bài toán được định nghĩa với đầu vào và đầu ra như sau:

- **Input:** Nội dung cụ thể của bài toán cần giải quyết.
- **Output:** Lời phản hồi từ mô hình, bao gồm quá trình suy nghĩ và đáp án cho câu hỏi.

Dựa vào nội dung mô tả trên, pipeline các bước xử lý trong bài này được minh họa theo hình dưới đây:



Hình 60.2: Huấn luyện khả năng suy luận cho mô hình ngôn ngữ lớn bằng GRPO.

Trong đó:

1. **Setup Environment:** Thiết lập môi trường làm việc, bao gồm cài đặt và import các thư viện phù hợp cho huấn luyện mô hình.
2. **Load Base Model:** Tải mô hình ngôn ngữ lớn (LLM) gốc từ thư viện như Hugging Face làm nền tảng để tinh chỉnh.
3. **Configure LoRA:** Cấu hình LoRA để giảm tài nguyên huấn luyện

bằng cách khóa các tầng gốc và chỉ tinh chỉnh các adapter nhẹ.

4. **Prepare Dataset:** Chuẩn bị và tiền xử lý dữ liệu training bao gồm câu hỏi và đáp án đúng theo định dạng đầu vào/đầu ra rõ ràng.
5. **Fine-tune LLM:** Huấn luyện mô hình trên tập dữ liệu toán học đã chuẩn bị bằng GRPO và kỹ thuật LoRA nhằm tối ưu hiệu suất.
6. **Save Checkpoints:** Lưu lại các checkpoint của mô hình trong quá trình huấn luyện để theo dõi tiến độ và phục hồi nếu cần.
7. **Run Evaluate:** Thực hiện suy luận trên các câu hỏi để đánh giá khả năng đọc hiểu của mô hình sau tinh chỉnh.

Qua pipeline trên, ta sẽ có một mô hình ngôn ngữ lớn đã được tinh chỉnh với chi phí thấp, có khả năng suy luận và trả lời hiệu quả các câu hỏi toán học.

## 60.2 Cài đặt chương trình

### 60.2.1 Cài đặt và import các thư viện cần thiết

Đầu tiên, chúng ta cần cài đặt và nạp các thư viện hỗ trợ quá trình huấn luyện với GRPO và LoRA trên nền Llama 3.2 1B. Trong đó:

- `unsloth` cung cấp lớp `FastLanguageModel` tối ưu cho inference và fine-tuning.
- `vllm` hỗ trợ việc tạo sinh output nhanh chóng với cấu hình linh hoạt.
- `datasets` dùng để tải và xử lý tập dữ liệu MetaMathQA.
- `trl` bao gồm `GRPOConfig` và `GRPOTrainer` cho module học tăng cường.

Chúng ta thực hiện cài đặt `unsloth` và `vllm` thông qua lệnh `pip install` dưới đây:

```
1 !pip install unsloth vllm==0.7.3
```

Sau khi cài đặt xong, chúng ta import các thư viện cần thiết để thiết lập mô hình, xử lý dữ liệu và cấu hình GRPO:

```
1 import os
2 import re
3
4 from vllm import SamplingParams
5 from unsloth import FastLanguageModel
6 from datasets import load_dataset, Dataset
7 from trl import GRPOConfig, GRPOTrainer
```

### 60.2.2 Load mô hình lớn và setup cài đặt LoRA

Tiếp theo, chúng ta khởi tạo mô hình [Llama 3.2 1B](#) từ `Unsloth` và kích hoạt LoRA để giảm số tham số cần huấn luyện. Lưu ý rằng, các bạn hoàn toàn có thể thay đổi sang các phiên bản mô hình Llama 3.2 với số lượng tham số lớn hơn để có thể đạt một mô hình với hiệu suất cao hơn nếu phần cứng cho phép.

```

1 max_seq_length = 2048
2 lora_rank = 64
3
4 model, tokenizer = FastLanguageModel.from_pretrained(
5 model_name="meta-llama/Llama-3.2-1B-Instruct",
6 max_seq_length=max_seq_length,
7 load_in_4bit=False,
8 fast_inference=True,
9 max_lora_rank=lora_rank,
10 gpu_memory_utilization=0.8,
11)
12
13 model = FastLanguageModel.get_peft_model(
14 model,
15 r=lora_rank,
16 target_modules=[
17 "q_proj", "k_proj", "v_proj", "o_proj",
18 "gate_proj", "up_proj", "down_proj",
19],
20 lora_alpha=lora_rank,
21 use_gradient_checkpointing="unslloth",
22 random_state=3407,
23)

```

- `max_seq_length` định nghĩa độ dài tối đa của input sequence.
- `lora_rank` xác định kích thước của ma trận adapter LoRA.
- `load_in_4bit=False` tắt lượng tử hóa để giữ độ chính xác cho reasoning.
- `fast_inference=True` tối ưu hóa throughput khi sinh text.
- `gpu_memory_utilization=0.8` cho phép sử dụng tối đa 80% bộ nhớ GPU.

### 60.2.3 Tải bộ dữ liệu

Trong bài này, ta sẽ tăng cường khả năng suy luận của mô hình LLM trong việc giải toán trắc nghiệm tiếng Việt. Theo đó, bộ dữ liệu [MetaMathQA-40K](#) phiên bản tiếng Việt sẽ được tải về thông qua đoạn code dưới đây:

```

1 dataset = load_dataset("5CD-AI/Vietnamese-meta-math-MetaMathQA-40K-gg-
translated", split="train")

```

Bài toán	Đáp án
Cha của Reggie đã cho anh ấy \$48. Reggie mua 5 cuốn sách, mỗi cuốn giá \$x và còn lại \$38. Hỏi giá trị của \$x là bao nhiêu?	2
Jean và ba người bạn chơi domino với bộ 28 quân, chia đều cho 4 người. Hỏi mỗi người nhận bao nhiêu quân?	7
Cally có 10 áo trắng, 5 áo màu, 7 quần đùi và 6 quần dài (tổng 28 bộ). Danny có 6 áo trắng, 8 áo màu, 10 quần đùi và 6 quần dài (tổng 30 bộ). Hỏi họ đã giặt bao nhiêu bộ quần áo?	58

Bảng 60.1: Một vài mẫu câu hỏi và đáp án trực tiếp (phần tiếng Việt) trong bộ MetaMathQA-40K.

#### 60.2.4 Cài đặt format prompt cho reasoning

Trước khi huấn luyện, ta cần chuẩn hóa dữ liệu để mô hình học được chuỗi suy nghĩ và đáp án tách biệt. Các bước bao gồm:

1. Dùng regex `answer_pattern` để trích xuất đúng phần “đáp án là...”.
2. Đánh dấu bắt đầu/kết thúc chuỗi suy nghĩ bằng `<thinking>...</thinking>` và phần đáp án bằng `<answer>...</answer>`.
3. Xây dựng `system_prompt` hướng dẫn model sinh reasoning rồi `answer`.
4. Chuyển danh sách thành `train_dataset` với hai trường `prompt` và `answer`.

<|begin\_of\_text|><|start\_header\_id|>system<|end\_header\_id|>

Cutting Knowledge Date: December 2023

Today Date: 28 Apr 2025

You are given a problem.

Think about the problem and provide your thought process.

Place it between <thinking> and </thinking>.

Then, provide your final answer between <SOLUTION></SOLUTION><|eot\_id|>

<|start\_header\_id|>user<|end\_header\_id|>

Cha của Reggie đã cho anh ấy 48 đô la. Reggie đã mua 5 cuốn sách, mỗi cuốn có giá x. Reggie còn lại 38 tiền. Giá trị của biến x chưa biết là bao nhiêu?<|eot\_id|>

<|start\_header\_id|>assistant<|end\_header\_id|>

Hình 60.3: Nội dung đầy đủ chi tiết của một mẫu prompt khi đưa vào mô hình Llama với một số thông tin về việc reasoning cũng như thông tin về bài toán.

```

1 answer_pattern = re.compile(
2 r"(đáp án là:|đáp án là :|câu trả lời là:|câu trả lời là :)\s*(.*)" ,
3 re.IGNORECASE
4)
5
6 formatted_dataset = []
7 for item in dataset:
8 response = item["response_vi"].strip().lower()
9 match = answer_pattern.search(response)
10 if match:
11 answer = match.group(2).strip()
12 formatted_dataset.append({
13 "question": item["query_vi"],
14 "answer": answer
15 })
16
17 reasoning_start = "<thinking>"
18 reasoning_end = "</thinking>"
19 solution_start = "<SOLUTION>"
20 solution_end = "</SOLUTION>"
21

```

```

22 system_prompt = \
23 f """You are given a problem.
24 Think about the problem and provide your thought process.
25 Place it between {reasoning_start} and {reasoning_end}.
26 Then, provide your final answer between {solution_start}{solution_end}"""
27
28 train_dataset = Dataset.from_list(formatted_dataset[:8000])
29 train_dataset = train_dataset.map(lambda x: {
30 "prompt": [
31 {"role": "system", "content": system_prompt},
32 {"role": "user", "content": x["question"]}],
33 "answer": x["answer"],
34 })
35

```

### 60.2.5 Định nghĩa các hàm reward cho học tăng cường

Đối với các thuật toán học tăng cường, việc đánh giá hiệu suất mô hình thường được thông qua các hàm điểm thưởng (reward function). Trong trường hợp này, các hàm reward được cân nhắc triển khai đánh giá chất lượng output theo hai tiêu chí: đúng format reasoning và đúng đáp án. Các bạn hoàn toàn có thể tìm hiểu để đưa ra thêm các tiêu chí điểm thưởng phù hợp.

#### Reward cho output đúng format toán

Đầu tiên, ta tạo ra hai hàm có chức năng tính điểm cho mô hình nếu nó cho ra một format output phù hợp. Ở đây, ta có hàm `match_format_exactly()`, nếu hoàn toàn khớp pattern reasoning + answer, cộng 3.0 điểm. Bên cạnh đó, hàm `match_format_approximately()` sẽ kiểm tra số lượng tag xuất hiện, cộng 0.5 mỗi tag đúng, trừ 1.0 nếu không đúng.

```

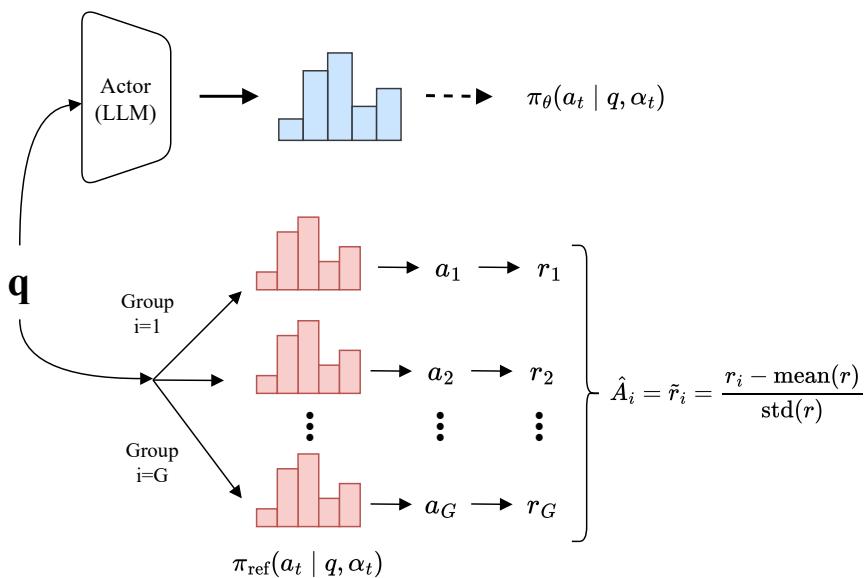
1 match_format = re.compile(
2 rf"^\s*\{\{0,\}\}""
3 rf"\{reasoning_start\}.+\?\{reasoning_end\}.*?""
4 rf"\{solution_start\}(.+?)\{solution_end\}"
5 rf"\s*\{\{0,\}\}\$\",
6 flags=re.MULTILINE | re.DOTALL
7)
8
9 def match_format_exactly(completions, **kwargs):

```

```

10 scores = []
11 for completion in completions:
12 score = 0
13 response = completion[0] ["content"]
14 if match_format.search(response) is not None:
15 score += 3.0
16 scores.append(score)
17 return scores
18
19 def match_format_approximately(completions, **kwargs):
20 scores = []
21 for completion in completions:
22 score = 0
23 response = completion[0] ["content"]
24 score += 0.5 if response.count(reasoning_start) == 1 else -1.0
25 score += 0.5 if response.count(reasoning_end) == 1 else -1.0
26 score += 0.5 if response.count(solution_start) == 1 else -1.0
27 score += 0.5 if response.count(solution_end) == 1 else -1.0
28 scores.append(score)
29 return scores

```



Hình 60.4: Minh họa quy trình GRPO, trong đó LLM đóng vai trò actor sinh phân phối xác suất cho hành động, sau đó lấy mẫu thành nhiều nhóm, đánh giá reward cho từng nhóm và chuẩn hóa kết quả để cập nhật chính sách.

## Reward cho output đúng đáp án

Tiếp theo, ta định nghĩa hàm `check_answer()` để so khớp chuỗi giữa tag `<answer>`, cộng 3.0 nếu đúng, 1.5 nếu chỉ khác khoảng trắng, trừ 1.5 nếu sai hoàn toàn. Cuối cùng, ta có hàm `check_numbers()` với chức năng trích xuất số, so sánh giá trị float, cộng 1.5 nếu đúng, trừ 0.5 nếu sai.

```

1 match_numbers = re.compile(
2 solution_start + r".*?([\d\.\,\,]{1,})",
3 flags=re.MULTILINE | re.DOTALL
4)
5
6 def check_answer(prompts, completions, answer, **kwargs):
7 responses = [completion[0]["content"] for completion in completions]
8
9 extracted_responses = [
10 guess.group(1)
11 if (guess := match_format.search(r)) is not None else None
12 for r in responses
13]
14
15 scores = []
16 for guess, true_answer in zip(extracted_responses, answer):
17 score = 0
18 if guess is None:
19 scores.append(0)
20 continue
21 if guess == true_answer:
22 score += 3.0
23 elif guess.strip() == true_answer.strip():
24 score += 1.5
25 else:
26 score -= 1.5
27 scores.append(score)
28 return scores
29
30 def check_numbers(prompts, completions, answer, **kwargs):
31 question = prompts[0][-1]["content"]
32 responses = [completion[0]["content"] for completion in completions]
33
34 extracted_responses = [
35 guess.group(1)
36 if (guess := match_numbers.search(r)) is not None else None
37 for r in responses
38]

```

```

39 count = getattr(check_numbers, 'counter', 0) + 1
40 check_numbers.counter = count
41 if count % 5 == 0:
42 print('*'*20, f"Question:{question}", f"\nResponse:\n{responses[0]}"
43 },
44 f"\nExtracted: {extracted_responses[0]}", f"\nGT Answer: {"
45 answer[0]}")

45
46 scores = []
47 for guess, true_answer in zip(extracted_responses, answer):
48 if guess is None:
49 scores.append(0)
50 continue
51 try:
52 true_answer = float(true_answer.strip())
53 # Remove commas like in 123,456
54 guess = float(guess.strip().replace(",", ""))
55 scores.append(1.5 if guess == true_answer else -0.5)
56 except:
57 scores.append(0)
58 return scores

```

## 60.2.6 Huấn luyện mô hình

Với tất cả các thông tin trên mà chúng ta đã khai báo, để tiến hành huấn luyện, ta chỉ cần cấu hình và chạy GRPOTrainer để thực hiện training học tăng cường với các hàm reward đã xây dựng:

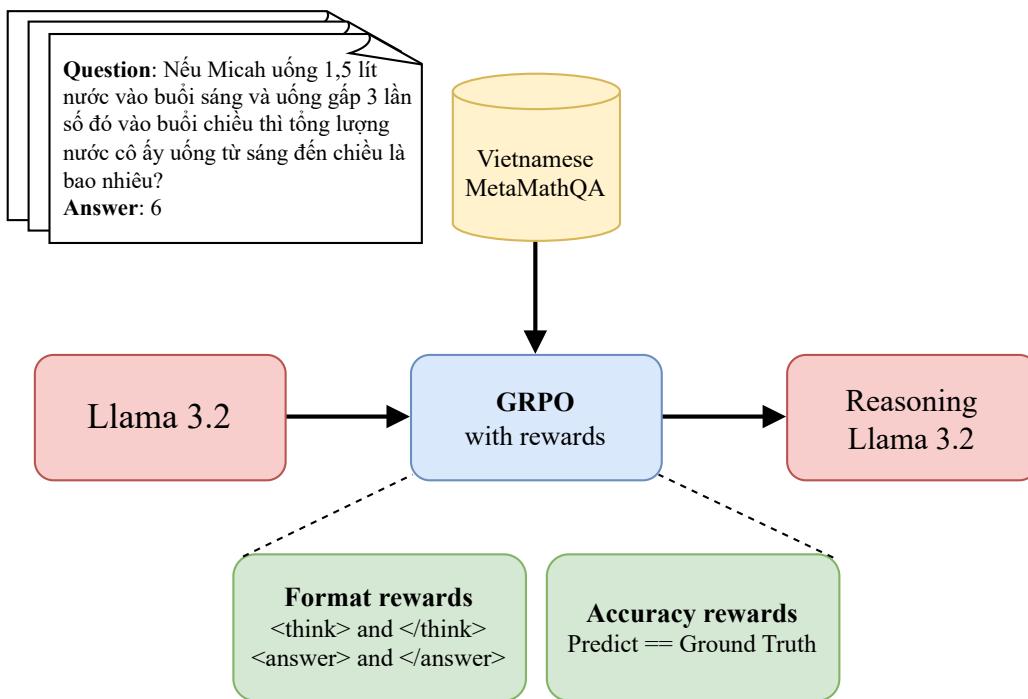
```

1 max_len = max(dataset.map(
2 lambda x: {"tokens": tokenizer.apply_chat_template(
3 x["prompt"], add_generation_prompt=True, tokenize=True)},
4 batched=True,
5).map(lambda x: {"length": len(x["tokens"])})["length"])
6
7 max_prompt_length = max_len + 1
8
9 training_args = GRPOConfig(
10 learning_rate=5e-6,
11 weight_decay=5e-4,
12 warmup_ratio=0.1,
13 lr_scheduler_type="cosine",
14 optim="adamw_torch_fused",
15 logging_steps=1,

```

```
16 per_device_train_batch_size=2,
17 gradient_accumulation_steps=64,
18 num_generations=8,
19 max_prompt_length=max_prompt_length,
20 max_completion_length=max_seq_length - max_prompt_length,
21 num_train_epochs=1,
22 max_steps=-1,
23 save_steps=250,
24 max_grad_norm=0.1,
25 report_to="wandb",
26 output_dir="outputs_bz2",
27)
28
29 trainer = GRPOTrainer(
30 model=model,
31 processing_class=tokenizer,
32 reward_funcs=[
33 match_format_exactly,
34 match_format_approximately,
35 check_answer,
36 check_numbers,
37],
38 args=training_args,
39 train_dataset=dataset,
40)
41 trainer.train()
```

- Tính `max_prompt_length()` dựa trên độ dài token tối đa của prompt đã format.
- Cấu hình `GRPOConfig` với learning rate, weight decay, warmup ratio, scheduler cosine, batch size, số lượng generations, gradient accumulation, và các bước lưu checkpoint.
- Khởi tạo `GRPOTrainer` với danh sách các hàm reward và dataset phù hợp.
- Gọi `trainer.train()` để bắt đầu huấn luyện.



Hình 60.5: Sử dụng thuật toán học tăng cường GRPO nhằm khuyến khích việc đưa ra các suy luận trước khi tạo sinh phương án cuối cùng của mô hình Llama.

### 60.2.7 Lưu trữ và inference

Sau khi huấn luyện xong, ta lưu adapter LoRA đã học và tiến hành thử suy luận trên một ví dụ:

```
1 model.save_lora("grpo_saved_lora")
```

```

1 idx = 0
2 messages = [
3 {"role": "system", "content": system_prompt},
4 {"role": "user", "content": train_dataset[idx]["question"]},
5]
6 sampling_params = SamplingParams(
7 temperature = 0.8,
8 top_p = 0.95,
9 max_tokens = 1024,
10)

```

```

11
12 text = tokenizer.apply_chat_template(
13 messages,
14 add_generation_prompt = True,
15 tokenize = False,
16)
17
18 path_lora = "grpo_saved_lora"
19 output = model.fast_generate(
20 [text],
21 sampling_params = sampling_params,
22 lora_request = model.load_lora(path_lora),
23)[0].outputs[0].text
24
25 print(f"Problem:\n{train_dataset[idx]['question']}")
26 print(f"Response:\n{output}")
27 print("GT Answer:", train_dataset[idx]["answer"])

```

**<thinking>**

Để tìm giá trị của x, ta cần hiểu rằng Reggie đã chi một khoản tiền  $48 - 38 = 10$  đô la để mua các sách. Vì Reggie đã mua 5 cuốn sách với giá x mỗi cuốn nên ta có thể tạo ra một công thức để tính tổng chi phí mua sách:  $5x = 10$ .

Để giải quyết cho x, ta chia cả hai bên của công thức bằng 5:  $x = 10 / 5$ .

Solvers lại ta có:  $x = 2$ .

Vậy giá của mỗi cuốn sách là 2 đô la.

**</thinking>**

**<SOLUTION>2</SOLUTION>**

Hình 60.6: Minh họa một kết quả tạo sinh của mô hình với phần reasoning và đáp án cuối cùng.

Ví dụ minh họa ở Hình 60.6 cho thấy Llama 3.2 1B sau khi được tinh chỉnh với GRPO và LoRA đã có khả năng sinh chuỗi suy nghĩ chi tiết bằng thẻ **<thinking>** trước khi đưa ra đáp án cuối cùng trong thẻ **<SOLUTION>**. Điều này khẳng định tính hiệu quả của việc kết hợp học tăng cường và kỹ thuật

LoRA, không chỉ nâng cao độ chính xác mà còn cải thiện tính giải thích được (explainability) cho các nhiệm vụ yêu cầu tư duy logic chặt chẽ. Theo đó, các bạn hoàn toàn có thể mở rộng phương pháp này cho các bài toán phức tạp hơn hoặc kết hợp thêm các tín hiệu đánh giá đa dạng để tiếp tục tối ưu khả năng suy luận của LLMs.

### 60.3 Câu hỏi trắc nghiệm

1. Theo bài viết, “LLM Reasoning” là gì?
  - a) Khả năng mô hình ngôn ngữ lớn sinh văn bản một cách mäch lạc.
  - b) Khả năng mô hình ngôn ngữ lớn mô phỏng quy trình tư duy logic và giải quyết vấn đề theo từng bước.
  - c) Khả năng mô hình ngôn ngữ lớn dịch thuật giữa các ngôn ngữ.
  - d) Khả năng mô hình ngôn ngữ lớn tóm tắt văn bản dài.
2. Mô hình ngôn ngữ lớn (LLM) gốc nào được sử dụng làm nền tảng trong bài viết?
  - a) OpenAI o1.
  - b) DeepSeek-R1.
  - c) Llama-3.2-1B-Instruct.
  - d) GPT-4.
3. Kỹ thuật nào được sử dụng để giảm số lượng tham số cần huấn luyện, bằng cách khóa các tầng gốc và chỉ tinh chỉnh adapter?
  - a) Quantization (Lượng tử hóa).
  - b) Gradient Checkpointing.
  - c) LoRA (Low-Rank Adaptation).
  - d) GRPO (Group Relative Policy Optimization).
4. Tập dữ liệu nào được sử dụng để huấn luyện mô hình giải toán tiếng Việt?
  - a) GSM8K.
  - b) MATH.
  - c) Vietnamese-meta-math-MetaMathQA-40K-gg-translated.
  - d) Alpaca.
5. Cặp thẻ nào được sử dụng để đánh dấu phần chuỗi suy nghĩ của mô hình trong format prompt?

- a) <logic>...</logic>.
  - b) <reasoning>...</reasoning>.
  - c) <stepbystep>...</stepbystep>.
  - d) <thinking>...</thinking>.
6. Cặp thẻ nào được sử dụng để đánh dấu phần đáp án cuối cùng của mô hình trong format prompt?
- a) <solution>...</solution>.
  - b) <final\_answer>...</final\_answer>.
  - c) <result>...</result>.
  - d) <answer>...</answer>.
7. Trong hàm reward `match_format_exactly()`, mô hình nhận được bao nhiêu điểm nếu output hoàn toàn khớp với pattern reasoning + answer?
- a) 1.0 điểm.
  - b) 1.5 điểm.
  - c) 3.0 điểm.
  - d) 0.5 điểm.
8. Trong hàm reward `check_answer()`, mô hình nhận được bao nhiêu điểm nếu đáp án đoán ra (guess) hoàn toàn trùng khớp (exact match) với đáp án đúng (true\_answer)?
- a) 1.5 điểm.
  - b) 3.0 điểm.
  - c) -1.5 điểm.
  - d) 1.0 điểm.
9. Thuật toán nào được sử dụng để huấn luyện mô hình với các hàm reward đã định nghĩa?
- a) PPO (Proximal Policy Optimization).
  - b) DPO (Direct Preference Optimization).
  - c) GRPO (Group Relative Policy Optimization).

- d) AdamW.
10. Hàm reward `check_numbers()` được thiết kế để đánh giá tiêu chí nào của output?
- a) Kiểm tra output có tuân thủ đúng định dạng thẻ `<thinking>` và `<answer>` hay không.
  - b) So sánh toàn bộ chuỗi văn bản trong thẻ `<answer>` với đáp án gốc.
  - c) Trích xuất và so sánh giá trị số học (dạng float) trong phần đáp án của output với đáp án gốc.
  - d) Đánh giá độ dài và chi tiết của phần giải thích trong thẻ `<thinking>`.

## 60.4 Phụ lục

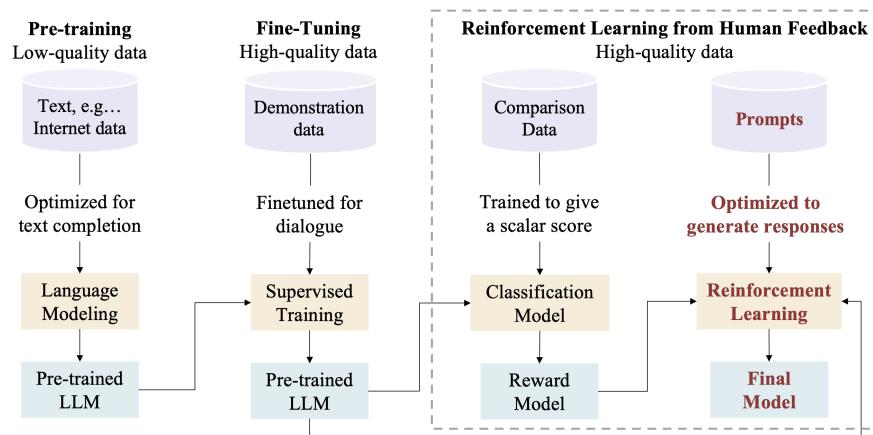
1. **Datasets:** Các file dataset được đề cập trong bài có thể được tải tại [đây](#).
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại [đây](#) (**Lưu ý:** Sáng chủ nhật khi hết deadline phần nội dung này, ad mới copy các tài liệu bài giải vào đường dẫn trên).
3. **Demo:** Web demo và mã nguồn của ứng dụng có thể được truy cập tại [đây](#).
4. **Rubric:**

- *Hết* -

# Chương 61

## Preference Tuning dùng RLHF và DPO

### 61.1 Giới thiệu



Hình 61.1: Reinforcement Learning from Human Feedback.

Trong những năm gần đây, mô hình ngôn ngữ lớn (LLMs) đã cho thấy khả năng ấn tượng, nhưng việc đảm bảo rằng hành vi của mô hình phù hợp với kỳ vọng của con người vẫn là thách thức. RLHF (Reinforcement Learning from Human Feedback) là một phương pháp tiếp cận giúp mô hình học cách tối ưu hóa phản hồi từ con người để cải thiện hành vi.

Khác với fine-tuning truyền thống (chỉ cần dữ liệu đầu ra đúng), RLHF xây dựng một hàm phần thưởng phức tạp phản ánh mức độ hài lòng của người dùng.

Các bước cơ bản:

- Supervised Fine-tuning (SFT): Huấn luyện ban đầu trên tập dữ liệu prompt → expected response bằng cross-entropy.

- Huấn luyện Reward Model: Từ dữ liệu cặp so sánh (prompt, chosen, rejected). Reward Model học đánh giá: chosen phải có điểm cao hơn rejected.

Reward model  $r_\phi$  được tối ưu bằng Binary Cross Entropy giữa sự khác biệt reward của chosen và rejected.

$$\mathcal{L}_{\text{reward}}(\phi) = -\mathbb{E}_{(x, y_c, y_r)} [\log \sigma(r_\phi(x, y_c) - r_\phi(x, y_r))]$$

Trong đó:

$(x, y_c, y_r)$ : Prompt, câu trả lời được chọn, câu trả lời bị từ chối.

$\sigma$  là hàm sigmoid.

- Tối ưu Policy bằng Reinforcement Learning (PPO): Fine-tune mô hình để tối đa hóa reward model output thay vì likelihood ban đầu.

Sau khi có reward model, ta dùng PPO (Proximal Policy Optimization) để update policy  $\pi_\theta$

$$\mathcal{L}_{\text{PPO}}(\theta) = \mathbb{E} \left[ \min \left( \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \hat{A}(s, a), \text{clip} \left( \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}(s, a) \right) \right]$$

Trong đó:

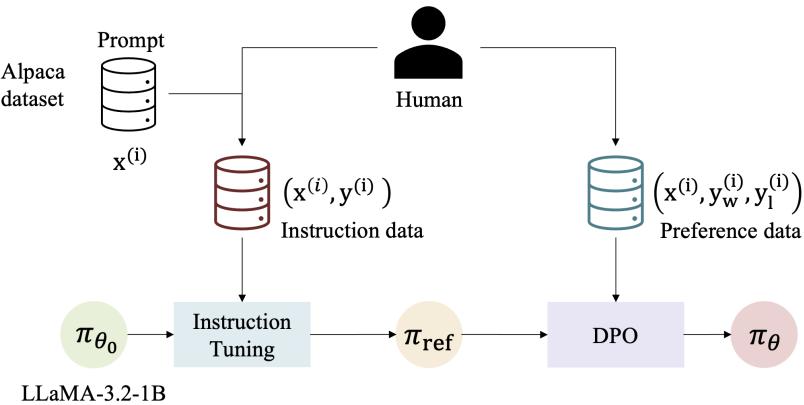
$\hat{A}(s, a)$ : Advantage function, đại diện cho độ tốt của hành động so với trung bình.

$\epsilon$  (ví dụ 0.2) để clip gradient, tránh update quá mạnh.

## Direct Preference Optimization (DPO)

DPO (Direct Preference Optimization) là phương pháp mới nhằm đơn giản hóa RLHF:

- Không huấn luyện reward model trung gian, không cần PPO.
- DPO tối ưu trực tiếp mô hình sinh ra đầu ra được người dùng ưa thích hơn.



Hình 61.2: Direct Preference Optimization.

Ý tưởng: nếu ta có (prompt, chosen, rejected), ta chỉ cần điều chỉnh policy sao cho:

$$P(chosen) > P(rejected)$$

Công thức tính hàm loss:

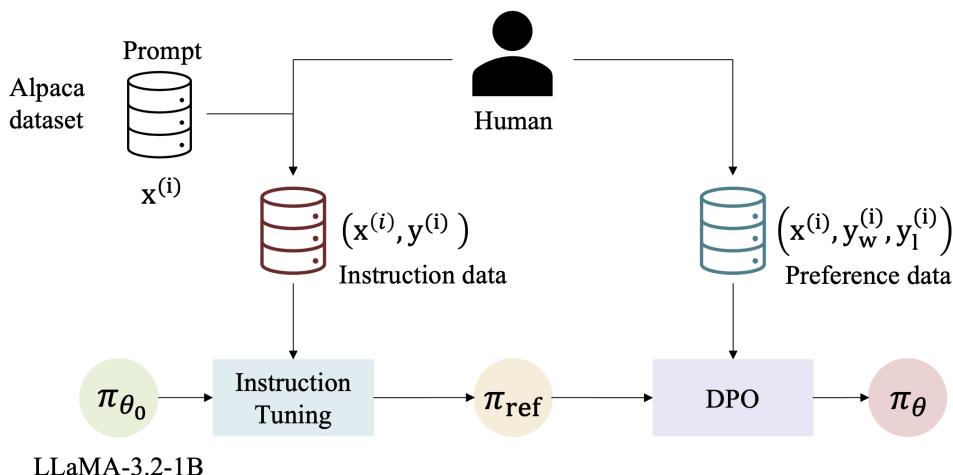
$$\mathcal{L}_{DPO}(\theta) = -\mathbb{E}_{(x, y_c, y_r)} \left[ \log \left( \frac{\exp(\beta \log \pi_\theta(y_c|x))}{\exp(\beta \log \pi_\theta(y_c|x)) + \exp(\beta \log \pi_\theta(y_r|x))} \right) \right]$$

Hoặc được viết lại:

$$\mathcal{L}_{DPO}(\theta) = -\mathbb{E} [\log \sigma (\beta (\log \pi_\theta(y_c|x) - \log \pi_\theta(y_r|x)))]$$

## 61.2 DPO for Vietnamese Custom Preference Dataset

Quá trình huấn luyện DPO cho một mô hình ngôn ngữ lớn gồm các bước, được minh họa trong hình



Hình 61.3: DPO pipeline ([source](#)).

Quá trình huấn luyện mô hình như sau:

## 1. Cài đặt thư viện và tải về bộ dữ liệu

Đầu tiên là cài đặt một số thư viện để có thể chạy được các mô hình ngôn ngữ lớn từ thư viện **transformers**. Bộ dữ liệu được sử dụng là bộ dữ liệu Alpaca từ Stanford đã được dịch sang tiếng Việt và chuyển về format { **question**, **chosen**, **rejected** }. Bộ dữ liệu được lưu trữ ở đây: [thaing107/Vi-Alpaca-Preference](#).

```
1 # Install libs
2 !pip install -q datasets trl peft bitsandbytes
3 sentencepiece
4
5 from datasets import load_dataset
6
7 dataset = load_dataset("thainq107/Vi-Alpaca-Preference",
8 cache_dir=CACHE_DIR)
```

```
7 dataset, dataset["train"][0]
```

Thông tin về bộ dữ liệu và một data point của bộ dữ liệu được in ra như sau:

```
1 (DatasetDict({
2 train: Dataset({
3 features: ["id", "question", "chosen", "rejected"
4],
5 num_rows: 65017
6 })
7 test: Dataset({
8 features: ["id", "question", "chosen", "rejected"
9],
10 num_rows: 2000
11 })
12 }),
13 {"id": "dolly-14239",
14 "question": ...,
15 "chosen": ...,
16 "rejected": ...
17 })
```

## 2. Modeling

Mô hình LLaMA-3.2-1B-Instruct được tải từ thư viện `transformers` với kỹ thuật lượng tử hóa 4-bit từ thư viện `bitsandbytes`, giúp giảm dung lượng và tối ưu hóa hiệu suất tính toán trên GPU.

```
1 # Model to fine-tune
2 model_name = "thainq107/Llama-3.2-1B-Instruct-sft"
3 cache_dir = "./cache"
4
5 bnb_config = BitsAndBytesConfig(
6 load_in_4bit=True,
7 bnb_4bit_quant_type="nf4",
8 bnb_4bit_compute_dtype=torch.bfloat16,
9 bnb_4bit_use_double_quant=True,
10)
11
12 base_model = AutoModelForCausalLM.from_pretrained(
13 BASE_MODEL_ID,
14 trust_remote_code=True,
15 device_map={':': torch.cuda.current_device()},
16 token=hf_token,
17 cache_dir=CACHE_DIR,
```

```

18 torch_dtype=torch.bfloat16,
19 quantization_config=bnb_config,
20)
21 base_model.config.use_cache = False
22
23 tokenizer = AutoTokenizer.from_pretrained(BASE_MODEL_ID,
24 cache_dir=CACHE_DIR, trust_remote_code=True)
25 if tokenizer.pad_token is None:
26 tokenizer.pad_token = tokenizer.eos_token
27 tokenizer.pad_token_id = tokenizer.eos_token_id

```

Tiếp theo, kỹ thuật LoRA được áp dụng lên nhiều thành phần của mô hình, bao gồm các modules attention (q\_proj, k\_proj, v\_proj, o\_proj), các modules feed-forward network (up\_proj, down\_proj), và module gating (gate\_proj).

```

1 # QLoRA configuration
2 peft_config = LoraConfig(
3 r=16,
4 lora_alpha=16,
5 lora_dropout=0.05,
6 bias="none",
7 task_type="CAUSAL_LM",
8 target_modules=[
9 "q_proj",
10 "k_proj",
11 "v_proj",
12 "o_proj",
13 "gate_proj",
14 "up_proj",
15 "down_proj"
16]
17)

```

Bên cạnh đó, gradient checkpointing được kích hoạt giúp tiết kiệm bộ nhớ trong quá trình huấn luyện mà không làm giảm hiệu suất.

### 3. Preprocessing

Để huấn luyện mô hình, ta sử dụng chat template có sẵn của Llama3.2 như sau:

```

1 conversation = [
2 {"role": "system", "content": "You are a helpful
assistant."},

```

```

3 {"role": "user", "content": "This is the prompt"},

4 {"role": "assistant", "content": "This is the chosen"

5 },

6 print(tokenizer.apply_chat_template(

7 conversation, tokenize=False, add_generation_prompt=

8 False

9))

```

và đây là kết quả khi in ra màn hình cấu trúc của prompt:

```

1 """<|begin_of_text|><|start_header_id|>system<|

2 end_header_id|>

3

4 Cutting Knowledge Date: December 2023

5 Today Date: 20 Apr 2025

6

7 You are a helpful assistant.<|eot_id|><|start_header_id|>

8 user<|end_header_id|>

9

10 This is the prompt<|eot_id|><|start_header_id|>assistant

 <|end_header_id|>

11

12 This is the chosen<|eot_id|>"""

```

#### 4. Training

Quá trình train DPO gồm 3 bước: pre-training, supervised fine-tuning, và policy optimizing. Khi chúng ta sử dụng mô hình Llama3.2-1B-Instruct, mô hình đã trải qua bước pre-training (và supervised fine-tuning). Do đó, ta cần huấn luyện lại mô hình trên preference dataset của mình ở hai bước supervised fine-tuning và policy optimizing.

Đầu tiên, ta thiếu lập các hyperparameters như sau:

```

1 hyperparameters = {

2 "per_device_train_batch_size": 2,

3 "gradient_accumulation_steps": 8,

4 "gradient_checkpointing": True,

5 "learning_rate": 3e-5,

6 "logging_steps": 500,

7 "max_steps": 5_000,

8 "save_strategy": "no",

9 "overwrite_output_dir": True,

10 "optim": "paged_adamw_8bit",

11 "warmup_steps": 500,

```

```
12 "bf16": True,
13 }
14 MAX_LENGTH = 512
```

Trong đó

- `per_device_train_batch_size = 2`: Số lượng mẫu dữ liệu được xử lý trên mỗi thiết bị (ví dụ mỗi GPU) trong một bước huấn luyện.
- `gradient_accumulation_steps = 8`: Tích lũy gradient trong 8 bước trước khi thực hiện cập nhật trọng số, giúp mô phỏng một batch lớn mà không cần dùng quá nhiều bộ nhớ.
- `gradient_checkpointing = True`: Kích hoạt cơ chế lưu trữ trung gian giúp giảm sử dụng bộ nhớ GPU bằng cách tính lại một số phần trong quá trình backpropagation.
- `learning_rate = 3e-5`: Tốc độ học (learning rate) của mô hình – là hệ số quyết định mức độ cập nhật trọng số sau mỗi bước huấn luyện.
- `logging_steps = 500`: Ghi lại log (ví dụ loss, learning rate,...) sau mỗi 500 bước huấn luyện để theo dõi tiến trình.
- `max_steps = 5000`: Tổng số bước huấn luyện sẽ được thực hiện.
- `save_strategy = "no"`: Không lưu mô hình tự động trong quá trình huấn luyện. Việc lưu mô hình cần được thực hiện thủ công hoặc bằng đoạn mã riêng.
- `overwrite_output_dir = True`: Cho phép ghi đè lên thư mục đầu ra nếu đã tồn tại nội dung từ các lần huấn luyện trước.
- `optim = "paged_adamw_8bit"`: Sử dụng trình tối ưu hoá AdamW phiên bản 8-bit kết hợp với phân trang (paged) để tiết kiệm bộ nhớ và phù hợp với mô hình lớn.
- `warmup_steps = 500`: Số bước warm-up, tức là giai đoạn khởi động với learning rate tăng dần từ nhỏ đến đầy đủ trong 500 bước đầu tiên.
- `bf16 = True`: Sử dụng định dạng số học **bfloat16** để tăng hiệu suất tính toán và giảm bộ nhớ mà vẫn giữ được độ chính xác tương đối tốt.

- MAX\_LENGTH = 512: Chiều dài tối đa của chuỗi đầu vào mà mô hình có thể xử lý, thường áp dụng cho tokenized input.

Ta chạy supervised fine-tuning như sau:

```

1 def formatting_prompt_with_chat_template(example):
2 conversation = [
3 {"role": "system", "content": "You are a helpful
4 assistant."},
5 {"role": "user", "content": example["question"]},
6 {"role": "assistant", "content": example["chosen"]
7],
8]
9 prompt = tokenizer.apply_chat_template(
10 conversation, tokenize=False,
11 add_generation_prompt=False
12)
13 return prompt
14
15 SFT_OUTPUT_DIR = f"{ROOT_OUTPUT_DIR}-SFT"
16
17 sft_config = SFTConfig(
18 **{
19 "hyperparameters", "output_dir": SFT_OUTPUT_DIR ,
20 "max_seq_length": MAX_LENGTH
21 }
22)
23 sft_trainer = SFTTrainer(
24 model=base_model,
25 peft_config=peft_config,
26 processing_class=tokenizer,
27 args=sft_config,
28 train_dataset=dataset["train"],
29 formatting_func=formatting_prompt_with_chat_template
30)
31
32 sft_trainer.train()

```

Khi các tham số huấn luyện đã được cấu hình, mô hình huấn luyện được bắt đầu với SFTTrainer và Các tham số huấn luyện được truyền vào qua đối tượng args.

Cuối cùng, sau khi kết thúc huấn luyện bước supervised fine-tuning này, ta lưu mô hình như sau:

```

1 sft_trainer.save_model(SFT_OUTPUT_DIR)
2 sft_trainer.push_to_hub(f"{HF_USER}/{MODEL_NAME}-SFT",
3 token=hf_token) # hf_token is your huggingface token

```

Sau khi huấn luyện mô hình với supervised fine-tuning, ta tiếp tục bước policy optimization.

Ta convert dữ liệu về conversational format:

```

1 def convert_to_conversational_preference_format(example):
2 return {
3 "id": example["id"],
4 "prompt": [{"role": "system", "content": "You are
5 a helpful assistant."}, {"role": "user", "content":
example["question"]}],
6 "chosen": [{"role": "assistant", "content":
example["chosen"]}],
7 "rejected": [{"role": "assistant", "content":
example["rejected"]}],
8 }
9
dpo_dataset = dataset.map(
 convert_to_conversational_preference_format)

```

Tương tự như supervised fine-tuning, ta convert dataset về conversational format.

Tiếp theo, ta load LoRA weights đã huấn luyện với supervised fine-tuning để tiếp tục quá trình policy optimization với DPOTrainer.

```

1 dpo_full_model = base_model.load_adapter(
2 SFT_OUTPUT_DIR, is_trainable=True, adapter_name="
3 dpo_full_adapter"
3)

```

Và cuối cùng, chúng ta dùng DPOTrainer để huấn luyện mô hình:

```

1 DPO_FULL_OUTPUT_DIR = f"{ROOT_OUTPUT_DIR}-DPO-full"
2 dpo_full_args = DPOConfig(
3 **{
4 **hyperparameters,
5 "output_dir": DPO_FULL_OUTPUT_DIR,
6 "max_length": MAX_LENGTH
7 }
8)
9
10 dpo_full_trainer = DPOTrainer(
11 dpo_full_model,
12 args=dpo_full_args,
13 train_dataset=dpo_dataset["train"],
14 processing_class=tokenizer,
15 peft_config=peft_config,
16)
17
18 dpo_full_trainer.train()

```

## 5. Save Models

Khi quá trình huấn luyện hoàn tất, mô hình được lưu để sử dụng hoặc chia sẻ sau này. Cụ thể:

```
1 dpo_full_trainer.save_model(DPO_FULL_OUTPUT_DIR)
2 dpo_full_trainer.push_to_hub(f"{HF_USER}/{MODEL_NAME}-DPO
 -full", token=hf_token)
```

- Mô hình sau huấn luyện được lưu cục bộ vào thư mục DPO\_FULL\_OUTPUT\_DIR bằng hàm `save_model()`.
- Sau đó, mô hình được đẩy lên Hugging Face Hub thông qua hàm `push_to_hub()`, giúp dễ dàng chia sẻ và tái sử dụng trong các dự án khác.
- Để thực hiện bước này, người dùng cần có tài khoản trên Hugging Face và phải cung cấp HuggingFace Token để xác thực quyền truy cập.

## 6. Inference

Sau khi huấn luyện, sử dụng mô hình để thực hiện suy luận (inference) trên một ví dụ như sau:

```
1 def get_model_response(model, tokenizer, instruction):
2 cur_conversation = [
3 {"role": "system", "content": "You are a helpful
4 assistant."},
5 {"role": "user", "content": instruction}
6]
7 cur_input_prompt = tokenizer.apply_chat_template(
8 cur_conversation, add_generation_prompt=True, tokenize
9 =True)
10 cur_output_ids = model.generate(input_ids=torch.
11 LongTensor([cur_input_prompt]).to(model.device),
12 max_new_tokens=1000)
13 cur_generated_ids = cur_output_ids[0][len(
14 cur_input_prompt):]
15 return tokenizer.decode(cur_generated_ids,
16 skip_special_tokens=True)
```

## 7. Deployment

Sau khi huấn luyện mô hình, triển khai ứng dụng sử dụng thư viện Gradio của Huggingface như sau: [Space](#)

### Vietnamese Custom Preference Model

Model: LLaMA-3.2-1B. Dataset: Alpaca-Vi

Input

Clear
Submit

Output

Trí tuệ nhân tạo (Artificial Intelligence, AI) là một loại công nghệ dựa trên hệ thống học máy, trong đó con người sử dụng các thuật toán để tạo ra các mô hình và hệ thống có thể tự động thực hiện các nhiệm vụ, quyết định và hành động theo hướng logic và sáng tạo. Nó có thể được sử dụng trong nhiều lĩnh vực, bao gồm nhưng không giới hạn ở kinh tế, y tế, giáo dục, bảo vệ an ninh, và nhiều hơn nữa.

Share via Link

Hình 61.4: Deployment.

### 61.3 Câu hỏi trắc nghiệm

**Câu hỏi 133** Trong RLHF, vai trò của Reward Model là gì?

- a) Sinh dữ liệu mới cho mô hình
- b) Đánh giá mức độ phù hợp của phản hồi
- c) Thay thế hoàn toàn Loss Function
- d) Tăng tốc quá trình training

**Câu hỏi 134** Khi huấn luyện Reward Model, loss function phổ biến là gì?

- a) Triplet Loss
- b) Cross-Entropy Loss
- c) Margin Loss giữa chosen và rejected
- d) Reconstruction Loss

**Câu hỏi 135** PPO trong RLHF có vai trò gì?

- a) Tối ưu likelihood sinh phản hồi
- b) Tối ưu trực tiếp Reward Model
- c) Cập nhật policy model để tối đa hóa reward
- d) Dự đoán xác suất từ prompt

**Câu hỏi 136** Ưu điểm nổi bật của DPO so với RLHF là gì?

- a) Huấn luyện nhanh hơn và ổn định hơn
- b) Sinh ra phản hồi dài hơn
- c) Tối ưu tốt hơn reward tuyệt đối

d) Cần ít dữ liệu hơn để huấn luyện reward model

**Câu hỏi 137** DPO loss chủ yếu dựa trên loại hàm nào?

- a) Mean Squared Error
- b) Margin Ranking Loss
- c) Sigmoid Binary Cross Entropy
- d) KL Divergence

**Câu hỏi 138** Công thức Advantage trong PPO được tính như thế nào?

- a)  $A(s, a) = r - V(s)$
- b)  $A(s, a) = V(s) - Q(s, a)$
- c)  $A(s, a) = \log \pi(a|s) - \log \pi_{\text{old}}(a|s)$
- d)  $A(s, a) = r(s, a)$

**Câu hỏi 139** Trong DPO, tham số  $\beta$  dùng để làm gì?

- a) Làm loss function smooth hơn
- b) Điều chỉnh độ sắc nét giữa các xác suất
- c) Kiểm soát learning rate
- d) Giảm memory usage trong training

**Câu hỏi 140** Đặc điểm chung của DPO và RLHF?

- a) Đều cần human feedback data
- b) Đều sử dụng reward model
- c) Đều yêu cầu reward số tuyệt đối
- d) Đều yêu cầu PPO cho policy update

**Câu hỏi 141** Trong DPO, nếu  $\beta$  tiến về 0, loss function sẽ gần giống với:

- a) Cross-entropy Loss giữa chosen và rejected
- b) Margin Ranking Loss với margin = 1
- c) Uniform random loss
- d) KL Divergence loss giữa two outputs

**Câu hỏi 142** Nếu reward model huấn luyện chưa tốt, điều gì xảy ra với RLHF?

- a) Policy học hành vi sai lệch
- b) Policy convergence nhanh hơn

- c) Mô hình đạt loss thấp hơn
- d) Không ảnh hưởng nhiều vì PPO cân bằng lại

## 61.4 Phụ lục

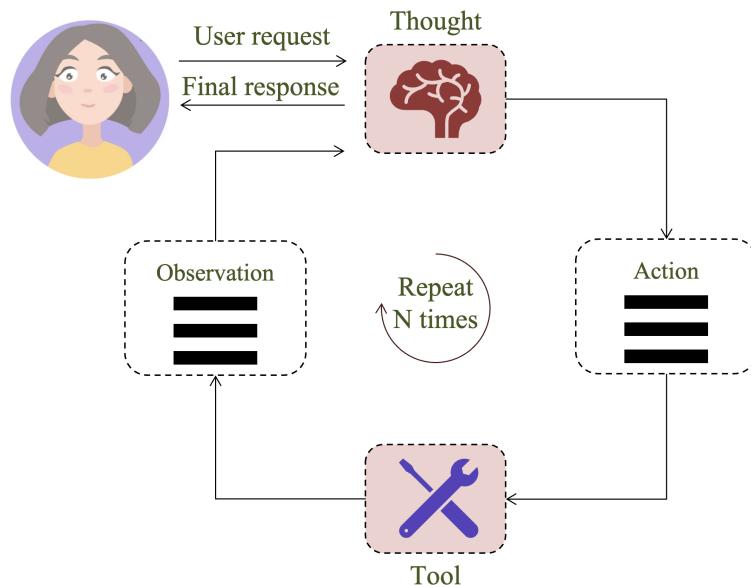
1. **Code:** Thư mục chứa [CODE](#)
2. **Code:** Tài liệu code trên [GITHUB](#)
3. **Rubric:**

- *Hết* -

# Chương 62

## ReAct Agent

### 62.1 Giới thiệu



Hình 62.1: ReAct Agent using LangGraph.

Mô hình ngôn ngữ lớn (LLM) đã tạo ra một bước đột phá trong khả năng xử lý ngôn ngữ tự nhiên. Tuy nhiên, để ứng dụng hiệu quả vào các tác vụ thực tế đòi hỏi khả năng tư duy nhiều bước, hành động có mục tiêu và phản ứng linh hoạt, chúng ta cần triển khai AI agents – những cấu trúc có khả năng lập kế hoạch, hành động, quan sát và suy luận. Hiện nay có nhiều thiết kế cho phép các Agent có khả năng tương tác và suy luận thông minh để hoàn thành mục tiêu, điển hình như ReAct, Reflexion, Plan-and-Execute hoặc Multi-Agents.

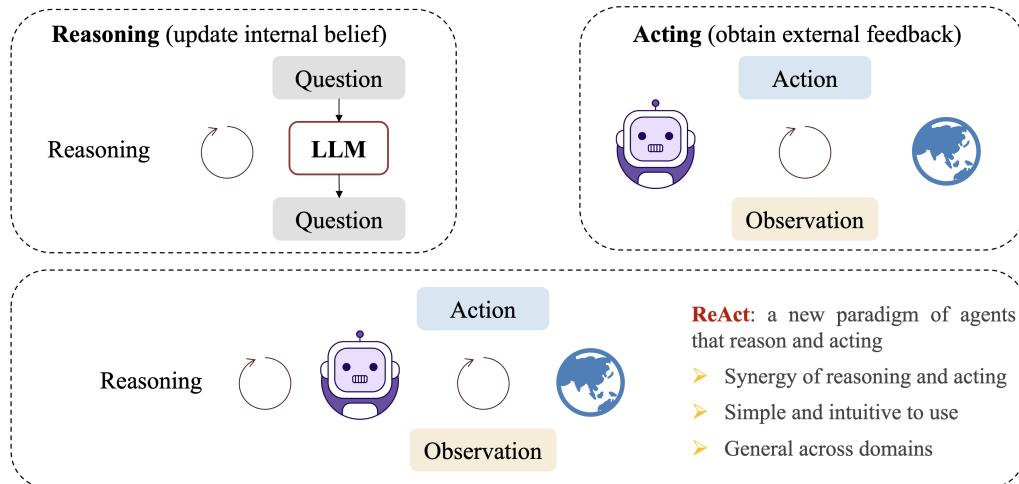
Trong phần nội dung này, chúng ta sẽ tìm hiểu hai kiến trúc reasoning agents nổi bật hiện nay là:

- **ReAct:** Reasoning and Acting – kết hợp suy nghĩ và hành động qua nhiều vòng lặp.
- **Plan-and-Execute:** tách bạch hoàn toàn giữa giai đoạn lên kế hoạch và thực thi.

Sau đó sẽ triển khai các agents thông qua thư viện LangGraph.

## 62.2 ReAct Agent

ReAct (Reasoning + Acting) là một phương pháp kết hợp giữa suy luận và hành động, cho phép các mô hình ngôn ngữ lớn (LLMs) vừa lý luận về các nhiệm vụ vừa thực hiện hành động liên quan đến nhiệm vụ đó. ReAct lần đầu tiên được giới thiệu là phương pháp tận dụng khả năng Reasoning của LLMs để giải quyết những tác vụ phức tạp trong bài báo: [ReAct: Synergizing Reasoning and Acting in Language Models](#)



Hình 62.2: ReAct là sự kết hợp ưu điểm của khả năng Reasoning + Acting LLMs.

Các mô hình chỉ có reasoning (suy luận) thường dựa hoàn toàn vào khả năng ngôn ngữ của LLM để tạo ra lời giải. Mặc dù chúng có thể mô phỏng tư duy logic, nhưng không thể tương tác với thế giới bên ngoài, như tìm kiếm thông tin mới, gọi công cụ, hay xác minh giả định. Điều này khiến chúng dễ bị giới hạn trong những gì đã được học trong quá trình huấn luyện, và dễ sinh ra câu trả lời “có vẻ hợp lý nhưng sai”.

Ngược lại, các mô hình chỉ có acting (hành động) – ví dụ như các hệ thống gán tool cho prompt một cách trực tiếp – có thể thực thi tác vụ nhanh chóng, nhưng lại thiếu năng lực suy luận để quyết định khi nào nên gọi tool nào, hoặc nên dừng lại khi đã đủ thông tin. Chúng có thể bị “ngây thơ”, thực hiện hành động không cần thiết, hoặc không biết xử lý khi tool trả về kết quả không mong muốn.

ReAct kết hợp cả hai thế giới: Nó cho phép mô hình “nghĩ thành tiếng” trước khi ra quyết định hành động, và sau khi hành động xong, quan sát kết quả để điều chỉnh suy luận tiếp theo. Điều này tạo ra một vòng lặp Thought → Action → Observation, giúp mô hình:

- Hiểu rõ về bối cảnh trước khi hành động
- Tự sửa sai nếu kết quả quan sát không như mong đợi
- Chọn lựa hành động một cách có lý do, thay vì theo rule cố định
- Kết hợp nhiều công cụ linh hoạt, ví dụ: tra cứu → tính toán → lập kế hoạch → trả lời

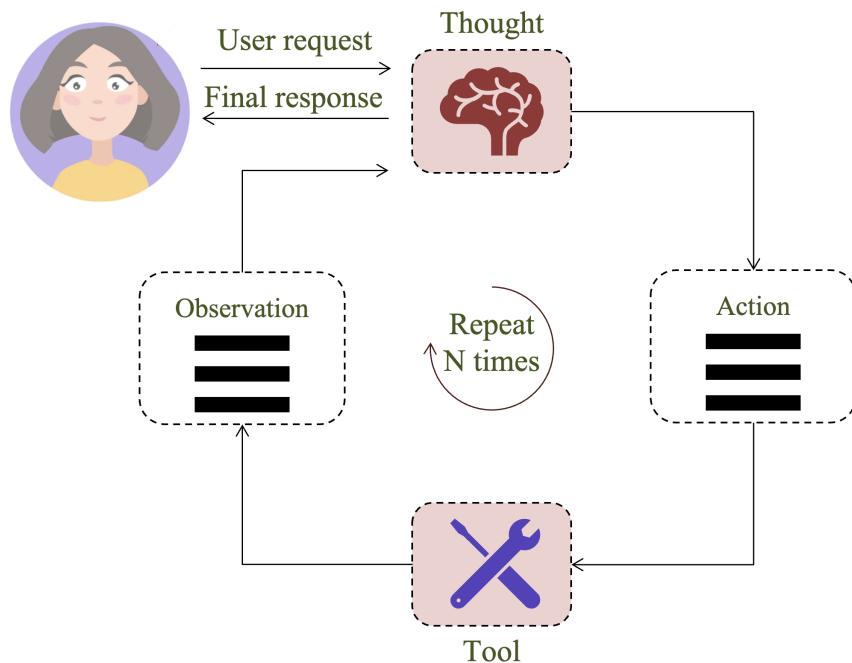
Nói cách khác, ReAct giống như một con người đang giải một bài toán: suy nghĩ, thử hành động, nhìn kết quả, rồi tiếp tục suy nghĩ – chứ không đơn thuần là “trả lời ngay” hay “làm ngay”.

Phương pháp này hoạt động dựa trên việc kết hợp:

- Reasoning (Suy luận): Mô hình suy nghĩ về vấn đề, phân tích tình huống và lập kế hoạch cho hành động tiếp theo.
- Acting (Hành động): Mô hình thực hiện các hành động dựa trên suy luận của mình, thu thập thông tin mới và tiếp tục quá trình.

ReAct thực hiện một quy trình lặp đi lặp lại gồm các bước sau:

- Suy luận (Thought): Mô hình suy nghĩ về tình huống hiện tại, phân tích các thông tin đã có và cân nhắc các bước tiếp theo. Mục tiêu để hoàn thiện tác vụ đặt ra và hiểu rõ khả năng hoàn thiện các tác vụ cho mỗi chu trình.
- Hành động (Action): Dựa trên suy luận, mô hình thực hiện một hành động cụ thể (ví dụ: tìm kiếm thông tin, gọi API, thực hiện tính toán). Vì vậy, việc định nghĩa và xác định các công cụ hữu ích cho quá trình thực hiện sẽ tối ưu tính hiệu quả khi xử lý các tác vụ khác nhau của Agent.



Hình 62.3: ReAct Agent sử dụng thư viện LangGraph.

- Quan sát (Observation): Sau khi thực hiện hành động, mô hình quan sát kết quả thu được. Dựa trên kết quả hành động và yêu cầu cho hành động để gửi thông điệp xem xét tính hiệu quả của mỗi chu trình thực hiện.
- Tiếp tục chu trình: Quay lại bước suy luận với thông tin và quan sát mới, tiếp tục quá trình cho đến khi đạt được mục tiêu. Số lượng chu trình thực hiện có thể được ước lượng cho trước để đảm bảo Agent có thể dừng chu trình suy luận trong trường hợp chưa tìm được câu trả lời phù hợp.

Ví dụ về quá trình suy luận của ReAct thực hiện để giải quyết tác vụ phức tạp được mô tả như hình sau:

**Trong phần này, chúng ta sẽ sử dụng thư viện LangGraph để xây dựng ReAct Agent**

### 1. Cài đặt thư viện

You are in the middle of a room. Looking quickly around you, you see a cabinet 6, a cabinet 1, a coffee machine 1, a countertop 3, a stove burner 1, and a toaster 1.  
**Your task is to:** Put some pepper shaker on a drawer.

ReAct

```

Act 1: Think[First I need to find a peppe shaker...more likely to appear in cabinets (1-6),
countertops (1-3), ...]
Act 2: Go to cabinet 1
Obs 2: On the cabinet 1, you see a vase 2.
(Here the agent goes to cabinet 1, 2, 3, 4, 5, 6, then countertop 1 and 2)

Act 9: Go to countertop 3
Obs 9: On the countertop 3, you see a apple 1, a bread 1, a pepper shaker 1, and a vase 3.
Act 10: Take pepper shaker 1 from countertop 3
Obs 10: You pick up the pepper shaker 1 from the countertop 3.
Act 11: Think[Now I find a pepper shaker 1. Next, I need to put it in/on drawer 1.]
Act 12: Go to drawer 1
Obs 12: Drawer 1 is closed.
Act 13: Open drawer 1
Obs 13: You open Drawer 1 ...
Act 14: Put pepper shaker 1 in/on drawer 1
Obs 14: You put pepper shaker 1 in/on the drawer 1.

```



Hình 62.4: Ví dụ về quá trình suy luận của ReAct.

Cài đặt một số thư viện và chuẩn bị các Key cho quá trình sử dụng LLMs cũng như lưu trữ thông tin thông qua công cụ LangSmith. Sau đó xây dựng lớp AgentState để thực thi gồm các thuộc tính như sau:

- messages: lưu trữ các thông tin trong quá trình thực hiện
- steps: để xác định số chu trình đã thực hiện

```

1 # Install libs
2 !pip install -U langchain==0.3.24 langchain-openai
 ==0.3.14 langgraph==0.3.33 langchain-tavily==0.1.6
3
4 import os
5 os.environ["LANGCHAIN_API_KEY"] = ### Your-key
6 os.environ["LANGCHAIN_PROJECT"] = "Reasoning-Agent"
7 os.environ["LANGCHAIN_TRACING_V2"] = "true"
8 os.environ["TAVILY_API_KEY"] = ### Your-key
9 os.environ["OPENAI_API_KEY"] = ### Your-key
10
11 from typing import Annotated, Sequence, TypedDict
12
13 from langchain_core.messages import BaseMessage
14 from langgraph.graph.message import add_messages
15
16 class AgentState(TypedDict):
17 """The state of the agent."""
18 messages: Annotated[Sequence[BaseMessage], add_messages]

```

```
19 number_of_steps: int
```

## 2. Định nghĩa các công cụ

Trong phần này, chúng ta xây dựng 2 công cụ để thực hiện hành động là tìm kiếm và thực hiện phép tính nhân. Bên cạnh đó, chúng ta sử dụng mô hình chatGPT để thực hiện.

```
1 from langchain_tavily import TavilySearch
2 from langchain_core.tools import tool
3 from langchain_openai import ChatOpenAI
4 from langchain_core.messages import SystemMessage
5
6 @tool
7 def triple(num: float) -> float:
8 """
9 :param num: a number to triple
10 :return: the number tripled -> multiplied by 3
11 """
12 return 3 * float(num)
13
14
15 tools = [TavilySearch(max_results=1), triple]
16
17 system_prompt = SystemMessage(
18 """
19 You are a reasoning agent. Always think step-by-step.
20
21 Use the following format:
22
23 Thought: what you are thinking
24 Action: the action to take, e.g. 'search', 'calculate'
25 Action Input: the input to the action
26 Observation: the result of the action
27
28 (Repeat Thought/Action/Observation if needed)
29
30 Final Answer: your answer to the user
31
32 Question: {input}
33 """
34)
35
36 model = ChatOpenAI(model="gpt-4.1-nano")
```

### 3. Xây dựng các hàm thực hiện các tác vụ

Sau khi đã có các nodes tương ứng là mô hình và công cụ, chúng ta xây dựng các hàm để thực hiện lời gọi đến mô hình ngôn ngữ hoặc các công cụ tương ứng. Gồm 3 hàm như sau:

- tool\_node: thực hiện lời gọi đến các công cụ và mô hình sẽ quyết định nên gọi đến công cụ nào cho một lần thực hiện hành động
- call\_model: thực hiện lời gọi đến mô hình chatGPT thông qua API
- should\_continue: định nghĩa hàm kiểm tra điều kiện để tiếp tục thực hiện hành động gọi tool hay là không để trả về kết quả.

```
1 import json
2 from langchain_core.messages import ToolMessage
3 from langchain_core.runnables import RunnableConfig
4
5 tools_by_name = {tool.name: tool for tool in tools}
6
7
8 # Define our tool node
9 def tool_node(state: AgentState):
10 outputs = []
11 for tool_call in state["messages"][-1].tool_calls:
12 tool_result = tools_by_name[tool_call["name"]].
13 invoke(tool_call["args"])
14 outputs.append(
15 ToolMessage(
16 content=json.dumps(tool_result),
17 name=tool_call["name"],
18 tool_call_id=tool_call["id"],
19)
20)
21 return {"messages": outputs}
22
23 # Define the node that calls the model
24 def call_model(
25 state: AgentState,
26 config: RunnableConfig,
27):
28 response = model.invoke([system_prompt] + state["messages"], config)
```

```

29 # We return a list, because this will get added to
30 # the existing list
31 return {"messages": [response]}
32
33 # Define the conditional edge that determines whether to
34 def should_continue(state: AgentState):
35 messages = state["messages"]
36 last_message = messages[-1]
37 # If there is no function call, then we finish
38 if not last_message.tool_calls:
39 return "end"
40 # Otherwise if there is, we continue
41 else:
42 return "continue"

```

#### 4. Xây dựng đồ thị

Cuối cùng để hoàn thiện, chúng ta xây dựng đồ thị cho ReAct Agent với thông tin như sau:

- Các nodes: agent (llms) và tool
- Vị trí bắt đầu khi người dùng gửi yêu cầu: agent
- Cạnh giữa agent đến tool: để kiểm tra nếu còn tool cần thực hiện sẽ tiếp tục nếu không thì sẽ kết thúc
- Cạnh giữa tool đến agent: sẽ trả về kết quả thực hiện hành động cho agent.

```

1 from langgraph.graph import StateGraph, END
2
3 # Define a new graph
4 workflow = StateGraph(AgentState)
5
6 # Define the two nodes we will cycle between
7 workflow.add_node("agent", call_model)
8 workflow.add_node("tools", tool_node)
9
10 # Set the entrypoint as agent. This means that this node
11 # is the first one called
11 workflow.set_entry_point("agent")
12
13 # We now add a conditional edge

```

```

14 workflow.add_conditional_edges(
15 # First, we define the start node. We use 'agent'.
16 # This means these are the edges taken after the 'agent' node is called.
17 "agent",
18 # Next, we pass in the function that will determine
19 # which node is called next.
20 should_continue,
21 {
22 "continue": "tools", # If 'tools', then we call
23 # the tool node.
24 "end": END, # Otherwise we finish.
25 },
26)
27 # We now add a normal edge from 'tools' to 'agent'.
28 # This means that after 'tools' is called, 'agent' node
29 # is called next.
30 workflow.add_edge("tools", "agent")
31 graph = workflow.compile()

```

Để kiểm tra đồ thị vừa xây dựng, chúng ta thực thi lệnh sau:

```

1 from IPython.display import Image, display
2
3 try:
4 display(Image(graph.get_graph().draw_mermaid_png()))
5 except Exception:
6 # This requires some extra dependencies and is
7 # optional
8 pass

```

Kết quả đồ thị như hình sau:

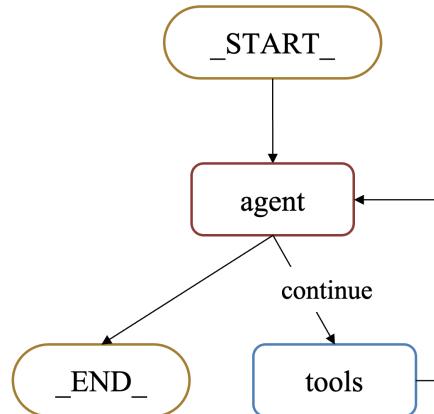
## 5. Kiểm tra

Sau đó, chúng ta thực hiện đoạn code sau và kiểm tra kết quả.

```

1 inputs = {"messages": [("user", "what is the weather in
2 vietnam? Then Triple it ")]}
3
4 for state in graph.stream(inputs, stream_mode="values"):
5 last_message = state["messages"][-1]
6 last_message.pretty_print()

```



Hình 62.5: ReAct Graph.

```

===== Human Message =====
what is the weather in vietnam? Then Triple it
===== Ai Message =====
Tool Calls:
 tavity_search (call_gN7sBfV4GynshBpXRt04V6x0)
 Call ID: call_gN7sBfV4GynshBpXRt04V6x0
 Args:
 query: current weather in Vietnam
 search_depth: basic
===== Tool Message =====
Name: tavity_search

{"query": "current weather in Vietnam", "follow_up_questions": null, "answer": null, "image": null}
===== Ai Message =====
Tool Calls:
 triple (call_NiT3Cptpz9kI2VBZ7putuI58)
 Call ID: call_NiT3Cptpz9kI2VBZ7putuI58
 Args:
 num: 28
===== Tool Message =====
Name: triple

84.0
===== Ai Message =====
...
Action Input: 28
Observation: The result of tripling is 84.0.

Final Answer: The current temperature in Vietnam is 28.2°C, and when tripled, it is 84.0°C.

```

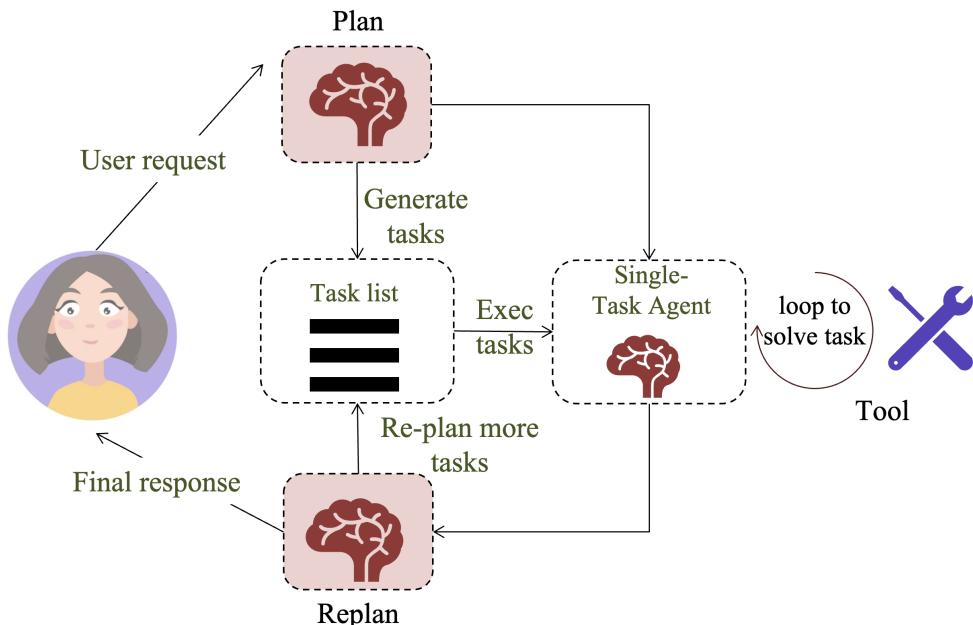
Hình 62.6: Test ReAct Agent.

### 62.3 Plan-and-Execute Agent

Plan-and-Execute (Lập kế hoạch và Thực thi) là một phương pháp trong đó agent đầu tiên phát triển một kế hoạch chi tiết cho toàn bộ nhiệm vụ, sau đó thực hiện kế hoạch đó theo từng bước.

Phương pháp Plan-and-Execute chia quá trình giải quyết vấn đề thành hai giai đoạn riêng biệt:

- Planning (Lập kế hoạch): Mô hình phát triển một kế hoạch tổng thể, chi tiết về các bước cần thực hiện để đạt được mục tiêu.
- Execution (Thực thi): Mô hình thực hiện từng bước một trong kế hoạch đã đề ra, theo đúng thứ tự đã định trước.



Hình 62.7: Plan-and-Execute Agent.

#### Giai đoạn lập kế hoạch:

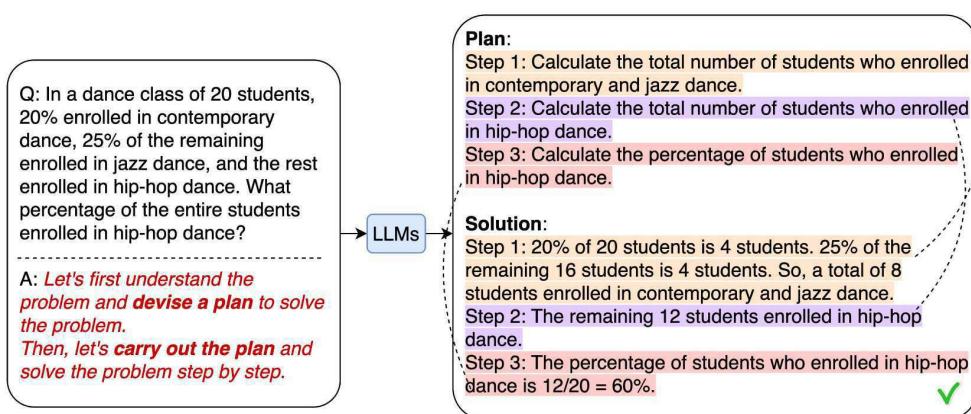
- Phân tích vấn đề: Mô hình phân tích kỹ lưỡng vấn đề, xác định mục tiêu và các điều kiện ràng buộc.

- Phân chia thành các bước nhỏ: Vấn đề được chia thành các bước nhỏ, có thể quản lý được.
- Sắp xếp thứ tự các bước: Các bước được sắp xếp theo một trình tự logic, xác định rõ điều kiện tiên quyết.
- Xác định tài nguyên cần thiết: Mô hình xác định các công cụ, thông tin hoặc tài nguyên cần thiết cho mỗi bước.

### Giai đoạn thực thi:

- Thực hiện từng bước: Mô hình thực hiện các bước theo kế hoạch đã đề ra.
- Thu thập kết quả: Sau mỗi bước, kết quả được ghi lại và đánh giá.
- Kiểm tra tiến độ: Tiến độ được theo dõi so với kế hoạch ban đầu.
- Hoàn thành mục tiêu: Quá trình tiếp tục cho đến khi tất cả các bước đã hoàn thành.

Ví dụ minh họa cho agent này như sau:



Hình 62.8: Plan-and-Execute Agent Example.

Trong phần này, chúng ta sẽ sử dụng thư viện LangGraph để xây dựng ReAct Agent

## 1. Cài đặt thư viện

Cài đặt một số thư viện và chuẩn bị các Key cho quá trình sử dụng LLMs cũng như lưu trữ thông tin thông qua công cụ LangSmith. Sau đó xây dựng lớp AgentState để thực thi gồm các thuộc tính như sau:

- input: Dữ liệu đầu vào
- plan: danh sách các bước cần thực hiện
- past\_steps: danh sách các bước đã thực hiện
- response: phản hồi

```

1 # Install libs
2 !pip install -U langchain==0.3.24 langchain-openai
 ==0.3.14 langgraph==0.3.33 langchain-tavily==0.1.6
3
4 import os
5 os.environ["LANGCHAIN_API_KEY"] = ### Your-key
6 os.environ["LANGCHAIN_PROJECT"] = "Reasoning-Agent"
7 os.environ["LANGCHAIN_TRACING_V2"] = "true"
8 os.environ["TAVILY_API_KEY"] = ### Your-key
9 os.environ["OPENAI_API_KEY"] = ### Your-key
10
11 import operator
12 from typing import Annotated, List, Tuple
13 from typing_extensions import TypedDict
14
15
16 class PlanExecute(TypedDict):
17 input: str
18 plan: List[str]
19 past_steps: Annotated[List[Tuple], operator.add]
20 response: str

```

## 2. Xây dựng các công cụ

Trong phần này, chúng ta xây dựng 2 công cụ để thực hiện hành động là tìm kiếm. Bên cạnh đó, chúng ta sử dụng mô hình chatGPT để thực hiện.

```

1 from langchain_tavily import TavilySearch
2
3 tools = [TavilySearch(max_results=1)]

```

### 3. Xây dựng mô hình lên kế hoạch

Mô hình lên kế hoạch là mô hình ngôn ngữ lớn, ở đây chúng ta sử dụng GPT và prompt để yêu cầu mô hình đề xuất các bước cần thực thi.

#### 4. Xây dựng mô hình thực thi

Mô hình thực thi cũng sẽ là mô hình GPT, trong đó sẽ có các thông tin đầy vào prompt là plan và past\_steps để đánh các bước thực thi và kết quả.

```
1 from typing import Union
2
3 class Response(BaseModel):
4 """Response to user."""
5
6 response: str
7
8 class Act(BaseModel):
9 """Action to perform."""
10
11 action: Union[Response, Plan] = Field(
12 description="Action to perform. If you want to
13 respond to user, use Response.
14 "If you need to further use tools to get the
15 answer, use Plan."
16)
17
18 replanner_prompt = ChatPromptTemplate.from_template(
19 """For the given objective, come up with a simple
20 step by step plan. \
21 This plan should involve individual tasks, that if
22 executed correctly will yield the correct answer. Do
23 not add any superfluous steps. \
24 The result of the final step should be the final answer.
25 Make sure that each step has all the information
26 needed - do not skip steps.
27
28 Your objective was this:
29 {input}
30
31 Your original plan was this:
32 {plan}
33
34 You have currently done the follow steps:
35 {past_steps}
36
37 Update your plan accordingly. If no more steps are needed
38 and you can return to the user, then respond with
39 that. Otherwise, fill out the plan. Only add steps to
40 the plan that still NEED to be done. Do not return
41 previously done steps as part of the plan."""
```

```

31)
32
33
34 replanner = replanner_prompt | ChatOpenAI(
35 model="gpt-4o", temperature=0
36).with_structured_output(Act)

```

## 5. Xây dựng chu trình thực hiện

Trong phần này chúng ta định nghĩa lời gọi hàm các công cụ để tạo sự liên kết giữa các thành phần. Bên cạnh đó định nghĩa hàm `should_end` để xem xét số lượng các bước thực hiện đã hết chưa. Nếu chưa tiếp tục bước tiếp theo nếu đã hết trả về kết quả.

```

1 from typing import Literal
2 from langgraph.graph import END
3
4 async def execute_step(state: PlanExecute):
5 plan = state["plan"]
6 plan_str = "\n".join(f"{i+1}. {step}" for i, step in
7 enumerate(plan))
8 task = plan[0]
9 task_formatted = f"""For the following plan:
10 {plan_str}\n\nYou are tasked with executing step
11 {1}, {task}."""
12 agent_response = await agent_executor.ainvoke(
13 {"messages": [("user", task_formatted)]})
14 return {
15 "past_steps": [(task, agent_response["messages"]
16 [-1].content)],
17 }
18
19 async def plan_step(state: PlanExecute):
20 plan = await planner.ainvoke({"messages": [("user",
21 state["input"])]})
22 return {"plan": plan.steps}
23
24 async def replan_step(state: PlanExecute):
25 output = await replanner.ainvoke(state)
26 if isinstance(output.action, Response):
27 return {"response": output.action.response}

```

```

28 return {"plan": output.action.steps}
29
30
31 def should_end(state: PlanExecute):
32 if "response" in state and state["response"]:
33 return END
34 else:
35 return "agent"

```

## 6. Xây dựng đồ thị

Xây dựng đồ thị với các thông tin như sau:

- Các nodes: agent, planner và replan
- Vị trí bắt đầu: planner
- Các cạnh tương ứng giữa planner và agent, agent và replan. Trong đó cạnh có điều kiện là từ replan đến agent để kết thúc chu trình thực hiện.

```

1 from langgraph.graph import StateGraph, START
2
3 workflow = StateGraph(PlanExecute)
4
5 # Add the plan node
6 workflow.add_node("planner", plan_step)
7
8 # Add the execution step
9 workflow.add_node("agent", execute_step)
10
11 # Add a replan node
12 workflow.add_node("replan", replan_step)
13
14 workflow.add_edge(START, "planner")
15
16 # From plan we go to agent
17 workflow.add_edge("planner", "agent")
18
19 # From agent, we replan
20 workflow.add_edge("agent", "replan")
21
22 workflow.add_conditional_edges(
23 "replan",
24 # Next, we pass in the function that will determine
 which node is called next.

```

```

25 should_end,
26 ["agent", END],
27)
28
29 # Finally, we compile it!
30 # This compiles it into a LangChain Runnable,
31 # meaning you can use it as you would any other runnable
32 app = workflow.compile()

```

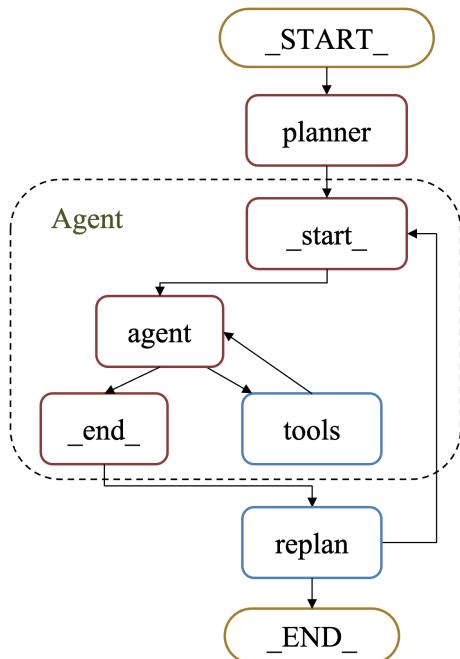
Sau khi xây dựng đồ thị, để kiểm tra và hiển thị đồ thị chúng ta chạy lệnh sau:

```

1 from IPython.display import Image, display
2
3 display(Image(app.get_graph(xray=True).draw_mermaid_png())
4)

```

Kết quả thu được:



Hình 62.9: Plan-and-Execute Graph.

## 7. Thử nghiệm mô hình

Để kiểm tra khả năng suy luận của agent, chúng ta chạy đoạn code sau:

```

1 config = {"recursion_limit": 50}
2
3 inputs = {"input": "Solve for all real solutions x to the
4 equation: sqrt(x - 2) = 3"}
5
6 async for event in app.astream(inputs, config=config):
7 for k, v in event.items():
8 if k != "__end__":
9 print(v)

```

Kết quả như hình sau:

- ✓ Input  
Solve for all real solutions x to the equation:  $\sqrt{x - 2} = 3$
- ✓ Response  
All steps have been completed successfully, and the solution  $(x = 11)$  has been verified as correct. No further steps are needed. The final answer is:
- \*\*x = 11\*\*
- ✓ Past Steps
  - ✓ 0
    - > 0 Start with the equation:  $\sqrt{x - 2} = 3$ .
    - > 1 Step 1 has already been specified in the plan. The equation give...
  - ✓ 1
    - > 0 Square both sides of the equation to eliminate the square root: ...
    - > 1 To execute step 1, we start with the equation:  $\sqrt{x - 2} = 3$  ...
  - ✓ 2
    - > 0 Add 2 to both sides to solve for x:  $x - 2 + 2 = 9 + 2$ .
    - > 1 In step 1, we start with the equation  $\sqrt{x - 2} = 3$ . To solve...
  - ✓ 3
    - > 0 Verify the solution by substituting  $x = 11$  back into the original equation.
    - > 1 To verify the solution by substituting  $x = 11$  back into the original equation.
- ✓ Plan
  - > 0 Verify the solution by substituting  $x = 11$  back into the original equation.
  - > 1 Simplify the left side:  $\sqrt{9} = 3$ .
  - > 2 Since both sides are equal,  $x = 11$  is a valid solution.

Hình 62.10: Plan-and-Execute Result.

## 62.4 Câu hỏi trắc nghiệm

**Câu hỏi 143** Reasoning Agents được định nghĩa như thế nào?

- a) Hệ thống AI chỉ thực hiện các hành động theo lệnh
- b) Hệ thống AI với khả năng suy luận có cấu trúc để giải quyết vấn đề phức tạp
- c) Hệ thống chỉ gọi đến các công cụ bên ngoài khi cần thiết
- d) Hệ thống đơn thuần phân tích dữ liệu lớn mà không lập kế hoạch

**Câu hỏi 144** Phương pháp ReAct trong Reasoning Agents là viết tắt của:

- a) Reactive Action
- b) Reasoning Activity
- c) Reasoning + Acting
- d) Refined Action

**Câu hỏi 145** Trình tự đúng của các bước trong phương pháp ReAct là gì?

- a) Hành động → Suy luận → Quan sát → Tiếp tục chu trình
- b) Suy luận → Hành động → Quan sát → Tiếp tục chu trình
- c) Quan sát → Suy luận → Hành động → Tiếp tục chu trình
- d) Suy luận → Quan sát → Hành động → Tiếp tục chu trình

**Câu hỏi 146** Sau khi ReAct-agent gửi một Action để gọi công cụ bên ngoài (ví dụ: Wikipedia-API), bước kế tiếp diễn hình là gì?

- a) Kết thúc phiên làm việc
- b) Thêm kết quả công cụ vào bối cảnh rồi tiếp tục Reasoning
- c) Huấn luyện lại trọng số mô hình
- d) Bỏ qua kết quả công cụ và suy luận tiếp

**Câu hỏi 147** Mục đích chính của việc xen kẽ Reasoning và Acting trong ReAct là gì?

- a) Tăng kích thước prompt để khai thác tối đa context của LLM
- b) Tách logic và dữ liệu để thuận tiện deploy micro-service
- c) Hạn chế “hallucination” bằng cách lấy dữ liệu thực qua mỗi bước hành động
- d) Giảm lượng token sinh ra nhằm tiết kiệm chi phí

**Câu hỏi 148** Đâu là ưu điểm chính của phương pháp Plan-and-Execute so với ReAct?

- a) Khả năng thích ứng cao hơn với các tình huống không lường trước
- b) Xử lý vấn đề nhanh hơn trong mọi trường hợp
- c) Có tầm nhìn tổng thể về toàn bộ nhiệm vụ từ đầu
- d) Tiêu tốn ít tài nguyên xử lý hơn

**Câu hỏi 149** Đâu là khác biệt quan trọng nhất về cách tiếp cận giữa ReAct và Plan-and-Execute?

- a) ReAct lặp đi lặp lại và linh hoạt, Plan-and-Execute tuân tự và có cấu trúc
- b) ReAct dùng cho nhiệm vụ đơn giản, Plan-and-Execute cho nhiệm vụ phức tạp
- c) ReAct chỉ phân tích, Plan-and-Execute chỉ hành động
- d) ReAct tốn nhiều tài nguyên hơn Plan-and-Execute

**Câu hỏi 150** Công cụ nào sau đây được sử dụng để tìm kiếm thông tin?

- a) Tavily
- b) Wikipedia
- c) LangGraph
- d) LangSmith

**Câu hỏi 151** Trong LangGraph, "edges"(cạnh) có chức năng gì?

- a) Kết nối node với dữ liệu đầu vào
- b) Tạo điều kiện để chuyển trạng thái dựa vào kết quả node trước đó
- c) Lưu trữ biến tạm
- d) Tăng tốc độ xử lý mô hình LLM

**Câu hỏi 152** LangGraph hỗ trợ loại agent nào sau đây?

- a) ReAct
- b) Plan-and-Execute
- c) Tool-augmented reasoning agents
- d) Tất cả các loại trên

## 62.5 Phụ lục

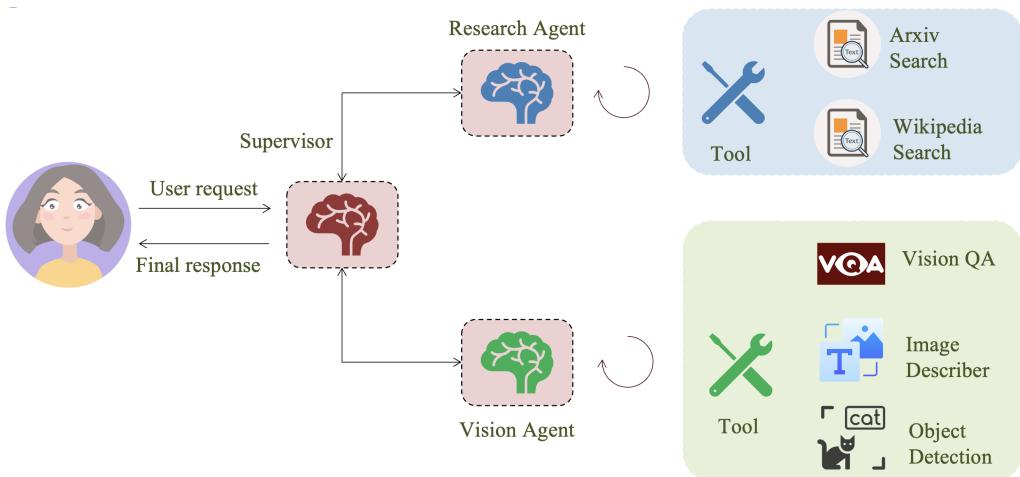
1. **Code:** Thư mục chứa mã nguồn notebook [CODE](#)
2. **Code:** Tài liệu code trên folder [CODE](#)
3. Tài liệu tham khảo:
  - [ReAct: Synergizing Reasoning and Acting in Language Models](#)
  - [Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models](#)
  - [LangGraph](#)
4. **Rubric:**

- *Hết* -

# Chương 63

## Project 2: Agentic AI Using Vision Language Model

### 63.1 Giới thiệu



Hình 63.1: Visual Agentic Agent.

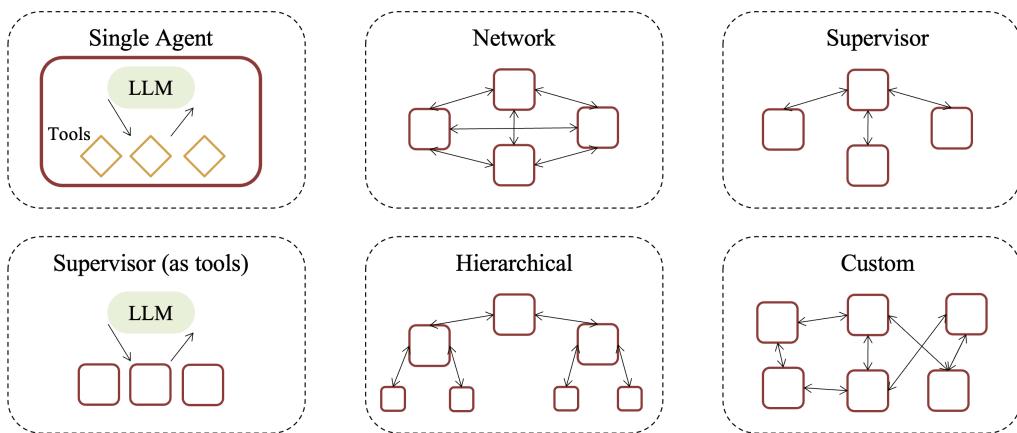
Với sự phát triển mạnh mẽ của các mô hình ngôn ngữ lớn (LLM), việc triển khai các hệ thống có khả năng tư duy, ra quyết định và phối hợp hành động đang trở thành xu hướng trong xây dựng ứng dụng AI thông minh. Tuy nhiên, khi chỉ dựa vào một tác nhân đơn lẻ (single-agent), ta dễ gặp phải các hạn chế như:

- Quá tải công cụ: mô hình phải chọn đúng công cụ từ hàng chục lựa chọn.
- Quá tải ngữ cảnh: khó quản lý các bước trung gian trong các tác vụ phức tạp.

- Thiếu chuyên môn hóa: một agent không thể đồng thời làm tốt các tác vụ lập kế hoạch, tìm kiếm, xử lý số liệu, lập trình, v.v.

Để giải quyết các thách thức này, kiến trúc multi-agent được đề xuất như một phương án khả thi: chia nhỏ hệ thống thành các agent độc lập, mỗi agent phụ trách một nhiệm vụ cụ thể. Các agent này có thể giao tiếp với nhau, hoặc được điều phối bởi một thực thể giám sát (supervisor).

Hình minh họa dưới đây thể hiện sáu dạng kiến trúc agent phổ biến, từ đơn giản đến phức tạp:



Hình 63.2: Multi-agent Architectures.

- Network:** Các agent kết nối với nhau theo mạng không định hướng (peer-to-peer). Mỗi agent có thể gửi thông tin cho các agent khác theo logic riêng. Phù hợp với các hệ thống phức tạp, cần khả năng phối hợp linh hoạt. Ví dụ ứng dụng: hệ thống tranh luận nhiều chiều, nhóm cộng tác AI.
- Supervisor:** Một agent đóng vai trò giám sát, điều phối hoạt động của các agent con. Supervisor nhận đầu vào từ người dùng và chia task cho các agent chuyên biệt. Phản hồi cuối cùng được tổng hợp từ các kết quả agent con gửi lên. Ưu điểm: đơn giản hóa điều phối, dễ theo dõi luồng tác vụ.
- Supervisor (as tools):** Biến các agent thành "tools" có thể được gọi bởi một LLM chính. LLM chọn tool nào để gọi tùy theo nội dung yêu cầu.

Tương tự cơ chế function calling, nhưng mỗi "tool" là một agent có logic riêng. Lợi ích: dễ tích hợp vào các pipeline hiện có, tận dụng được mô hình LLM như một router động.

- **Hierarchical** Các agent được sắp xếp theo dạng cây (tree), với luồng điều phối từ trên xuống. Mỗi agent cha chia task cho agent con, và có thể nhận kết quả để đưa ra hành động tiếp theo. Phù hợp với các bài toán cần phân chia tác vụ theo nhiều lớp, ví dụ: chiến lược → chiến thuật → hành động cụ thể. Đặc trưng: rõ ràng, có cấu trúc – dễ kiểm soát nhưng thiếu linh hoạt nếu task thay đổi liên tục.
- **Custom** Các hệ thống được thiết kế linh hoạt theo logic riêng, không tuân theo hình mẫu nào cố định. Có thể kết hợp cả supervisor, peer-to-peer, hoặc các cơ chế lập lịch riêng. Phù hợp với hệ thống sản phẩm lớn, cần kiểm soát hiệu năng, chi phí, độ tin cậy. Ví dụ: hệ thống trợ lý doanh nghiệp AI gồm nhiều thành phần: trợ lý email, lập lịch, tìm kiếm, xử lý báo cáo...

## 63.2 Visual Agentic AI

Trong phần này, chúng ta sẽ sử dụng thư viện LangGraph để xây dựng agent dựa trên kiến trúc **Supervisor** thông qua thư viện LangGraph.

LangGraph là một framework xây dựng trên LangChain, cho phép mô hình hóa các quy trình tác vụ của LLM dưới dạng đồ thị trạng thái có vòng lặp (stateful graph). Một trong những cấu trúc mạnh mẽ mà LangGraph hỗ trợ là multi-agent supervisor architecture – nơi một agent đóng vai trò "supervisor" (giám sát), điều phối các agent con có chuyên môn khác nhau để hoàn thành tác vụ.

Kiến trúc này phản ánh mô hình phân quyền trong các tổ chức thực tế: supervisor giống như người quản lý, còn các agents con là nhân viên chuyên môn (ví dụ: tìm kiếm thông tin, phân tích số liệu, lập kế hoạch).

Trong đó:

- Supervisor: Là agent trung tâm có mục đích điều phối hoạt động thực hiện các tác vụ và tương tác với các agent khác.
- Hai agents thực hiện hành động chính: Research agent và Vision agent. Các tác nhân này tương tác trực tiếp với Supervisor Agent.
- Các agent thiết kế dựa vào ReAct Agent.

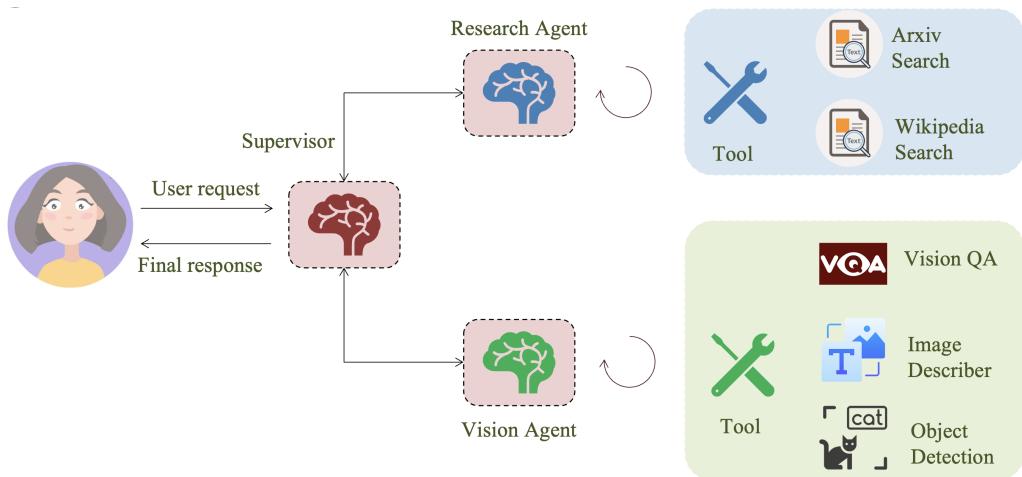
Kiến trúc của agent như sau:

ReAct Agent hoạt động dựa trên việc kết hợp:

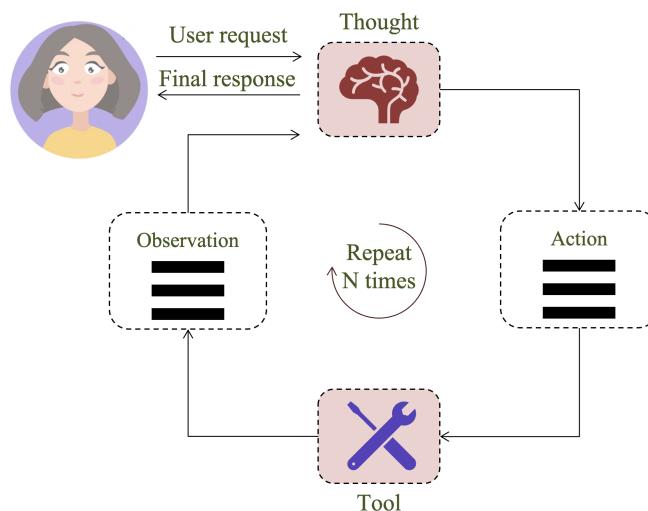
- Reasoning (Suy luận): Mô hình suy nghĩ về vấn đề, phân tích tình huống và lập kế hoạch cho hành động tiếp theo.
- Acting (Hành động): Mô hình thực hiện các hành động dựa trên suy luận của mình, thu thập thông tin mới và tiếp tục quá trình.

### 1. Cài đặt thư viện

Cài đặt một số thư viện, bao gồm LangGraph, các thư viện hỗ trợ công cụ tìm kiếm như Tavily, Arxiv, Wikipedia, và chuẩn bị các API key cho quá trình sử dụng LLMs.



Hình 63.3: Visual Agentic Agent.



Hình 63.4: ReAct Agent sử dụng thư viện LangGraph.

```

1 # Install libs
2 !pip install -U -qq langchain==0.3.24 langchain-openai
 ==0.3.14 langgraph==0.3.33 langchain-tavily==0.1.6
 langgraph-supervisor
3 !pip install --upgrade -qq langchain-core langchain-
 community arxiv duckduckgo-search wikipedia python-
 magic ultralytics

```

```
4
5 import os
6 os.environ["LANGCHAIN_API_KEY"] = ... ### Your-key
7 os.environ["LANGCHAIN_PROJECT"] = "Visual-Agent-AI"
8 os.environ["LANGCHAIN_TRACING_V2"] = "true"
9 os.environ["TAVILY_API_KEY"] = ... ### Your-key
10 os.environ["OPENAI_API_KEY"] = ... ### Your-key
11
12 import base64
13 import os
14 from typing import Annotated, List, Optional, Union
15
16 import magic
17 import requests
18 from IPython.display import Image, display
19 from langchain.chat_models import init_chat_model
20 from langchain.output_parsers import PydanticOutputParser
21 from langchain.prompts import PromptTemplate
22 from langchain_community.tools import (
23 ArxivQueryRun,
24 DuckDuckGoSearchResults,
25 WikipediaQueryRun,
26)
27 from langchain_community.utilities import (
28 ArxivAPIWrapper,
29 DuckDuckGoSearchAPIWrapper,
30 WikipediaAPIWrapper,
31)
32 from langchain_core.callbacks import (
33 AsyncCallbackManagerForToolRun,
34 CallbackManagerForToolRun,
35)
36 from langchain_core.messages import (
37 AnyMessage,
38 HumanMessage,
39 SystemMessage,
40 convert_to_messages,
41)
42 from langchain_core.prompts import ChatPromptTemplate
43 from langchain_core.tools import BaseTool, tool
44 from langchain_core.tools.base import ArgsSchema
45 from langchain_openai import ChatOpenAI
46 from langgraph.graph import END, START, StateGraph
47 from langgraph.graph.message import add_messages
48 from langgraph.prebuilt import ToolNode,
```

```

 create_react_agent, tools_condition
49 from langgraph_supervisor import create_supervisor
50 from pydantic import BaseModel, Field
51 from typing_extensions import TypedDict
52 from ultralytics import YOLO

```

## 2. Khởi tạo LLM

Trong phần này, chúng ta sử dụng mô hình gpt-4.1-nano để làm “bộ não” cho các agents của chúng ta.

```

1 llm = ChatOpenAI(model="gpt-4.1-nano")

```

Để chạy thử mô hình LLM đã load, chúng ta có thể dùng method `invoke()` như sau:

```

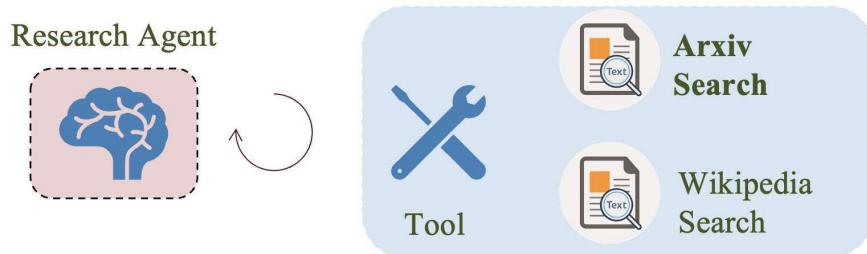
1 llm.invoke("How are you?")
2
3 # Output result
4 # AIMessage(content="I'm doing well, thank you! How can I
 assist you today?", additional_kwargs={"refusal":
 None}, response_metadata={"token_usage": {
 "completion_tokens": 15, "prompt_tokens": 11, "total_tokens": 26, "completion_tokens_details": {"accepted_prediction_tokens": 0, "audio_tokens": 0, "reasoning_tokens": 0, "rejected_prediction_tokens": 0}, "prompt_tokens_details": {"audio_tokens": 0, "cached_tokens": 0}}, "model_name": "gpt-4.1-nano-2025-04-14", "system_fingerprint": "fp_8fd43718b3", "id": "chatcmpl-BWDKARIICI3n9a2wJGyhWhtY4zUn1N", "finish_reason": "stop", "logprobs": None}, id="run--005f9422-82d7-41ae-9e2a-d26d961d2a1a-0",
 usage_metadata={"input_tokens": 11, "output_tokens": 15, "total_tokens": 26, "input_token_details": {"audio": 0, "cache_read": 0}, "output_token_details": {"audio": 0, "reasoning": 0}})

```

## 3. Xây dựng Research Agent

Trong phần này, chúng ta xây dựng research agent, một công cụ có thể hỗ trợ chúng ta trong việc tìm kiếm, đọc, tóm các công trình nghiên cứu khoa học, hay giải thích một khái niệm học thuật nào đó.

Đầu tiên, chúng ta định nghĩa các tools (công cụ) mà research agent của chúng ta có thể sử dụng được cho quá trình tìm kiếm các công



Hình 63.5: Research Agent.

trình nghiên cứu. Đó là hai công cụ Arxiv (để tìm kiếm các bài báo) và Wikipedia (để tìm các định nghĩa, khái niệm mới).

Để load công cụ tìm kiếm bài báo trên Arxiv, ta tạo đối tượng `ArxivAPIWrapper` với các tham số `top_k_results=2` (chỉ lấy 2 kết quả hàng đầu) và `doc_content_chars_max=1000` (giới hạn tối đa 1000 ký tự cho mỗi tài liệu trả về). Tiếp theo, sử dụng đối tượng này để khởi tạo `ArxivQueryRun` với `description` là tìm kiếm các bài báo trên Arxiv theo chủ đề. Description này rất quan trọng để mô hình LLM có thể hiểu được tool Arxiv này được dùng cho mục đích gì và có thể chọn đúng tool để gọi trong quá trình sử dụng.

```

1 arxiv_wrapper = ArxivAPIWrapper(
2 top_k_results=2, doc_content_chars_max=1000
3)
4 arxiv = ArxivQueryRun(
5 api_wrapper=arxiv_wrapper,
6 description="Search for papers on a given topic using
7 Arxiv"
8)
9 arxiv.invoke("Rotary Positional Encoding")
9 ### Output: Published: 2025-03-03\nTitle: Rotary Outliers\
 and Rotary Offset Features in Large Language Models\
 nAuthors: Andre Jonasson\nSummary: Transformer-based
 Large Language Models (LLMs) rely on positional
 encodings\nthat provide sequence position information to
 their attention mechanism. Rotary\nPositional
 Encodings (RoPE), which encode relative position by
 rotating queries\nand keys, have become widely used in
 modern LLMs. We study the features and\npatterns that
 emerge in queries and keys when using rotary
 embeddings. Our\nanalysis reveals consistent patterns
 within the same model across layers and\nattention

```

```

heads and across different models and architectures.
We present and\apply analysis techniques and show how
the queries and keys use RoPE to\construct various
attention patterns, including attention sinks. We find
and\analyze outliers across models in queries and
keys and find that they are\unlikely to be found in
rotary features with partial cycles. We derive bounds\
that tell us what

```

Tương tự, ta khởi tạo công cụ Wikipedia:

```

1 wikipedia_wrapper = WikipediaAPIWrapper()
2 wikipedia = WikipediaQueryRun(
3 api_wrapper=wikipedia_wrapper,
4 description="Search for information on a given topic
5 using Wikipedia"
6)
7 wikipedia.invoke("machine learning")
7 ### Output: Page: Machine learning\nSummary: Machine
learning (ML) is a field of study in artificial
intelligence concerned with the development and study
of statistical algorithms that can learn from data and
generalise to unseen data, and thus perform tasks
without explicit instructions. Within a subdiscipline
in machine learning, advances in the field of deep
learning have allowed neural networks, a class of
statistical algorithms, to surpass many previous
machine learning approaches in performance.\nML finds
application in many fields, including natural language
processing, computer vision, speech recognition,
email filtering, agriculture, and medicine. The
application of ML to business problems is known as
predictive analytics.\nStatistics and mathematical
optimisation (mathematical programming) methods
comprise the foundations of machine learning. Data
mining is a related field of study, focusing on
exploratory data analysis (EDA) via unsupervised
learning. \nFrom a theoretical viewpoint, pr

```

#### 4. Xây dựng các hàm in thông tin ra màn hình

Để xem cách agent hoạt động trong quá trình chạy, chúng ta thiết kế các hàm sau:

- Hàm `pretty_print_message` nhận vào một danh sách tin nhắn (`message`) và flag `indent` xác định có thực đầu dòng khi in hay

không. Tin nhắn này sẽ được chuyển sang dạng biểu diễn đẹp hơn (`pretty_repr`) hỗ trợ hiển thị dưới dạng HTML. Nếu `indent=False`, hàm sẽ in trực tiếp nội dung tin nhắn; ngược lại, mỗi dòng trong nội dung sẽ được thêm ký tự tab để thụt đầu dòng trước khi hiển thị.

- Hàm thứ hai, `pretty_print_messages`, được sử dụng để hiển thị các tin nhắn trong một cập nhật (`update`), có thể đến từ một subgraph hoặc từ một node riêng biệt. Nếu `update` là một tuple, hàm sẽ trích xuất namespace (`ns`) và nội dung cập nhật, bỏ qua các cập nhật từ graph cha (namespace rỗng) và hiển thị tên subgraph nếu có. Với mỗi nút trong cập nhật, hàm sẽ in ra tiêu đề kèm theo tên nút, sau đó chuyển các tin nhắn từ dạng nguyên thủy sang đối tượng message bằng hàm `convert_to_messages`. Nếu tham số `last_message=True`, chỉ tin nhắn cuối cùng được hiển thị. Cuối cùng, mỗi tin nhắn được hiển thị bằng cách gọi lại hàm `pretty_print_message` với tùy chọn thụt đầu dòng tương ứng với subgraph hay không.

```

1 from langchain_core.messages import convert_to_messages
2
3 def pretty_print_message(message, indent=False):
4 pretty_message = message.pretty_repr(html=True)
5 if not indent:
6 print(pretty_message)
7 return
8
9 indented = "\n".join("\t" + c for c in pretty_message
10 .split("\n"))
11 print(indented)
12
13 def pretty_print_messages(update, last_message=False):
14 is_subgraph = False
15 if isinstance(update, tuple):
16 ns, update = update
17 # skip parent graph updates in the printouts
18 if len(ns) == 0:
19 return
20
21 graph_id = ns[-1].split(":")[0]
22 print(f"Update from subgraph {graph_id}:")
23 print("\n")
24 is_subgraph = True

```

```

24
25 for node_name, node_update in update.items():
26 update_label = f"Update from node {node_name}:"
27 if is_subgraph:
28 update_label = "\t" + update_label
29
30 print(update_label)
31 print("\n")
32
33 messages = convert_to_messages(node_update[""
34 "messages"])
35 if last_message:
36 messages = messages[-1:]
37
38 for m in messages:
39 pretty_print_message(m, indent=is_subgraph)
print("\n")

```

## 5. Xây dựng Research Agent dựa trên ReAct Agent Pattern

ReAct agent là một AI agent kết hợp khả năng suy luận (reasoning) và hành động (acting) của các mô hình ngôn ngữ lớn (LLMs) và các công cụ (tools) trong một vòng lặp có cấu trúc. Agent tương tác với môi trường xung quanh thông qua các công cụ và đưa ra quyết định tiếp theo dựa trên khả năng suy luận của chính bản thân.

Ta xây dựng research agent dựa theo ReAct agent pattern để mô hình có thể tự lựa chọn các công cụ để tìm kiếm thông tin và trả về kết quả liên quan đến hướng nghiên cứu chúng ta mong muốn.

Chúng ta sử dụng hàm `create_react_agent` từ thư viện LangChain để tạo một ReAct agent có tên là `research_agent`. Agent này có “bộ não” là mô hình `gpt-4o-mini` và hai công cụ là `arxiv` và `wikipedia` để tra cứu các thông tin cần thiết.

```

1 research_agent = create_react_agent(
2 model="gpt-4o-mini",
3 tools=[arxiv, wikipedia],
4 prompt=(
5 "You are a research agent.\n\n"
6 "INSTRUCTIONS:\n"
7 "- Assist ONLY with research-related tasks, DO
NOT do any math\n"
8 "- After you're done with your tasks, respond to
the supervisor directly\n"

```

```

9 "- Respond ONLY with the results of your work, do
10 NOT include ANY other text."
11),
12 name="research_agent",
13)

```

Ta chạy research agent với query "machine learning" như sau:

```

1 for chunk in research_agent.stream(
2 {"messages": [{"role": "user", "content": "machine
3 learning"}]}
4):
4 pretty_print_messages(chunk)

```

và kết quả là:

```

1 Update from node agent:
2
3
4 ====== Ai Message
4 ======
5 Name: research_agent
6 Tool Calls:
7 arxiv (call_ElbwV5iP84M8L63N9CGwYxMX)
8 Call ID: call_ElbwV5iP84M8L63N9CGwYxMX
9 Args:
10 query: machine learning
11 wikipedia (call_9C5iKyuNCtGuxOYsGnHCQg2j)
12 Call ID: call_9C5iKyuNCtGuxOYsGnHCQg2j
13 Args:
14 query: machine learning
15
16
17 Update from node tools:
18
19
20 ====== Tool Message
20 ======
21 Name: arxiv
22
23 Published: 2019-09-08
24 Title: Lecture Notes: Optimization for Machine Learning
25 Authors: Elad Hazan
26 Summary: Lecture notes on optimization for machine
26 learning, derived from a course at
27 Princeton University and tutorials given in MLSS, Buenos
27 Aires, as well as

```

28 Simons Foundation, Berkeley.  
29  
30 Published: 2018-11-11  
31 Title: An Optimal Control View of Adversarial Machine  
Learning  
32 Authors: Xiaojin Zhu  
33 Summary: I describe an optimal control view of  
adversarial machine learning, where the  
34 dynamical system is the machine learner, the input are  
adversarial actions, and  
35 the control costs are defined by the adversary's goals to  
do harm and be hard  
36 to detect. This view encompasses many types of  
adversarial machine learning,  
37 including test-item attacks, training-data poisoning, and  
adversarial reward  
38 shaping. The view encourages adversarial machine learning  
researcher to utilize  
39 advances in control theory and reinforcement learning.  
40 ===== Tool Message  
=====

41 Name: wikipedia  
42  
43 Page: Machine learning  
44 Summary: Machine learning (ML) is a field of study in  
artificial intelligence concerned with the development  
and study of statistical algorithms that can learn  
from data and generalise to unseen data, and thus  
perform tasks without explicit instructions. Within a  
subdiscipline in machine learning, advances in the  
field of deep learning have allowed neural networks, a  
class of statistical algorithms, to surpass many  
previous machine learning approaches in performance.  
45 ML finds application in many fields, including natural  
language processing, computer vision, speech  
recognition, email filtering, agriculture, and  
medicine. The application of ML to business problems  
is known as predictive analytics.  
46 Statistics and mathematical optimisation (mathematical  
programming) methods comprise the foundations of  
machine learning. Data mining is a related field of  
study, focusing on exploratory data analysis (EDA) via  
unsupervised learning.  
47 From a theoretical viewpoint, probably approximately  
correct learning provides a framework for describing

machine learning.

48

49 Page: Neural network (machine learning)

50 Summary: In machine learning, a neural network (also artificial neural network or neural net, abbreviated ANN or NN) is a computational model inspired by the structure and functions of biological neural networks.

51 A neural network consists of connected units or nodes called artificial neurons, which loosely model the neurons in the brain. Artificial neuron models that mimic biological neurons more closely have also been recently investigated and shown to significantly improve performance. These are connected by edges, which model the synapses in the brain. Each artificial neuron receives signals from connected neurons, then processes them and sends a signal to other connected neurons. The "signal" is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs, called the activation function. The strength of the signal at each connection is determined by a weight, which adjusts during the learning process.

52 Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer) to the last layer (the output layer), possibly passing through multiple intermediate layers (hidden layers). A network is typically called a deep neural network if it has at least two hidden layers.

53 Artificial neural networks are used for various tasks, including predictive modeling, adaptive control, and solving problems in artificial intelligence. They can learn from experience, and can derive conclusions from a complex and seemingly unrelated set of information.

54

55 Page: Attention (machine learning)

56 Summary: Attention is a machine learning method that determines the importance of each component in a sequence relative to the other components in that sequence. In natural language processing, importance is represented by "soft" weights assigned to each word in a sentence. More generally, attention encodes vectors called token embeddings across a fixed-width sequence that can range from tens to millions of tokens in size.

```
57 Unlike "hard" weights, which are computed during the
 backwards training pass, "soft" weights exist only in
 the forward pass and therefore change with every step
 of the input. Earlier designs implemented the
 attention mechanism in a serial recurrent neural
 network (RNN) language translation system, but a more
 recent design, namely the transformer, removed the
 slower sequential RNN and relied more heavily on the
 faster parallel attention scheme.
58 Inspired by ideas about attention in humans, the
 attention mechanism was developed to address the
 weaknesses of leveraging information from the hidden
 layers of recurrent neural networks. Recurrent neural
 networks favor more recent information contained in
 words at the end of a sentence, while information
 earlier in the sentence tends to be attenu
59
60
61 Update from node agent:
62
63
64 ====== Ai Message
65 ======
65 Name: research_agent
66
67 **Arxiv Papers on Machine Learning:**
68
69 1. **Lecture Notes: Optimization for Machine Learning**
70 - **Published:** 2019-09-08
71 - **Authors:** Elad Hazan
72 - **Summary:** Lecture notes on optimization for
 machine learning, derived from a course at Princeton
 University and tutorials given in MLSS, Buenos Aires,
 as well as Simons Foundation, Berkeley.
73
74 2. **An Optimal Control View of Adversarial Machine
 Learning**
75 - **Published:** 2018-11-11
76 - **Authors:** Xiaojin Zhu
77 - **Summary:** This paper describes an optimal control
 view of adversarial machine learning, where the
 dynamical system is the machine learner and
 adversarial actions serve as inputs.
78
79 **Wikipedia Summary on Machine Learning:**
```

```

80
81 - **Machine Learning (ML)** is a field of study in
 artificial intelligence focusing on the development
 and study of statistical algorithms that learn from
 data and make generalizations to unseen data. It
 performs tasks without explicit instructions and finds
 applications in various fields such as natural
 language processing, computer vision, and medicine.
82
83 - **Key Concepts:**
84 - Foundations in statistics and mathematical
 optimization.
85 - Distinction from data mining, which focuses on
 exploratory data analysis.
86 - Theoretical frameworks like probably approximately
 correct learning.
87
88 - **Neural Networks:** A class of algorithms in ML
 modeled on biological neural networks. They comprise
 layers of artificial neurons that process signals and
 learn from data.
89
90 - **Attention Mechanism:** A method in ML that determines
 the importance of components in sequences and
 enhances the processing of data in tasks like language
 translation.
91
92 This summarizes the latest findings in the field of
 machine learning from both arxiv and Wikipedia sources
 .

```

## 6. Xây dựng Visual Agent



Hình 63.6: Visual Agent.

Chúng ta tiếp tục xây dựng một visual agent có khả năng xử lý các thông tin liên quan đến hình ảnh, ví dụ như nhận diện vật thể trong hình, đếm vật thể, v.v...

Đối với OpenAI API, ta phải mã hoá hình ảnh dưới dạng base64 để gửi trong message đến server. Do đó, ta thiết kế hàm `encode_image` dùng để đọc hình ảnh từ file path hoặc từ URL và trả về dưới dạng base64 như sau:

```
1 def encode_image(image_path_or_url: str, get_mime_type:
2 bool = False):
3 if image_path_or_url.startswith("http"):
4 try:
5 response = requests.get(image_path_or_url,
6 stream=True)
6 response.raise_for_status()
7 image = response.content
8 mime_type = response.headers.get("content-
9 type", None)
10 base64_encoded = base64.b64encode(image).
11 decode("utf-8")
12 if get_mime_type:
13 return base64_encoded, mime_type
14 else:
15 return base64_encoded
16 except requests.exceptions.RequestException as e:
17 print(f"Request error: {e}")
18 if get_mime_type:
19 return None, None
20 else:
21 return None
22 else:
23 if not os.path.exists(image_path_or_url):
24 return None, None
25 mime_type = magic.Magic(mime=True).from_file(
26 image_path_or_url)
27 if mime_type.startswith("image/"):
28 with open(image_path_or_url, "rb") as
29 image_file:
30 if get_mime_type:
31 return base64.b64encode(image_file.
32 read()).decode("utf-8"), mime_type
33 else:
34 return base64.b64encode(image_file.
35 read()).decode("utf-8")
36 else:
37 if get_mime_type:
38 return None, None
39 else:
```

33

```
 return None
```

Tiếp theo, ta thiết kế một tool URL extractor để trích xuất image URL từ message của người dùng. Ta ví dụ rằng nếu người dùng đưa một message như

`What is the object inside this image: https://www.abc.com/image.png` thì tool có khả năng trích xuất image URL trong message là `https://www.abc.com/image.png` để có thể xử lý ảnh dưới dạng base64 trước khi gửi đến server OpenAI.

URL extractor tool được thiết kế như sau:

```
1 class ImageInput(BaseModel):
2 image_path_or_url: str = Field(description="Image
3 path or URL")
4
5
6 prompt = PromptTemplate.from_template(
7 "Extract the image path or URL from the following
8 input:\n\n{input}\n\n{format_instructions}"
9).partial(format_instructions=parser.
10 get_format_instructions())
11
12 extractor_chain = prompt | llm | parser
13 output = extractor_chain.invoke({"input": "Please
14 describe the following image in detailed: \"https://
15 example.com/image.png\""})
16
17
18 # Output: ImageInput(image_path_or_url="https://example.
19 com/image.png")
```

Bên cạnh việc mô hình LLM của chúng ta có thể xử lý hình ảnh, ta cũng có thể trang bị thêm các công cụ hỗ trợ để mô hình có thêm nhiều thông tin từ hình ảnh hơn, từ đó đưa ra câu trả lời chính xác.

Đầu tiên, ta xây dựng tool miêu tả hình ảnh `image_describer_tool` như sau:

```
1 class ImageDescription(BaseModel):
2 image_description: str = Field(description="Detailed
3 description of the image")
4
5 def image_describer_prompt_func(inputs: dict):
6 image_path_or_url = inputs["image_path_or_url"]
```

```
6 image_b64, image_mime_type = encode_image(
7 image_path_or_url, get_mime_type=True)
8
9 image_describer_chat_template = ChatPromptTemplate.
10 from_messages([
11 SystemMessage(
12 content="""You are an expert image describer.
13 When presented with an image, provide a detailed,
14 accurate, and objective description of its visible
15 content. Focus on aspects such as:
16 - Objects present, their positions, and
17 relationships
18 - Colors, lighting, composition, and textures
19 - Actions or dynamics, if any (e.g., people
20 walking, water flowing)
21 - Contextual or inferred information (e.g.,
22 likely setting, era, or activity)
23
24 Avoid adding information that is not visible
25 or cannot be reasonably inferred from the image. Do
26 not speculate or inject personal opinion unless
27 explicitly requested. If text appears in the image,
28 transcribe it accurately.""""),
29 HumanMessage(content=[
30 {"type": "text", "text": "Describe the
31 following image for me:"},
32 {
33 "type": "image_url",
34 "image_url": {"url": f"data:{image_mime_type};base64,{image_b64}"}, "detail": "low"
35 }
36])
37]
38)
39 return image_describer_chat_template.invoke({})
40
41 class ImageDescriberInput(BaseModel):
42 text: str = Field(description="Path or URL to the
43 image in the format PNG or JPG/JPEG")
44
45 class ImageDescriberTool(BaseTool):
46 name: str = "image_describer"
47 description: str = "This tool can describe the image
48 in a detailed way"
49 args_schema: Optional[ArgsSchema] =
50 ImageDescriberInput
```

```

34 return_direct: bool = True
35
36 def _run(self, text: str, run_manager: Optional[
37 CallbackManagerForToolRun] = None) -> str:
38 """Use the tool."""
39 try:
40 parsed: ImageInput = extractor_chain.invoke({
41 "input": text})
42 except Exception as e:
43 return f"Failed to extract image URL: {str(e)}"
44
45 image_path_or_url = parsed.image_path_or_url
46 if not image_path_or_url:
47 return "No image URL found in the input."
48 output = image_describer_agent.invoke({
49 "image_path_or_url": image_path_or_url})
50 return output.image_description
51
52 async def _arun(self, image_path_or_url: str,
53 run_manager: Optional[AsyncCallbackManagerForToolRun]
54 = None) -> str:
55 """Use the tool asynchronously."""
56 return self._run(image_path_or_url, run_manager=
57 run_manager)

```

Trong đó, `ImageDescription` là output format của `image_describer_agent` mà mình quy định. Ở đây, ta định nghĩa output format chứa `image_description` dưới dạng string và miêu tả field này một cách cụ thể để mô hình có thể hiểu công dụng của field này.

Tiếp theo, ta thiết kế prompt cho `image_describer` gồm system message và human message. System message quy định ảnh sẽ được miêu tả cụ thể, chi tiết, rõ ràng, còn human message chứa ảnh từ người dùng.

Cuối cùng, chúng ta viết class `ImageDescriberTool` và implement hai method `_run` và `_arun` để định nghĩa cách tool hoạt động.

Ta dùng hàm `.invoke()` để chạy thử tool như sau:

```

1 image_describer_tool = ImageDescriberTool()
2 image_describer_tool.invoke("https://github.githubassets.
 com/assets/GitHub-Mark-ea2971cee799.png")
3 # Output: The image features a black circle with a white
 silhouette of a cat-like figure in the center. The
 figure has a rounded head with two pointed ears, a

```

```
 small rounded body, and a curved tail extending to the
 left. The design is simple and stylized, with no
 additional details or features.
```

Tương tự với image describer tool, ta implement một tool có khả năng detect object và đếm số lượng object có trong hình. Tool này sử dụng mô hình YOLOv11 để nhận dạng và đếm object.

Chúng ta load mô hình YOLO11x như sau:

```
1 yolo_model = YOLO("yolo11x.pt")
```

Sau đó, chúng ta có thể tạo tool từ một hàm với decorator `@tool` được cung cấp sẵn từ thư viện LangChain:

```
1 class ObjectDetectingAndCountingInput(BaseModel):
2 text: str = Field(description="Path or URL to the
3 image in the format PNG or JPG/JPEG")
4
5 @tool(
6 "detect_and_count_objects",
7 description="Detect and count objects within the
8 image. The return will be a dictionary, containing the
9 counting dictionary (counting how many instance of
10 each object class) and a list of dictionaries,
11 containing the object names, confidence scores, and
12 location in the image (in (x1, x2, y1, y2) format).",
13 args_schema=ObjectDetectingAndCountingInput
14)
15 def detect_and_count_object_tool(
16 text: Annotated[str, "Path or URL to the image"]
17):
18 """Detect objects within the image using YOLOv11
19 model"""
20
21 try:
22 parsed: ImageInput = extractor_chain.invoke({
23 "input": text})
24 except Exception as e:
25 return f"Failed to extract image URL: {str(e)}"
26
27 image_path_or_url = parsed.image_path_or_url
28 if not image_path_or_url:
29 return "No image URL found in the input."
30
31 results = yolo_model(image_path_or_url, verbose=False
32)
```

```

24
25 detections = []
26 counting = {}
27
28 # Process each result
29 for result in results:
30 boxes = result.boxes
31 class_names = result.names
32
33 for box in boxes:
34 class_id = int(box.cls[0])
35 class_name = class_names[class_id]
36 confidence = float(box.conf[0])
37 x1, y1, x2, y2 = map(int, box.xyxy[0])
38
39 detections.append({
40 "class": class_name,
41 "confidence": confidence,
42 "bbox": (x1, y1, x2, y2)
43 })
44
45 counting[class_name] = counting.get(
46 class_name, 0) + 1
47
48 return str({"counting": counting, "detections": detections})

```

Tương tự research agent ở phần trước, ta có thể tạo visual agent dựa trên pattern reasoning-acting của ReAct với hàm `create_react_agent` và hai tool

`image_describer_tool`, `detect_and_count_object_tool` như sau:

```

1 vision_agent = create_react_agent(
2 model="gpt-4o-mini",
3 tools=[image_describer_tool,
4 detect_and_count_object_tool],
5 prompt=(
6 "You are a vision agent.\n\n"
7 "INSTRUCTIONS:\n"
8 "- Assist ONLY with visual tasks (e.g.,
9 describing images, detecting and counting objects)\n"
10 "- Use only the tools provided to analyze visual
11 inputs\n"
12 "- After completing your task, respond to the
13 supervisor directly\n"

```

```

10 "- Respond ONLY with the results of your work, do
11 NOT include ANY other text."
12),
13 name="vision_agent"
14)

```

Ta có thể sử dụng visual agent như sau:

```

1 for chunk in vision_agent.stream(
2 {"messages": [{"role": "user", "content": "how many
3 dog in image: https://s3.amazonaws.com/cdn-origin-etr.
4 akc.org/wp-content/uploads/2018/04/24144817/American-
5 Staffordshire-Terrier-lying-outdoors-next-to-a-kitten-
6 that-is-playing-with-the-dogs-nose.jpg"}]}
7):
8 pretty_print_messages(chunk)

```

và kết quả nhận được là:

```

1 Update from node agent:
2
3
4 ===== Ai Message
5 =====
5 Name: vision_agent
6 Tool Calls:
7 detect_and_count_objects (call_ZMQi8IvMsQU8nL3vZZP2aIPh
8)
9 Call ID: call_ZMQi8IvMsQU8nL3vZZP2aIPh
10 Args:
11 text: https://s3.amazonaws.com/cdn-origin-etr.akc.org
12 /wp-content/uploads/2018/04/24144817/American-
13 Staffordshire-Terrier-lying-outdoors-next-to-a-kitten-
14 that-is-playing-with-the-dogs-nose.jpg
15
16
17 Found https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-
18 content/uploads/2018/04/24144817/American-
19 Staffordshire-Terrier-lying-outdoors-next-to-a-kitten-
20 that-is-playing-with-the-dogs-nose.jpg locally at
21 American-Staffordshire-Terrier-lying-outdoors-next-to-
22 a-kitten-that-is-playing-with-the-dogs-nose.jpg
23 Update from node tools:
24
25
26 ===== Tool Message
27 =====

```

```

18 Name: detect_and_count_objects
19
20 {"counting": {"dog": 2}, "detections": [{"class": "dog",
21 "confidence": 0.9411600232124329, "bbox": (287, 73,
22 618, 445)}, {"class": "dog", "confidence":
23 0.8354390263557434, "bbox": (159, 120, 369, 418)}]}
24
25
26 Update from node agent:
27 ====== Ai Message
28 ======
29 Name: vision_agent
30
31 2

```

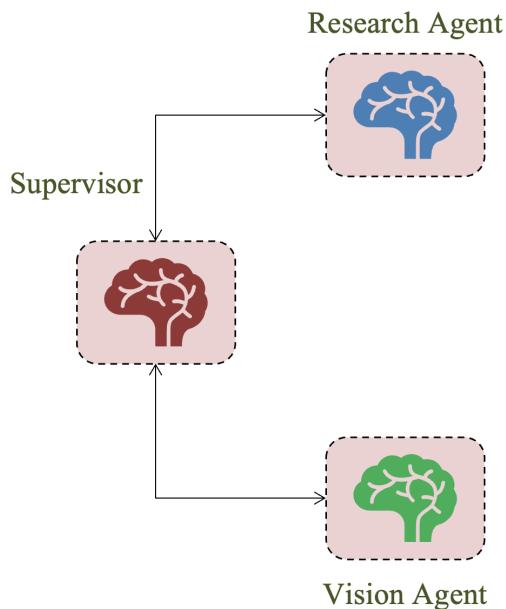
## 7. Xây dựng supervisor agent

Hai agent chúng ta vừa xây dựng hoạt động độc lập, riêng lẻ với nhau. Để tích hợp cả hai vào cùng một hệ thống và hệ thống tự động chuyển query của người dùng đến agent tương ứng có khả năng xử lý query đó, ta tạo một supervisor agent như sau:

```

1 supervisor = create_supervisor(
2 model=init_chat_model("gpt-4o-mini"),
3 agents=[research_agent, vision_agent],
4 prompt=(
5 "You are a supervisor managing two agents:\n"
6 "- research_agent: Use this agent ONLY for
7 research-related tasks (e.g., searching information,
8 finding papers, summarizing documents).\n"
9 "- vision_agent: Use this agent ONLY for visual
10 tasks (e.g., describing images, detecting and counting
11 objects).\n\n"
12 "RULES:\n"
13 "- Assign tasks to only one agent at a time.\n"
14 "- Do NOT call multiple agents in parallel.\n"
15 "- Do NOT perform any work yourself - always
16 delegate.\n"
17 "- Be concise when handing off tasks to agents,
18 only provide what is necessary for them to complete
19 the job."
20),
21 add_handoff_back_messages=True,
22)

```



Hình 63.7: Supervisor Agent.

```

15 output_mode="full_history", # or "final_output" if
 you want clean result
16).compile()

```

Trong đó, prompt định nghĩa system prompt của supervisor agent của chúng ta. Ta đưa vào hai agents là [research\_agent, vision\_agent] và sử dụng mô hình gpt-4o-mini làm bộ não của supervisor agent.

Kiến trúc của supervisor agent có thể được miêu tả trong ảnh 63.8.

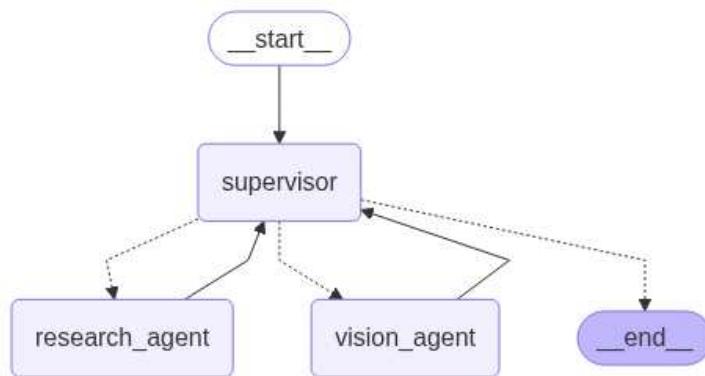
## 8. Inference

Để sử dụng hệ thống trên, ta có thể dùng chế độ stream để các message xuất hiện lần lượt trên màn hình như sau:

```

1 for chunk in supervisor.stream(
2 {
3 "messages": [
4 {
5 "role": "user",
6 "content": "What is the concept
visualized in the image? Image: https://huggingface.co
/datasets/tmnam20/Storage/resolve/main/rope.png

```



Hình 63.8: Supervisor Agent State Diagram.

```

 Provide me detailed information about the concept. If
 possible, give me some research papers about it.",
 }
]
 },
10) :
11 pretty_print_messages(chunk, last_message=True)
12
13 final_message_history = chunk["supervisor"]["messages"]

```

Kết quả được hiển thị như dưới đây:

```

1 Update from node supervisor:
2
3
4 ===== Tool Message
5 =====
5 Name: transfer_to_vision_agent
6
7 Successfully transferred to vision_agent
8
9
10 https://huggingface.co/datasets/tmnam20/Storage/resolve/
11 main/rope.png
11 Update from node vision_agent:
12
13
14 ===== Tool Message
15 =====
15 Name: transfer_back_to_supervisor

```

```
16
17 Successfully transferred back to supervisor
18
19
20 Update from node supervisor:
21
22
23 ===== Tool Message
24 =====
24 Name: transfer_to_research_agent
25
26 Successfully transferred to research_agent
27
28
29 Update from node research_agent:
30
31
32 ===== Tool Message
33 =====
33 Name: transfer_back_to_supervisor
34
35 Successfully transferred back to supervisor
36
37
38 Update from node supervisor:
39
40
41 ===== Ai Message
42 =====
42 Name: supervisor
43
44 The image visualizes the concept of Rotary Position
Embedding (RoPE) applied in transformer models. It
demonstrates how RoPE encodes positional information
by rotating query and key vectors.
45
46 **Detailed Information**:
47 - **Purpose**: RoPE enhances transformers' ability to
utilize relative positional information, crucial for
understanding sequences in natural language processing
and other tasks.
48 - **Mechanism**: The effect is achieved by manipulating
position matrices that rotate input vectors, leading
to improved contextual understanding of tokens in
layers of the transformer architecture.
```

```
49
50 **Research Papers**:
51 1. **Title**: Rotary Outliers and Rotary Offset Features
 in Large Language Models
 - **Author**: Andre Jonasson
 - **Published**: March 3, 2025
 - **Summary**: The paper investigates RoPE's impact on
 attention mechanisms, identifying patterns and
 anomalies in queries and keys and establishing the
 relevance of rotary embeddings across layer and
 architecture differences.
55
56 You can refer to various platforms like arXiv or Google
 Scholar for more research-related content on Rotary
 Position Embedding.
```

### 63.3 Câu hỏi trắc nghiệm

**Câu hỏi 153** “Agent” trong hệ thống AI là gì?

- a) Một tập dữ liệu dùng để huấn luyện mô hình
- b) Một công cụ toán học trong hệ thống học sâu
- c) Một hệ thống thông minh tương tác với môi trường
- d) Một API trung gian giữa người dùng và mô hình

**Câu hỏi 154** Cấu trúc ReAct Agent bao gồm gì?

- a) Hành động → Suy luận
- b) Tool → API → Trả lời
- c) Suy luận → Hành động → Quan sát → Lặp lại
- d) Prompt → Kết quả

**Câu hỏi 155** LangGraph là gì?

- a) Một công cụ visualize loss function
- b) Framework hỗ trợ mô hình hóa luồng tác vụ của LLM dưới dạng đồ thị
- c) Trình dịch mô hình LLM sang graph neural network
- d) Thư viện vẽ đồ thị toán học trong AI

**Câu hỏi 156** Kiến trúc supervisor trong multi-agent gồm đặc điểm nào sau đây?

- a) Mỗi agent tự quyết định đường đi riêng
- b) Supervisor điều phối và phân công agent con
- c) Supervisor thực hiện toàn bộ tác vụ
- d) Supervisor là một mô hình CNN

**Câu hỏi 157** Research Agent trong hệ thống Visual Agentic AI sử dụng những công cụ nào?

- a) Wikipedia và Bing
- b) Arxiv và Wikipedia
- c) Google Search và YouTube
- d) PubMed và GPT

**Câu hỏi 158** Vision Agent được thiết kế để làm gì?

- a) Viết văn bản sáng tạo

- b) Mô phỏng robot
- c) Phân tích ảnh: mô tả và đếm vật thể
- d) Tìm kiếm bài báo khoa học

**Câu hỏi 159** Trong kiến trúc LangGraph, “state” là gì?

- a) Một dạng memory cố định
- b) Không có ý nghĩa cụ thể
- c) Vùng lưu trữ kết quả trung gian giữa các node
- d) Công cụ xử lý hình ảnh

**Câu hỏi 160** Component nào chịu trách nhiệm chính điều phối agent trong hệ thống Visual Agentic AI?

- a) Research Agent
- b) Vision Agent
- c) Supervisor Agent
- d) LangChain Router

**Câu hỏi 161** Vision Agent sử dụng mô hình nào để detect object?

- a) ResNet50
- b) YOLOv11
- c) BERT-Image
- d) RetinaNet

**Câu hỏi 162** Điều nào sau đây KHÔNG phải là lợi ích của việc sử dụng kiến trúc multi-agent?

- a) Dễ kiểm thử và bảo trì
- b) Dễ dàng mở rộng chức năng mới
- c) Loại bỏ hoàn toàn nhu cầu dùng LLM
- d) Cho phép chuyên môn hóa từng tác vụ

## 63.4 Phụ lục

1. **Hint:** Dựa vào file mô tả trong thư mục sau [Visual Agentic AI](#) để hoàn thiện các đoạn code.
2. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về [tại đây](#) (Lưu ý: Sáng khi hết deadline phần project, admin mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Code:** Tài liệu code public [Repo](#)
4. Tài liệu tham khảo:
  - [ReAct: Synergizing Reasoning and Acting in Language Models](#)
  - [LangGraph](#)

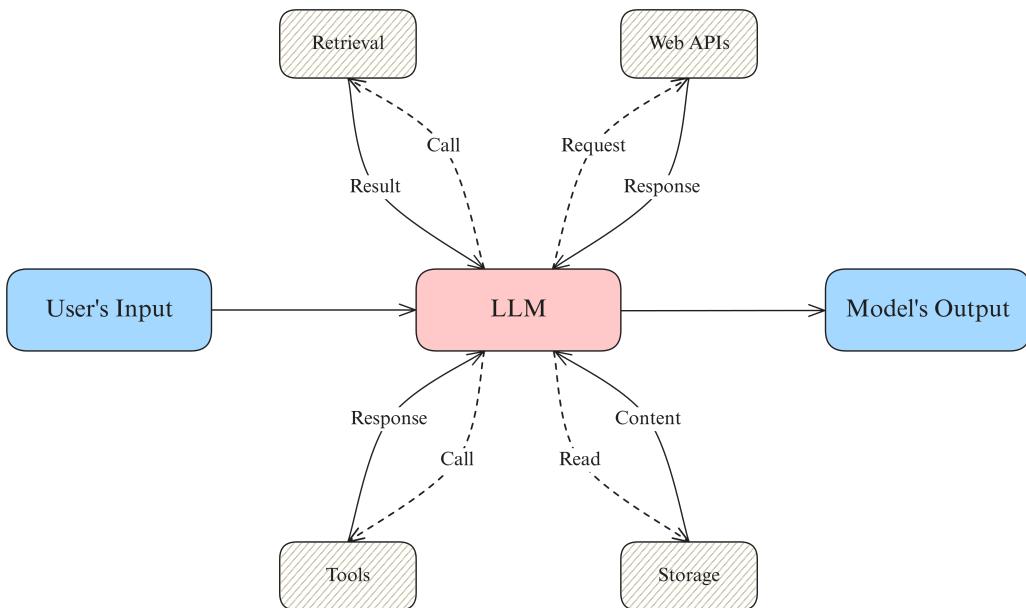
# Chương 64

## Giao thức MCP

### 64.1 Giới thiệu

Khi các ứng dụng AI assistant ngày càng được sử dụng rộng rãi, rất nhiều nghiên cứu đã đầu tư vào việc nâng cao năng lực của mô hình, từ đó đạt được những bước tiến lớn về khả năng suy luận (reasoning ability) và hiệu suất của trên các benchmarks thực tế. Tuy nhiên, ngay cả những mô hình tiên tiến nhất vẫn bị hạn chế bởi sự tách biệt khỏi dữ liệu: mô hình không được tiếp tục huấn luyện để cập nhật những thông tin mới nhất. Ví dụ như GPT-4o có giới hạn dữ liệu huấn luyện đến tháng 6 năm 2024 hay của mô hình Llama 4 là tháng 8 năm 2024.

Để giải quyết vấn đề này, chúng ta tích hợp thông tin mới nhất vào mô hình bằng cách tìm kiếm thông tin liên quan đến vấn đề cần giải quyết từ các nguồn dữ liệu bên ngoài như cơ sở dữ liệu, kho tri thức, internet, v.v., và tích hợp những thông tin đó vào trong prompt của chúng ta trước khi gửi request đến mô hình (ví dụ như hệ thống RAG). Điều này cho phép mô hình có thể gián tiếp truy cập vào những thông tin không chứa trong knowledge.



Hình 64.1: LLM tích hợp tri thức ngoài

Tuy nhiên, mỗi công cụ hoặc cơ sở dữ liệu đều có cách thức truy cập và tương tác khác biệt, từ việc cấu trúc API, truy vấn SQL, đến các giao thức kết nối khác nhau. Điều này dẫn đến một thách thức lớn trong việc xây dựng các ứng dụng AI có thể kết nối với nhiều nguồn dữ liệu và công cụ khác nhau một cách hiệu quả, đó là việc dữ liệu bị phân mảnh (data fragmentation). Mỗi mô hình AI cần phải được tích hợp riêng với từng nguồn dữ liệu, dẫn đến việc developers phải viết code lặp đi lặp lại và khó khăn trong việc bảo trì cũng như thêm các tools khác vào mô hình.

Để giải quyết vấn đề này, **Model Context Protocol (MCP)** được phát triển như một giao thức thống nhất. MCP chuẩn hóa các kết nối giữa mô hình AI với các nguồn dữ liệu và công cụ khác nhau, giúp giảm thiểu sự phức tạp trong việc tích hợp và cho phép các mô hình AI truy cập vào dữ liệu một cách hiệu quả hơn.

Trong bài tutorial này, chúng ta sẽ tìm hiểu về Model Context Protocol (MCP), kiến trúc và các thành phần của MCP, và cuối cùng là xây dựng một công cụ tích hợp vào mô hình AI của chúng ta với MCP.

## 64.2 Model Context Protocol

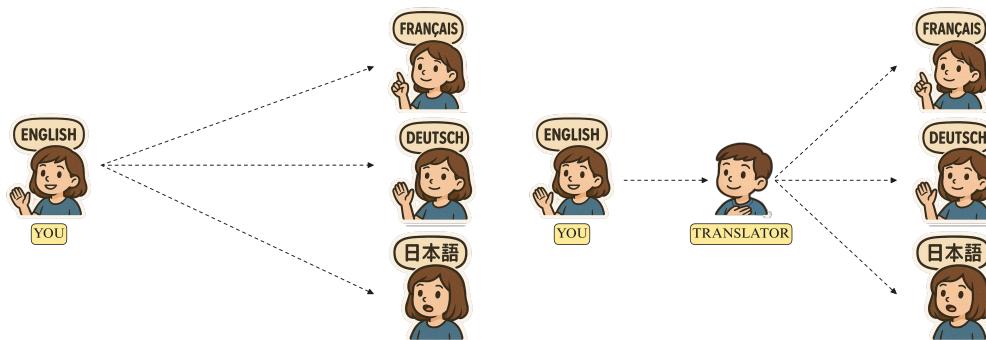
### 64.2.1 Tổng quan

Model Context Protocol là gì?

**Model Context Protocol (MCP)** là một giao thức mở, giúp chuẩn hóa kết nối giữa các ứng dụng cung cấp dữ liệu và các mô hình ngôn ngữ lớn. Chúng ta có thể hình dung MCP giống như một cổng USB-C dành riêng cho các ứng dụng AI. Nếu cổng USB-C giúp chúng ta dễ dàng kết nối thiết bị điện tử của chúng ta (như laptop, điện thoại, v.v) với nhiều phụ kiện khác nhau (ổ cứng di động, màn hình, action camera, v.v.), thì MCP cũng giúp chúng ta kết nối các mô hình ngôn ngữ lớn với những nguồn dữ liệu (databases) và công cụ (tools) theo một quy chuẩn được định sẵn.

Để minh họa cho vai trò của Model Context Protocol (MCP), ta có thể hình dung một tình huống thực tế như sau:

- Giả sử bạn chỉ biết tiếng Anh, và trong một buổi tiệc giao lưu, bạn gặp gỡ ba người bạn quốc tế đến từ các nước khác nhau. Một người chỉ nói được tiếng Pháp, một người chỉ nói tiếng Đức, và người còn lại chỉ nói tiếng Trung Quốc (Hình 64.2a).
- Bạn rất muốn kết bạn và trao đổi thông tin với từng người, nhưng rào cản ngôn ngữ khiến việc này trở nên cực kỳ khó khăn. Nếu bạn quyết định tự mình học từng ngôn ngữ một, bạn sẽ phải đầu tư rất nhiều thời gian, công sức và có thể không hiệu quả ngay tức thì. Ngược lại, nếu từng người bạn mới cố gắng học tiếng Việt, thì thử thách cũng không hề nhỏ.
- Tuy nhiên, giả sử trong buổi tiệc đó xuất hiện thêm một người phiên dịch thành thạo cả bốn ngôn ngữ: tiếng Việt, tiếng Anh, tiếng Đức và tiếng Pháp. Người phiên dịch này trở thành trung gian giúp bạn giao tiếp dễ dàng với từng người bạn mới mà không cần trực tiếp học thêm ngôn ngữ mới nào.
- Cuộc trò chuyện diễn ra vô cùng suôn sẻ nhờ sự hỗ trợ của người phiên dịch, mọi hiểu lầm hoặc khó khăn đều được giải quyết nhanh chóng (Hình 64.2b). Bạn cảm thấy thoải mái hơn, tiết kiệm thời gian và công sức hơn.



Hình 64.2: Ví dụ về MCP thông qua hình ảnh thông dịch viên

Trong bối cảnh xây dựng các ứng dụng AI, Model Context Protocol đóng vai trò tương tự như người phiên dịch. MCP hoạt động như một cầu nối trung gian, giúp các hệ thống, ứng dụng và dữ liệu “nói chuyện” và hiểu nhau một cách nhanh chóng và hiệu quả mà không cần phải mất thời gian thay đổi hoặc điều chỉnh từng thành phần riêng lẻ.

## Tại sao MCP ra đời?

Để hiểu tại sao MCP ra đời, chúng ta sẽ đi tìm hiểu về function calling.

**Function calling** Function calling là một tính năng của các mô hình ngôn ngữ lớn (LLM), cho phép các mô hình không chỉ dừng lại ở việc tạo ra văn bản mà còn có thể tương tác với các công cụ và dịch vụ bên ngoài. Khi chúng ta hỏi một chatbot về thời tiết hôm nay, thay vì chỉ đưa ra câu trả lời chung chung (ví dụ như: “Tôi không thể trả lời câu hỏi của bạn vì tôi không có kiến thức về thời tiết ngày hôm nay”), mô hình có thể gửi request đến API thời tiết để lấy thông tin chính xác và được cập nhật theo thời gian thực.

Về bản chất, function calling giúp LLM có khả năng thực hiện các hành động cụ thể thông qua việc gọi các hàm được định nghĩa trước. Cơ chế này rất hữu ích khi chúng ta cần xây dựng các ứng dụng có sự tương tác phức tạp như chatbot hỗ trợ khách hàng có thể tra cứu đơn hàng, AI assistant có thể gửi email, hoặc hệ thống phân tích có thể truy vấn cơ sở dữ liệu.

Function calling chủ yếu được sử dụng trong 3 trường hợp chính như sau:

1. Mở rộng kiến thức: Mô hình có thể truy cập và khai thác dữ liệu từ các nguồn bên ngoài như cơ sở dữ liệu, API, kho kiến thức chuyên biệt, v.v., giúp cung cấp thông tin mới theo thời gian thực.
2. Nâng cao chức năng: Bằng cách gửi request đến các công cụ hỗ trợ như máy tính, thư viện đồ họa, biểu đồ, v.v., mô hình có thể thực hiện tính toán và khai thác dữ liệu dễ dàng hơn.
3. Thực hiện hành động: Mô hình có thể tương tác trực tiếp với hệ thống qua API để thực hiện các hành động như đặt lịch, tạo hóa đơn, gửi email, điều khiển thiết bị smarthome, v.v. Tất cả đều được thực hiện tự động và chính xác dựa theo lệnh của mô hình.

**Cách function calling hoạt động** Để ví dụ về việc function calling hỗ trợ nâng cao chức năng của LLM và đồng thời hiểu được quy trình hoạt động của function calling, chúng ta hãy so sánh việc sử dụng LLM có function calling và không có function calling để tính toán giá trị của biểu thức toán học sau đây:

$$9.896 - 4.012 + (-13.23456908) - (-2^{-1.05}) \quad (64.1)$$

Trước khi tích hợp function calling, ta sử dụng mô hình Gemini 2.0 Flash để tính toán biểu thức trên:

#### Dùng LLM giải quyết bài toán trên khi không dùng thêm tool ngoài

```

1 from google import genai
2 from google.genai import types
3
4 client = genai.Client(api_key="your_api_key")
5 config = types.GenerateContentConfig(temperature=0.0, top_k=1,
 max_output_tokens=10)
6
7 response = client.models.generate_content(
8 model="gemini-2.0-flash",
9 contents="Just give me the answer directly without calling any
 external tools or support: 9.896 -
 4.012 + (-13.23456908) - (- 2**(-

```

```

10 config=config,
11)
12
13 print(response.text)

```

và nhận được kết quả như sau:

### Kết quả khi dùng LLM mà không có external tool để giải bài toán trên

-7.17

Tuy nhiên, nếu sử dụng máy tính, kết quả chính xác là  $-6.867600915537576$ , khác với kết quả mô hình trả về là  $-6.86828308$ .

Bây giờ, chúng ta sẽ tích hợp function calling vào LLM để tính toán biểu thức trên theo các bước như sau:

0. Ta sẽ định nghĩa và miêu tả hàm `evaluate_math_expression` để hàm nhận vào một biểu thức dưới dạng string như sau:

#### Định nghĩa và miêu tả hàm `evaluate_math_expression`

```

1 # Function Declaration
2 def evaluate_math_expression(expression: str, precision: int =
3 None) -> float:
4 allowed_names = {k: v for k, v in math.__dict__.items() if
5 not k.startswith("__")}
6 allowed_names.update({"abs": abs, "round": round, "min":
7 min, "max": max, "pow": pow})
8
9 try:
10 result = eval(expression, {"__builtins__": {}},
11 allowed_names)
12 if precision is not None:
13 return round(result, precision)
14 return result
15 except Exception as e:
16 raise ValueError(f"Error evaluating expression: {e}")
17
18 # Function Definition

```

```

15 evaluate_math_expression_function = {
16 "name": "evaluate_math_expression",
17 "description": "Evaluates a mathematical expression given
 as a string and returns the
 numeric result.",
18 "parameters": {
19 "type": "object",
20 "properties": {
21 "expression": {
22 "type": "string",
23 "description": "The mathematical expression to
 evaluate, e.g., '3 * (4 + 5) /
 2'.",
24 },
25 "precision": {
26 "type": "number",
27 "description": "Number of decimal places to
 round the result to.",
28 },
29 },
30 "required": ["expression"],
31 },
32 }

```

Lưu ý rằng định nghĩa hàm (Function Definition) được đưa vào cùng với request gửi đến LLM. Còn việc khai báo hàm (Function Declaration) là để thực thi hàm khi hàm được gọi đến.

- Chúng ta sẽ gửi request đến LLM cùng với dữ liệu về hàm đã định nghĩa trước đó:

#### Request đến LLM

```

1 tools = types.Tool(function_declarations=[
 evaluate_math_expression_function
])
2 config = types.GenerateContentConfig(tools=[tools],
 temperature=0.0, top_k=1)
3
4 conversation = [
5 types.Content(
6 role="user", parts=[types.Part(text="9.896 - 4.012 +

```

```

7)
8]
9
10 response = client.models.generate_content(
11 model="gemini-2.0-flash",
12 contents=conversation,
13 config=config,
14)
15 print(response.candidates[0].content.parts[0].function_call)

```

2. Khi mô hình quyết định gọi hàm, nó sẽ trả về tên hàm cần gọi cùng các đối số (input arguments) dưới dạng dữ liệu có cấu trúc:

#### Kết quả trả về với lệnh function calling

FunctionCall(id=None, args='expression': '9.896 - 4.012 + (-13.23456908) - (- 2\*\*(- 1.05))', name='evaluate\_math\_expression')

3. Chúng ta tiến hành phân tích phản hồi từ mô hình để trích xuất hàm được chọn và xử lý việc gọi hàm tương ứng, cụ thể ở đây là hàm evaluate\_math\_expression.

#### Thực thi hàm evaluate\_math\_expression với arguments được trả về

```

1 tool_call = response.candidates[0].content.parts[0].
2 function_call
3
4 if tool_call.name == "evaluate_math_expression":
5 result = evaluate_math_expression(**tool_call.args)
6 print(f"Function execution result: {result}")

```

và output nhận được như sau:

**Kết quả thực thi hàm**

Function execution result: -6.867600915537576

4. Chúng ta cung cấp kết quả gọi hàm cho mô hình để mô hình có thể tích hợp chúng vào phản hồi cuối cùng của mình và gửi conversation đã update kết quả function call đến LLM như sau:

**Gửi kết quả chạy hàm cho LLM trong luồng conversation**

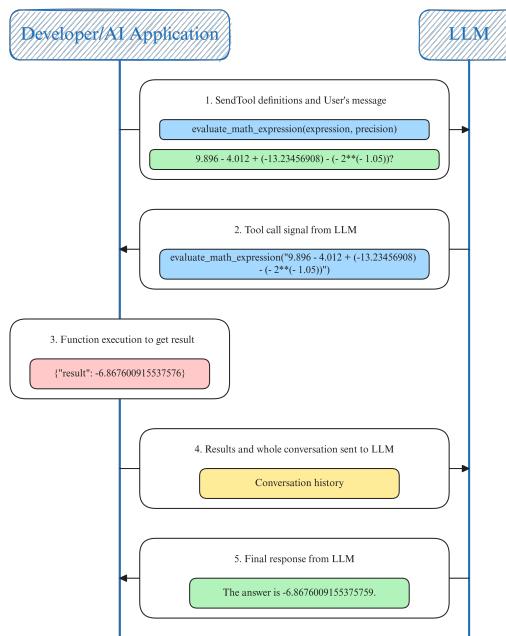
```
1 # Append function call and result of the function execution to
 the conversation
2 function_response_part = types.Part.from_function_response(
3 name=tool_call.name,
4 response={"result": result},
5)
6 conversation.append(response.candidates[0].content)
7 conversation.append(types.Content(role="user", parts=[
8 function_response_part]))
9
10 final_response = client.models.generate_content(
11 model="gemini-2.0-flash",
12 contents=conversation,
13 config=config,
14)
15 print(final_response.text)
```

5. Cuối cùng, chúng ta nhận được kết quả như sau:

**Kết quả trả về**

The answer is -6.867600915537576.

Như vậy, thông qua function calling, chúng ta đã tăng thêm khả năng tính toán cho LLM. Ngoài ra, chúng ta có thể tích hợp vô số hàm khác nhau như gửi email, kiểm tra thời tiết, truy vấn cơ sở dữ liệu. Từ đó, ta thấy function calling hỗ trợ LLM trong việc tương tác với các công cụ, giúp LLMs giải quyết được những vấn đề còn tồn đọng trong bản thân mô hình như không thể tính toán chính xác, khó khăn trong quá trình tương tác trực tiếp với môi trường, v.v.



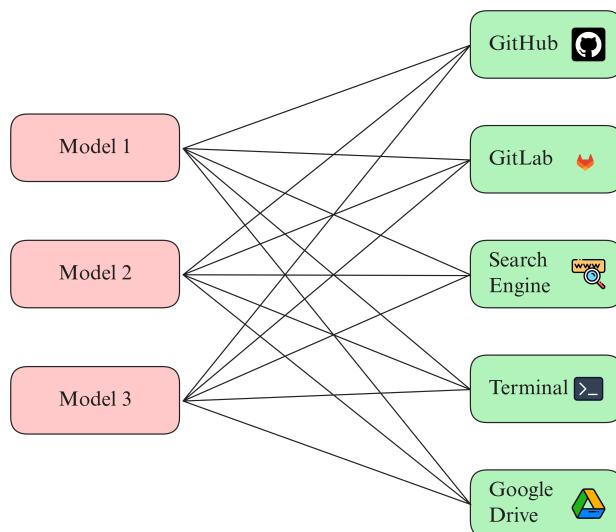
Hình 64.3: Function Calling Workflow

Quy trình làm việc của function calling ở trên có thể được minh họa từng bước như trong Hình 64.3. Chúng ta có thể thấy tools được implement và thực thi ở bên phía AI applications của chúng ta. **Nếu chúng ta có sự thay đổi về mặt logic của tools, cấu trúc của tools hay số lượng tham số của tools, chúng ta phải viết lại tools cũng như định nghĩa của tools.** Cuối cùng là khởi động lại ứng dụng AI.

**Vấn đề của function calling và cách MCP giải quyết bài toán**  
 Function calling hỗ trợ LLM trong việc tăng khả năng xử lý thông tin một cách chính xác, giúp LLM có thể hành động và tương tác với môi trường, bổ sung kiến thức mới cho mô hình. Tuy nhiên, vấn đề nảy sinh khi ta cần tích hợp nhiều tools vào các ứng dụng AI của chúng ta.

Trước khi MCP ra đời, để kết nối một mô hình AI với nhiều nguồn dữ liệu (như GitHub, GitLab, databases, local file system, v.v.) hoặc công cụ (như web browser, Slack, Google Drive, v.v.), các nhà phát triển phải viết nhiều connector riêng biệt. Các công cụ pháp lục đó thường mang tính thủ công như: implement logic riêng cho từng công cụ, design prompt để mô hình gọi riêng các tool, hoặc viết tool để gọi API đến nền tảng.

Hệ quả là một bài toán tích hợp nổi tiếng: bài toán  $M \times N$ . Nghĩa là, nếu có  $M$  mô hình AI và  $N$  nguồn dữ liệu hay công cụ khác nhau, thì chúng ta cần xây dựng đến  $M \times N$  kết nối riêng lẻ, dẫn tới việc vừa phức tạp vừa khó mở rộng quá trình tích hợp các công cụ vào ứng dụng AI (minh họa trong Hình 64.4). Bên cạnh đó, các tools được implement và thực thi ở phía ứng dụng AI (xem Hình 64.3), dẫn tới việc phải khởi động lại ứng dụng AI khi có tools mới hoặc cập nhật các tools có sẵn.



Hình 64.4: Mỗi mô hình AI phải kết nối đến từng tools

Trong ví dụ về function calling ở phần trước, hàm `evaluate_math_expression` được định nghĩa với hai tham số `expression` và `precision`, và nếu tham số `precision` không được thiết lập, hàm sẽ trả về số chữ số thập phân một cách tùy ý. Tuy nhiên, ở một ứng dụng AI khác, hàm dùng để tính toán biểu thức có thể được định nghĩa dưới tên khác (ví dụ như `get_expression_value`, chỉ nhận một tham số đầu vào là biểu thức dưới dạng chuỗi, và trả về kết quả là giá trị được làm tròn đến hai chữ số thập phân dưới dạng chuỗi (xem đoạn code minh họa 64.2.1)). Trong trường hợp hàm `get_expression_value` có sự thay đổi về mặt implementation để tích hợp thêm argument `precisions` (đại diện cho số chữ số thập phân sau dấu phẩy), developers phải sửa lại toàn bộ hàm, viết định nghĩa cho tool mới và khởi động lại ứng dụng AI. Những rắc rối trên bắt nguồn từ việc chúng ta implement tools theo những cách khác nhau, dẫn đến sự bất đồng bộ trong

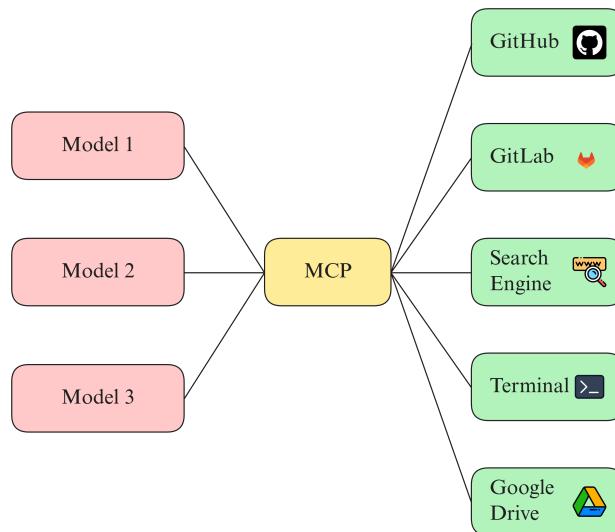
việc quản lí và sử dụng tools đó.

### Một cách implement khác của hàm evaluate\_math\_expression

```
1 import ast
2 import operator as op
3
4 def get_expression_value(expression: str) -> str:
5
6 def _eval_node(node):
7 # Supported operators
8 OPERATORS = {
9 ast.Add: op.add,
10 ast.Sub: op.sub,
11 ast.Mult: op.mul,
12 ast.Div: op.truediv,
13 ast.Mod: op.mod,
14 ast.Pow: op.pow,
15 ast.USub: op.neg,
16 ast.UAdd: op.pos,
17 ast.FloorDiv: op.floordiv,
18 }
19 if isinstance(node, ast.Num): # for Python <3.8
20 return node.n
21 elif isinstance(node, ast.Constant): # for Python >=3.8
22 if isinstance(node.value, (int, float)):
23 return node.value
24 raise TypeError(f"Unsupported constant type: {type(node.value).__name__}")
25 elif isinstance(node, ast.BinOp):
26 left = _eval_node(node.left)
27 right = _eval_node(node.right)
28 operator = OPERATORS.get(type(node.op))
29 if operator is None:
30 raise TypeError(f"Unsupported binary operator: {type(node.op).__name__}")
31 return operator(left, right)
32 elif isinstance(node, ast.UnaryOp):
33 operand = _eval_node(node.operand)
34 operator = OPERATORS.get(type(node.op))
35 if operator is None:
36 raise TypeError(f"Unsupported unary operator: {type(node.op).__name__}")
37 return operator(operand)
```

```
38 else:
39 raise TypeError(f"Unsupported expression type: {type(
40 node).__name__}")
41
42 node = ast.parse(expression, mode="eval")
43 result = _eval_node(node.body)
44 result = round(result, 2) # Round to two decimal places
45 return f"{result:.2f}"
46
47 print(get_expression_value("9.896 - 4.012 + (-13.23456908) - (- 2
48 *(- 1.05))"))
OUTPUT
-6.87
```

MCP giải quyết vấn đề này bằng cách đưa ra một giao thức chuẩn ở vị trí trung gian giữa LLM và các công cụ. Thay vì phải xây dựng  $M \times N$  tích hợp riêng biệt giữa từng ứng dụng AI và từng công cụ, chúng ta chỉ cần  $M + N$  lần triển khai: mỗi ứng dụng AI chỉ cần tích hợp MCP ở phía client một lần, và mỗi công cụ hoặc nguồn dữ liệu chỉ cần triển khai MCP ở phía server một lần (xem Hình 64.5). Nhờ đó, tất cả mọi người trong cuộc trò chuyện đều “dùng chung một ngôn ngữ”. Khi cần kết nối một AI mới với một công cụ mới, chúng ta không cần viết thêm code để integrate các tools vào mô hình mới này vì chúng đã có thể “giao tiếp với nhau nhau” thông qua MCP.

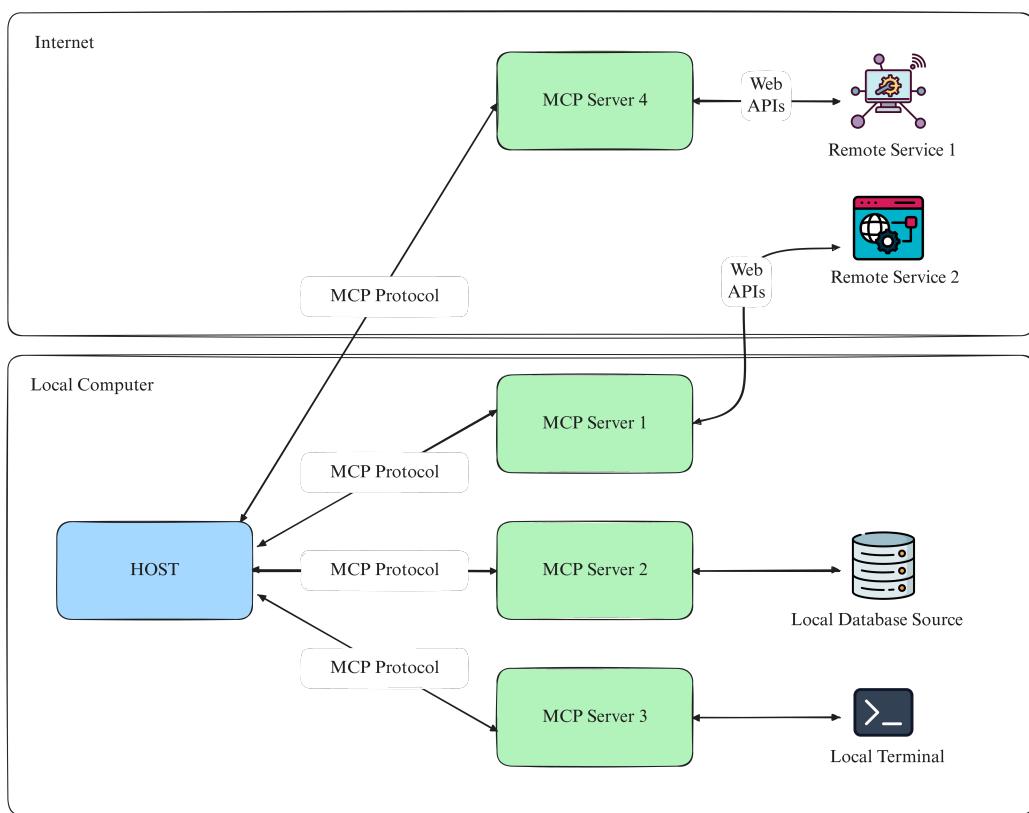


Hình 64.5: MCP giải quyết bài toán tích hợp tools vào các mô hình AI

### 64.2.2 Kiến trúc của MCP

Về bản chất, MCP vận hành theo kiến trúc client-server, trong đó một ứng dụng chủ (host) có thể kết nối đến nhiều máy chủ (server) khác nhau thông qua các clients (xem Hình 64.6). Cấu trúc này bao gồm các thành phần sau:

- **MCP Host:** Là các chương trình mà người dùng sử dụng như Claude Desktop, các IDE (Cursor, Codeium, Windsurf, v.v.) hoặc các ứng dụng AI có nhu cầu truy cập dữ liệu thông qua MCP.
- **MCP Client:** Là các clients tuân theo giao thức MCP. Mỗi MCP client chỉ giữ kết nối 1:1 với một MCP duy nhất.
- **MCP Server:** Là những chương trình nhẹ, chịu trách nhiệm cung cấp các năng lực chuyên biệt thông qua giao thức MCP.
- **Nguồn dữ liệu cục bộ (Local Data Sources):** Bao gồm tệp tin, cơ sở dữ liệu và dịch vụ đang chạy trên máy tính của chúng ta. Đây là một loại database mà các MCP servers có thể truy cập khi có sự đồng ý của người dùng.
- **Dịch vụ từ xa (Remote Services):** Là các hệ thống bên ngoài có



Hình 64.6: General MCP Architecture

thể kết nối qua Internet (thường qua API), và MCP server cũng có thể truy cập những dịch vụ này.

Kiến trúc lõi của MCP gồm ba thành phần là: Host, MCP client, và MCP server. Trong đó, giao tiếp giữa MCP client và MCP server được xử lý bởi transport layer và các messages truyền đi đều tuân theo chuẩn [JSON-RPC 2.0](#). Để hỗ trợ triển khai MCP server linh hoạt trong nhiều tình huống khác nhau, MCP hỗ trợ các cơ chế truyền tải như sau:

1. **Stdio Transport:** Giao tiếp thông qua luồng chuẩn đầu vào/đầu ra (standard i/o) và phù hợp cho các tiến trình chạy cục bộ trên Host. Ưu điểm của cơ chế truyền tải này là đơn giản, hiệu quả, không yêu cầu kết nối mạng.
2. **Streamable HTTP Transport:** Sử dụng giao thức HTTP để truyền dữ liệu giữa client và server. Phương thức này hỗ trợ Server-Sent Events (SSE) để truyền dữ liệu dạng streaming từ server về client. Các yêu cầu từ client đến server được gửi qua HTTP POST.

Trong Hình [64.6](#), kết nối giữa các MCP Client ở Host và MCP Server 4 là dùng Streamable HTTP Transport, còn kết nối giữa các MCP Client và MCP Server 1, MCP Server 2, và MCP Server 3 có thể dùng Stdio Transport hoặc Streamable HTTP Transport (khi đó, IP address của MCP Server là <http://localhost>).

## Host

**Host** là ứng dụng AI của người dùng, nơi mà các mô hình AI hoạt động và tương tác trực tiếp với chúng ta. Host có thể là các môi trường lập trình (IDE) tích hợp AI như Cursor, Windsurf, GitHub Copilot, v.v., hoặc là ứng dụng chatbot như ChatGPT và Claude Desktop, hoặc là ứng dụng trợ lý ảo tích hợp AI.

Trong hệ thống MCP, Host đóng vai trò là thành phần chủ động khởi tạo kết nối đến các MCP server khi cần. Host thực hiện các nhiệm vụ sau:

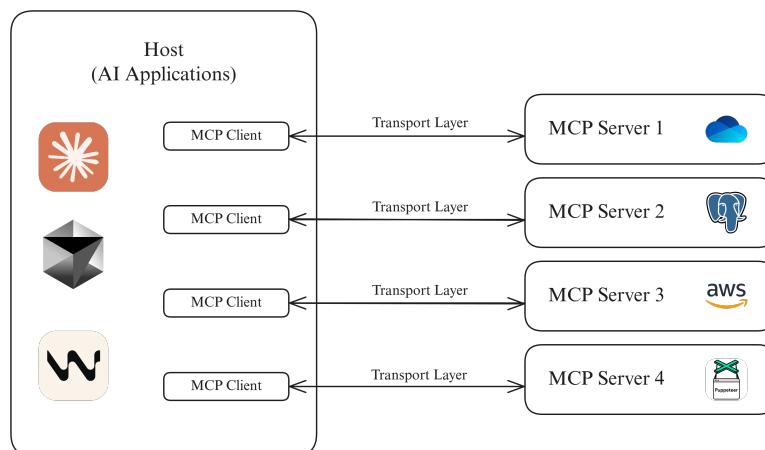
- Tạo và quản lý nhiều clients hoạt động song song, từ đó kết nối tới nhiều MCP servers cùng một lúc.

- Kiểm soát quyền kết nối và vòng đời của từng client.
- Áp dụng các chính sách bảo mật cũng như các yêu cầu liên quan đến quyền truy cập dữ liệu.
- Xử lý các quyết định ủy quyền từ phía người dùng khi một client yêu cầu quyền truy cập đến dữ liệu cá nhân của người dùng.
- Điều phối quá trình tích hợp mô hình AI/LLM, bao gồm quá trình sinh ra phản hồi khi có lệnh từ phía MCP server (sampling).

### MCP Client

**MCP client** là một thành phần trung gian đóng vai trò "phiên dịch", hoạt động bên trong ứng dụng Host. Nếu ví Host như "bộ não" – nơi đưa ra quyết định và chỉ đạo hành động – thì MCP client chính là "người thông dịch", giúp truyền tải ý định của Host theo ngôn ngữ của giao thức MCP đến server, đồng thời mang kết quả trở lại.

Mỗi MCP client thiết lập và duy trì kết nối 1:1 với một MCP server duy nhất. Khi Host cần làm việc với nhiều services khác nhau (ví dụ: lưu trữ tệp, truy vấn cơ sở dữ liệu, phân tích dữ liệu), nó sẽ khởi tạo từng MCP client riêng biệt cho từng MCP server (xem Hình 64.7). Cách thiết kế này giúp cô lập từng kết nối, ngăn chặn rò rỉ dữ liệu và đảm bảo các sessions không bị chồng chéo.



Hình 64.7: Host, MCP Clients và MCP Servers

Khi một MCP client kết nối đến MCP server, quá trình bắt đầu bằng một bước handshake – tức là thương lượng giao thức giữa hai bên. Tại đây, client và server thống nhất về các tính năng được hỗ trợ, bao gồm: cách thức mã hóa thông tin, định dạng truyền và nhận tin nhắn, các công cụ mà MCP server cung cấp,... Sau khi thương lượng thành công, cả hai bên sẽ duy trì kết nối một cách liên tục để phục vụ việc giao tiếp sau đó.

Trong suốt phiên làm việc, MCP client thực hiện các nhiệm vụ sau:

- Truyền request từ host đến MCP server,
- Nhận phản hồi từ Server và gửi ngược về Host,
- Quản lý các đăng ký hoặc thông báo theo thời gian thực, ví dụ khi server chủ động gửi cập nhật cho Host.

Tóm lại, ứng dụng host giữ vai trò điều phối, quyết định khi nào và sử dụng MCP client nào cho từng tác vụ cụ thể. Trong khi đó, mỗi MCP client đảm nhiệm trọng trách kết nối và tương tác với MCP server, bao gồm việc thiết lập kết nối, giao tiếp theo chuẩn MCP, bảo mật và duy trì đối thoại với MCP server. Nhờ vậy, host có thể tập trung vào quá trình tương tác với người dùng, trong khi các MCP client lo toàn bộ phần tương tác với MCP server phía dưới.

## MCP Server

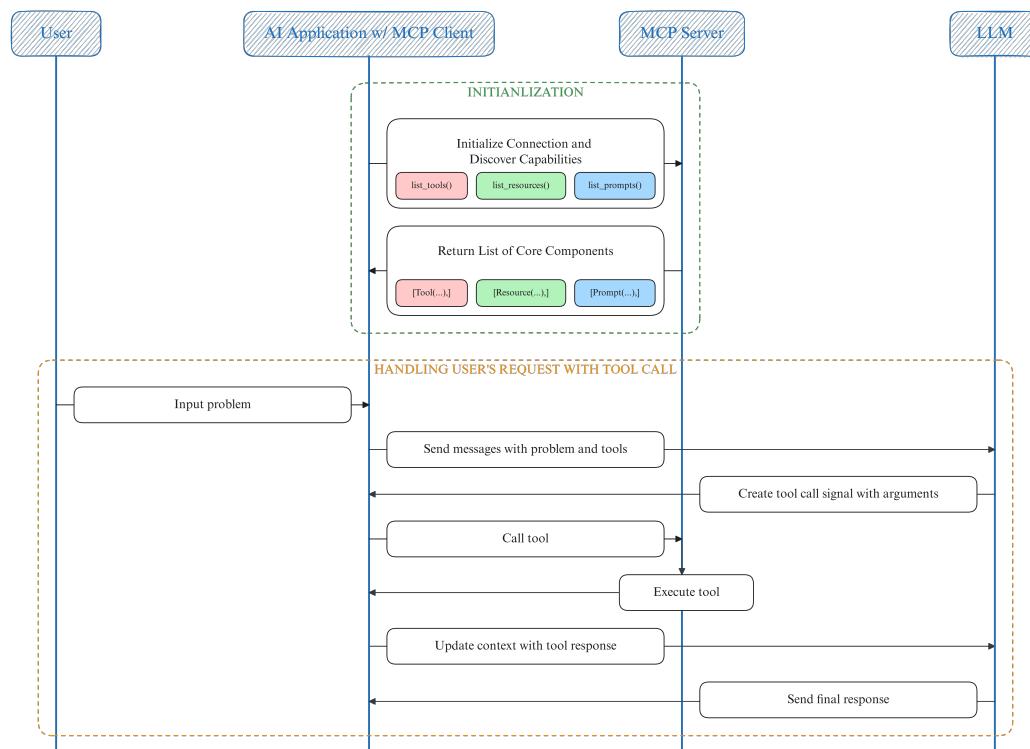
MCP server là một dịch vụ ngoài (không nằm chung với Host), đóng vai trò như một wrapper hoặc bộ chuyển đổi (adapter) cho các kho dữ liệu cục bộ, cơ sở dữ liệu, API services, cloud services, v.v.. MCP server đóng gói các khả năng chuyên biệt như công cụ, kho dữ liệu, hoặc hành động, và công bố chúng thông qua một giao diện chuẩn hóa do MCP định nghĩa. Với cấu trúc đồng nhất, MCP server cho phép các Host dễ dàng khám phá, gọi thực thi, và nhận kết quả trả về theo định dạng chuẩn.

Thông thường, MCP server cung cấp ba loại tài nguyên mà MCP client có thể sử dụng như sau:

1. **Resources (Tài nguyên dữ liệu):** Đại diện cho ngữ cảnh dữ liệu – chẳng hạn như nội dung file, bản ghi cơ sở dữ liệu, hoặc metadata của tài liệu.

2. **Tools:** Là các hàm có thể gọi được – ví dụ như: `queryCustomers()`, `readFile()`, `findTickets()`, hoặc `execute_query()`.
3. **Prompts:** Là các mẫu tin nhắn được định nghĩa sẵn, dùng để hướng dẫn mô hình trong quá trình suy luận hoặc gọi lệnh.

## MCP Workflow



Hình 64.8: MCP Workflow

Khi chúng ta khởi động AI application (vd như Claude Desktop), chúng ta sẽ khởi tạo kết nối tới MCP server và lấy những tài nguyên mà MCP server cung cấp, bao gồm tools, resources và prompts.

Khi người dùng gửi yêu cầu bài toán, AI application sẽ gửi thông tin bài toán và danh sách các tài nguyên của MCP server đến LLM để LLM quyết định có gọi tool hoặc dùng một tài nguyên khác. Nếu LLM có sử dụng tool,

lúc này LLM sẽ gửi về ứng dụng AI của chúng ta gồm tên tool cần chạy và các arguments. AI application sẽ gửi yêu cầu chạy tool đến MCP server, MCP server chạy tool và trả kết quả cho AI application. Khi nhận được kết quả chạy tool, AI application sẽ update context để đưa kết quả vào conversation và gửi đến LLM để LLM quyết định bước tiếp theo. Quá trình này có thể lặp đi lặp lại cho đến khi LLM sinh ra kết quả cuối cùng.

### 64.2.3 MCP Client Features

#### Sampling

Trong MCP, **Sampling** là cơ chế cho phép một MCP server chủ động yêu cầu MCP client sử dụng mô hình ngôn ngữ (LLM) của chính phía client để sinh ra nội dung mới – thường là văn bản. Đây là một *mô hình đảo ngược vai trò* đặc biệt: thay vì chỉ có client gọi server, thì lúc này, server gọi ngược lại client để khai thác năng lực sinh ngôn ngữ.

Cụ thể, khi một MCP server cần một câu trả lời phức tạp (như tóm tắt dữ liệu, lập kế hoạch, sáng tác câu trả lời, v.v.), thay vì tự thực hiện hoặc kết nối với LLM API bên ngoài, server có thể gửi một yêu cầu `sampling/createMessage` cho client – nội dung yêu cầu chứa các tin nhắn đầu vào (`messages`) và tham số bổ sung. Client, sở hữu mô hình ngôn ngữ đã cấu hình sẵn, sẽ thực hiện việc "sample" (tức là sinh ra câu trả lời dưới dạng văn bản) và trả kết quả về server.

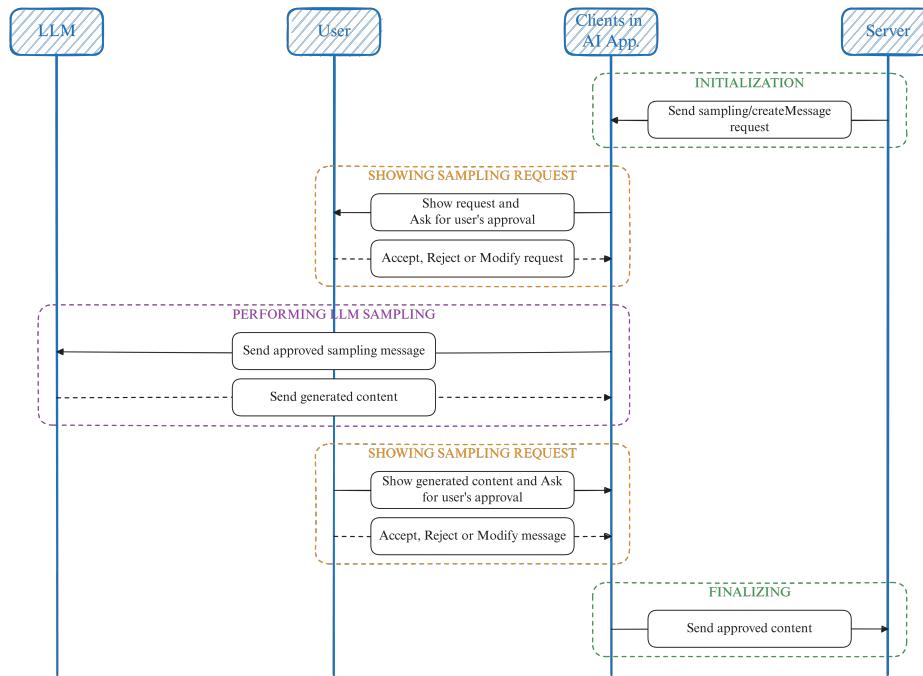
Cơ chế sampling mang lại nhiều lợi ích quan trọng như sau:

- **Tiết kiệm chi phí và hạ tầng:** MCP server không cần triển khai một mô hình AI riêng hay phải setup một luồng gọi API. Thay vào đó, MCP server sử dụng LLM của MCP client như một cách tận dụng tài nguyên đã sẵn có.
- **Tăng tính linh hoạt và agentic:** MCP server có thể tự sinh ra câu trả lời dựa trên ngữ cảnh của tình huống, ví dụ: “Tôi nên làm gì tiếp theo với dữ liệu này?”, từ đó server có thể linh hoạt tương tác với người dùng giống như AI agent.
- **Giữ quyền kiểm soát cho người dùng:** Vì sampling luôn được thực hiện bởi MCP client, người dùng có thể kiểm duyệt prompt hoặc output messages trước khi nó được gửi trả về MCP server.

- **Giữ cho thiết kế của server nhẹ và dễ triển khai:** MCP server không cần cấu hình API, triển khai một LLM riêng, hay logic phức tạp mà chỉ cần biết khi nào cần nhờ “bộ não” phía client hỗ trợ quá trình xử lý thông tin.

**Sampling workflow** Quy trình sampling hoạt động được mô tả qua 5 bước sau (xem Hình 64.9):

1. Server gửi yêu cầu sampling: Thông qua RPC `sampling/createMessage`, server truyền chuỗi tin nhắn và chỉ thị để mô hình tiếp tục sinh ra câu trả lời.
2. Client đánh giá và xử lý yêu cầu: MCP client có thể hiển thị nội dung prompt cho người dùng xác nhận, từ chối hoặc chỉnh sửa nếu thấy không an toàn.
3. Gọi mô hình ngôn ngữ: Nếu được chấp thuận, client dùng LLM đã cấu hình để sinh ra nội dung tiếp theo.
4. Hậu xử lý nội dung sinh ra bởi LLM: Client có thể kiểm duyệt, cắt ngắn, lọc nội dung để đảm bảo phù hợp với đối tượng người dùng (vd như loại bỏ nội dung tiêu cực, độc hại, trích xuất keywords, tìm kiếm nội dung tương tự, v.v.).
5. Gửi kết quả về server: Câu trả lời được sinh ra từ LLM được đóng gói dưới dạng `CreateMessageResult` và gửi trả cho server để sử dụng cho các bước tiếp theo.



Hình 64.9: Sampling Workflow trong MCP

Ví dụ dưới đây minh họa một tool trong MCP server sử dụng sampling để viết một bài thơ với chủ đề do người dùng cung cấp:

### Sampling sử dụng LLM phía client

```

1 from mcp.server.fastmcp import Context, FastMCP
2 from mcp.types import SamplingMessage, TextContent
3
4 mcp = FastMCP(name="Sampling Example")
5
6 @mcp.tool()
7 async def generate_poem(topic: str, ctx: Context) -> str:
8 """Generate a short poem about the topic using LLM sampling."""
9 prompt = f"Write a short poem about {topic}."
10 result = await ctx.session.create_message(
11 messages=[
12 SamplingMessage(
13 role="user",
14 content=TextContent(type="text", text=prompt),

```

```
15)
16],
17 max_tokens=100
18)
19 return result.content.text if result.content.type == "text" else
 str(result.content)
```

Trong đó,

- Biến `ctx` chứa phiên kết nối `session`, từ đó gửi request đến LLM ở phía client bằng method `create_message()`.
- Prompt được cấu trúc thành một `SamplingMessage` từ prompt template và tham số `topic`.
- Có thể tùy chỉnh các tham số đi kèm trong quá trình sampling như `max_tokens`, `temperature`, `stop_sequences`, hoặc `includeContext`.

**Sampling use cases** Chúng ta có thể sử dụng sampling cho các trường hợp sau:

- **Tóm tắt kết quả:** Sau khi tool lấy dữ liệu, server có thể yêu cầu LLM tạo tóm tắt: “Tóm tắt dữ liệu sau...”.
- **Lập kế hoạch tác vụ:** Trong chuỗi agent, sampling được dùng để hỏi LLM “Tiếp theo nên làm gì?” sau mỗi bước xử lý.
- **Tạo câu trả lời tự nhiên:** Các tool như `explain_concept()` có thể dùng LLM sinh lời giải thích thay vì hard-code.
- **Đàm phán giữa agents:** Một MCP server dùng AI agent có thể dùng sampling để mô phỏng phản ứng của agent khác dựa trên prompt do mình xây dựng.
- **Phân tích kết quả tìm kiếm:** Sau khi gọi tool tìm kiếm, server có thể dùng sampling để tổng hợp hoặc đưa khuyến nghị dựa trên nội dung tìm được.

Tóm lại, sampling là cơ chế chiến lược của MCP client, cho phép MCP server kết hợp năng lực của LLM bên phía MCP client để tạo nên các quy trình AI linh hoạt, mở rộng và mang tính tác tử (agentic). Nhờ vậy, các AI agents có thể thực hiện tác vụ phức tạp mà không cần triển khai một mô hình ngôn ngữ riêng trong server.

### Elicitation

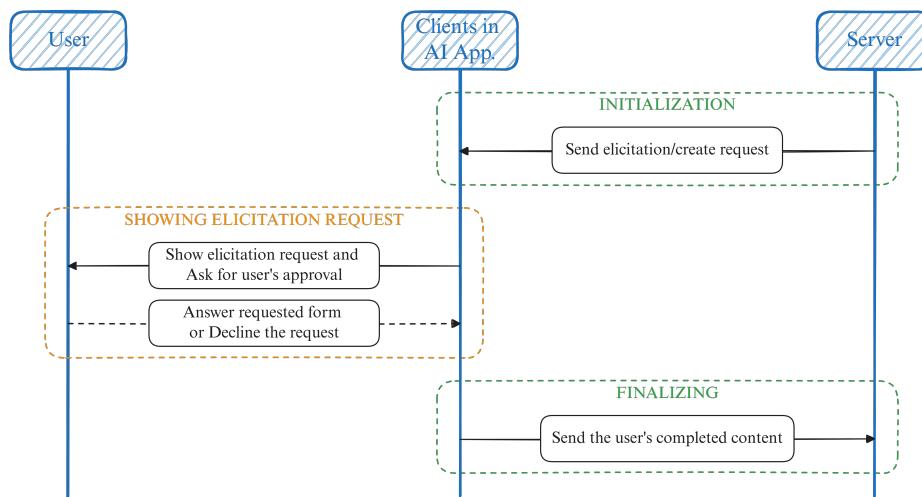
Trong quá trình tương tác giữa Host, MCP client và MCP server, có những tình huống mà server cần thêm thông tin từ người dùng để tiếp tục xử lý một tác vụ. Trong những trường hợp đó, cơ chế **elicitation** (gợi hỏi) của MCP được sử dụng. Đây là tính năng cho phép MCP server tạm dừng thực thi một công cụ, chủ động yêu cầu client hiển thị một biểu mẫu tương tác cho người dùng, và chờ người dùng điền bổ sung thông tin cần thiết rồi mới tiếp tục quá trình thực thi.

Khác với cách truyền thống là server phải nhận đủ toàn bộ input ngay khi gọi method, cơ chế elicitation tạo ra khả năng tương tác theo nhiều lượt (multi-turn interaction), từ đó giúp các workflow trở nên linh hoạt và thân thiện hơn với người dùng. Tính năng này đặc biệt hữu ích trong các công cụ phức tạp, cần xác thực thông tin hoặc xác nhận hành vi của người dùng tại thời điểm sử dụng.

**Elicitation Workflow** Luồng hoạt động của một phiên elicitation tuân theo các bước sau (minh họa trong Hình 64.10):

1. **Server gửi yêu cầu elicitation:** MCP server sử dụng một lời gọi JSON-RPC đặc biệt (`elicitation/create`) kèm theo một thông điệp hướng dẫn người dùng và một schema định nghĩa cấu trúc dữ liệu mong đợi.
2. **Client hiển thị message:** MCP client tiếp nhận yêu cầu và hiển thị một biểu mẫu tương tác đến người dùng (trong Host), với các fields cần nhập được hiển thị tự động dựa trên schema.
3. **Người dùng phản hồi:** Người dùng có thể chọn điền thông tin và xác nhận (accept), từ chối (decline), hoặc hủy bỏ thao tác (cancel).

4. **Client gửi kết quả về server:** Nếu người dùng xác nhận, dữ liệu sẽ được validate theo schema và gửi lại cho server. Ngược lại, nếu từ chối hoặc hủy, response sẽ mang thông tin tương ứng.
5. **Server tiếp tục xử lý:** MCP server nhận kết quả và quyết định tiếp tục thực thi tool, hủy bỏ, hoặc chuyển hướng theo logic xử lý.



Hình 64.10: Elicitation Workflow trong MCP

**Elicitation use cases** Chúng ta có một số tình huống áp dụng elicitation như sau:

- Thu thập thông tin còn thiếu: ví dụ khi người dùng không cung cấp đủ tham số, cung cấp thêm thông tin cho search engine, nhập email mới khi email đã điền bị trùng, v.v.
- Làm rõ ý định: như xác định username nào khi có nhiều người trùng tên.
- Xác nhận hành động nguy hiểm: ví dụ xác nhận xóa tệp hoặc gửi email hàng loạt.
- Giao tiếp theo bước: như quá trình cài đặt một phần mềm không có sẵn trên máy, biểu mẫu nhiều trang, hoặc cấu hình hệ thống trong nhiều file.

**Elicitation Implementation and Demo** Chúng ta có ví dụ sau về cách thiết kế elicitation trong MCP server:

### Elicitation

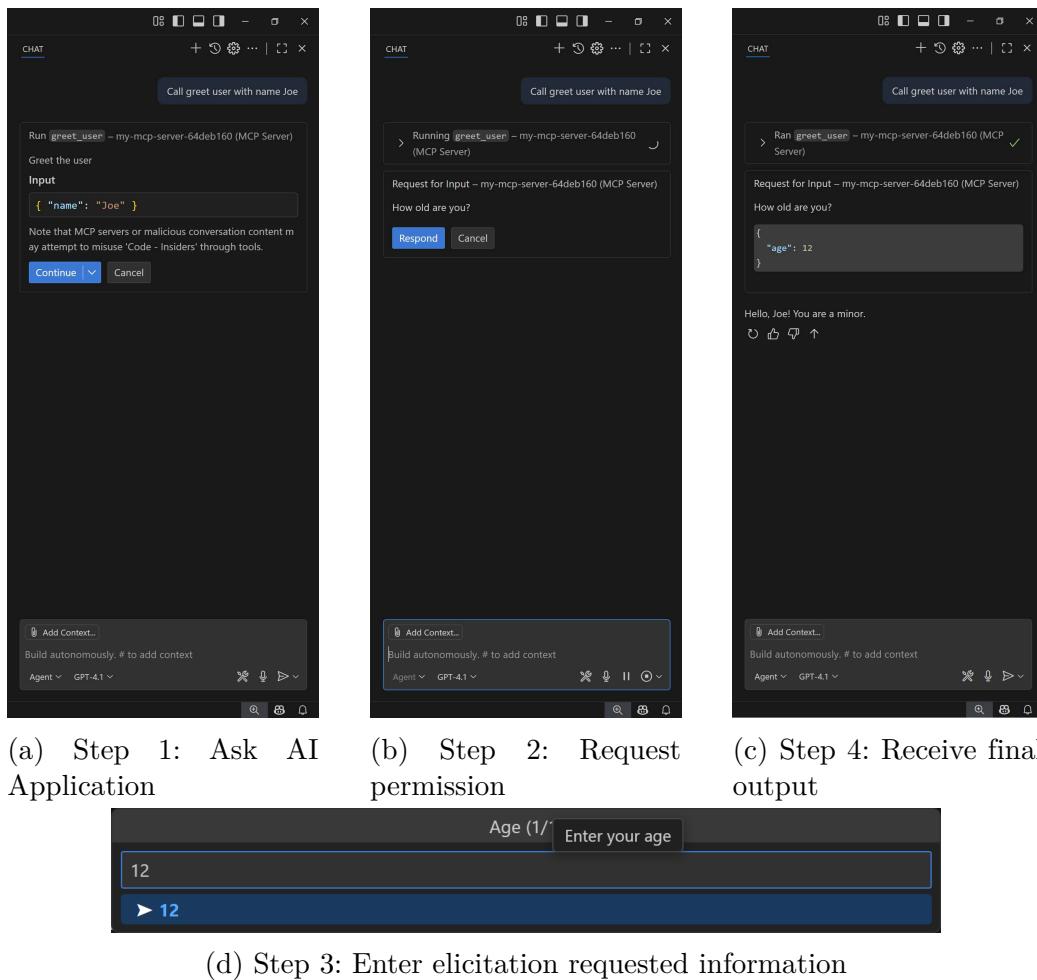
```

1 from pydantic import BaseModel, Field
2 from mcp.server.fastmcp import Context, FastMCP
3
4 class AgeAskingSchema(BaseModel):
5 age: int = Field(description="Enter your age", ge=0, le=120)
6
7 mcp = FastMCP(name="Server with Elicitation Example")
8
9 @mcp.tool(name="greet_user", description="Greet the user")
10 async def greet_user(name: str, ctx: Context) -> str:
11 greeting_name = f"Hello, {name}!"
12
13 asking_age_result = await ctx.elicit(
14 message="How old are you?",
15 schema=AgeAskingSchema,
16)
17
18 if asking_age_result.action == "accept" and asking_age_result.
19 data:
20 if asking_age_result.data.age < 18:
21 greeting_name += " You are a minor."
22 else:
23 greeting_name += " You are an adult."
24 elif asking_age_result.action == "reject":
25 greeting_name += " You chose not to provide your age."
26
27 return greeting_name
28
29 if __name__ == "__main__":
30 mcp.run(transport="streamable-http")

```

Khi tool `greet_user` được gọi, server sẽ tạo một prompt chào người dùng bằng tên đã nhập, rồi dùng cơ chế elicitation thông qua `ctx.elicit()` để hỏi thêm về độ tuổi với schema `AgeAskingSchema`. Nếu người dùng đồng ý cung cấp thông tin, server sẽ kiểm tra xem họ dưới hay trên 18 tuổi để đưa ra lời nhắn phù hợp. Nếu người dùng từ chối trả lời, tool vẫn hoạt động bình thường và thêm một câu thông báo rằng họ không muốn tiết lộ tuổi.

Kết quả quá trình gọi tool `greet_user` với cơ chế elicitation được minh họa trong Hình 64.11. Ví dụ này minh họa rõ cách MCP sử dụng elicitation để thu thập thông tin bổ sung một cách linh hoạt, tôn trọng quyền lựa chọn của người dùng và thích ứng theo tình huống.



Hình 64.11: MCP Elicitation demo on Visual Studio Code Insider

Elicitation là một tính năng quan trọng trong MCP client giúp kết nối chặt chẽ hơn giữa AI application và người dùng. Cơ chế này góp phần mang lại trải nghiệm tương tác tự nhiên, linh hoạt, thay vì buộc người dùng phải đoán trước và cung cấp mọi thông tin ngay từ đầu.

## Roots

Trong MCP, **Roots** (gốc truy cập) là một cơ chế quan trọng nhằm định nghĩa rõ phạm vi hoạt động của MCP server. Một MCP root (gọi tắt là root về sau trong bài) về bản chất là một URI (Uniform Resource Identifier) – có thể là đường dẫn tới thư mục cục bộ hoặc một endpoint từ xa – mà Host cung cấp để hướng dẫn MCP server biết phạm vi nào được phép truy cập và xử lý dữ liệu. Ta có ví dụ định nghĩa về roots như sau:

```
{
 "roots": [
 {
 "uri": "file:///home/user/projects/myapp",
 "name": "Project Directory"
 },
 {
 "uri": "https://api.example.com/v1",
 "name": "API Endpoint"
 }
]
}
```

Code Listing 64.1: Ví dụ về cách định nghĩa roots

Ví dụ trên chỉ định rằng MCP server chỉ nên hoạt động trong thư mục dự án ở local với đường dẫn /home/user/projects/myapp và endpoint API https://api.example.com/v1. Thông tin này được truyền từ MCP client đến MCP server trong quá trình khởi tạo phiên làm việc hoặc khi có thay đổi bằng cách gửi các roots đã update thông qua các routes như `roots/list` hoặc thông báo `rootsListChanged`.

Một điểm cần lưu ý là MCP không bắt buộc server phải tuân thủ tuyệt đối roots và đây là một cơ chế mang tính *thỏa thuận tin cậy* (informational contract). Tuy nhiên, các MCP server được thiết kế đúng chuẩn sẽ luôn tuân theo phạm vi roots đã được chỉ định, đặc biệt khi xử lý tài nguyên hệ thống như tệp tin hoặc endpoint nhạy cảm. Ngoài ra, roots có thể được gán thêm tên hiển thị thân thiện (friendly name), phục vụ cho UI/UX ở phía client hoặc để phân biệt trong môi trường multi-root.

Vai trò của roots bao gồm:

- **Hướng dẫn phạm vi truy cập (Scope Guidance):** MCP server sẽ

biết chính xác nên tập trung vào phạm vi nào trong hệ thống tập tin hoặc các endpoint từ xa. Ví dụ, khi hỗ trợ lập trình, chỉ cần phân tích và đọc file trong thư mục dự án được chỉ định, bỏ qua toàn bộ thư mục và các file khác không liên quan (như file .env).

- **Giới hạn quyền truy cập (Security Boundary):** Giống như một sandbox, roots giới hạn việc MCP server truy cập ngoài vùng cho phép. Server sẽ không được phép đọc hoặc thao tác với các tài nguyên ngoài danh sách roots do client cung cấp.

Chúng ta có thể liệt kê các trường hợp sử dụng roots phổ biến như sau:

- **Coding workspace:** Hạn chế MCP server chỉ thao tác trong thư mục workspace của người dùng, ví dụ như các thư mục `src` hay `source` của dự án hiện tại,
- **Workspace chứa tài liệu:** Chỉ định thư mục chứa tài liệu Word, Markdown, Notion API workspace, hoặc các tài liệu nội bộ khác.
- **Endpoint API:** giới hạn các API được gọi của MCP server.
- **Multi-root:** Hỗ trợ người dùng làm việc trên nhiều vùng dữ liệu hoặc dự án song song (ví dụ: hai repo Git khác nhau).

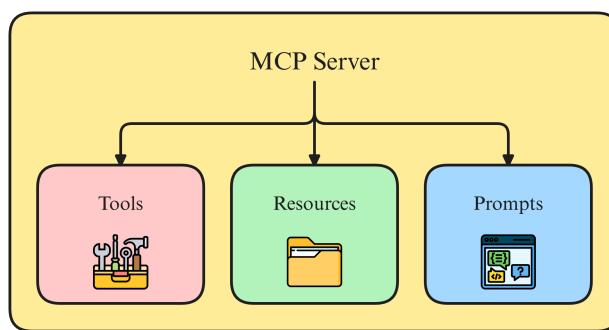
Tóm lại, roots trong MCP client là một cơ chế bảo vệ và định hướng quan trọng, vừa đảm bảo tính an toàn truy cập, vừa giúp AI agent hoạt động đúng ngữ cảnh. Việc sử dụng roots đúng cách chính là cách để triển khai nguyên tắc **least privilege** – chỉ cấp quyền vừa đủ cho tác vụ cần thiết.

#### 64.2.4 MCP Server Features

MCP server đóng vai trò là nền tảng cơ bản để bổ sung ngữ cảnh cho các mô hình AI thông qua ba thành phần cơ bản là tools, resources và prompts. Chúng ta có thể tóm gọn miêu tả về từng thành phần cơ bản như sau:

- **Tools:** Các hàm có thể thực thi, cho phép mô hình thực hiện hành động hoặc truy xuất thông tin.
- **Resources:** Dữ liệu có cấu trúc hoặc nội dung bổ sung ngữ cảnh cho mô hình,

- **Prompts:** Các template hoặc hướng dẫn được định nghĩa sẵn nhằm định hướng cách mô hình ngôn ngữ phản hồi,



Hình 64.12: MCP Server core primitives

Trong phần này, chúng ta sẽ vừa tìm hiểu về các thành phần cơ bản của một MCP server, vừa xây dựng một MCP server hỗ trợ quản lý sách với thư viện [MCP](#).

Đầu tiên, ta có một class `LibraryManagement` để quản lý những quyền sách trong thư viện

### Class LibraryManagement

```

1 class LibraryManagement:
2 def __init__(self, books_path: Path):
3 self.books_path = books_path
4 if not books_path.exists():
5 books_path.write_text("[]", encoding="utf-8")
6 self.books = json.loads(books_path.read_text(encoding="utf-8"))
7
8 def save_books(self):
9 self.books_path.write_text(json.dumps(self.books, indent=4),
10 encoding="utf-8")
11
12 def add_book(self, book: Book) -> str:
13 if any(b["isbn"] == book.isbn.strip() for b in self.books):
14 return f"Book with ISBN '{book.isbn}' already exists."
15
16 if not all([book.title.strip(), book.author.strip(), book.

```

```
 isbn.strip()]):
16 return "Title, author, and ISBN cannot be empty."
17
18 clean_tags = [t.strip() for t in book.tags if isinstance(t,
19 str) and t.strip()]
20 self.books.append(
21 {
22 "title": book.title.strip(),
23 "author": book.author.strip(),
24 "isbn": book.isbn.strip(),
25 "tags": clean_tags,
26 }
27)
28 self.save_books()
29 return f"Book '{book.title}' by {book.author} added to the
30 library."
31
32 def remove_book(self, isbn: str) -> str:
33 updated = [b for b in self.books if b["isbn"] != isbn.strip()
34 ()]
35 if len(updated) == len(self.books):
36 return f"No book found with ISBN '{isbn}'."
37 self.books = updated
38 self.save_books()
39 return f"Book with ISBN '{isbn}' removed from the library."
40
41 def get_num_books(self) -> int:
42 return len(self.books)
43
44 def get_all_books(self) -> list:
45 return self.books
46
47 def get_book_by_index(self, index: int) -> dict:
48 if 0 <= index < len(self.books):
49 return self.books[index]
50 return {"error": "Book not found."}
51
52 def get_book_by_isbn(self, isbn: str) -> dict:
53 for b in self.books:
54 if b["isbn"] == isbn.strip():
55 return b
56 return {"error": "Book not found."}
57
58 def get_suggesting_random_book_prompt(self) -> str:
```

```
56 return "Suggest a random book from the library. The
57 suggestion should include the
58 title, author, and a brief
59 description."
60
61 def get_suggesting_book_title_by_abstract_prompt(self, abstract:
62 str) -> str:
63 return f"Suggest a memorable, descriptive title for a book
64 based on the following abstract: {
65 abstract}"
66
67 def get_analyzing_book_messages(self, book: dict, query: str) ->
68 list[dict[str, str]]:
69 return [
70 {
71 "role": "user",
72 "content": "This is the book I want to analyze: " +
73 json.dumps(book),
74 },
75 {
76 "role": "assistant",
77 "content": "Sure! Let's analyze this book together.
78 What would you like to know?",
79 },
80 {"role": "user", "content": query},
81]
```

Class này có các methods sau:

- **save\_books**: Ghi toàn bộ danh sách sách hiện tại vào tệp JSON, đảm bảo mọi thay đổi (thêm hoặc xoá sách) được lưu lại một cách nhất quán.
- **add\_book**: Thêm một quyển sách mới vào thư viện. Hàm sẽ kiểm tra tính hợp lệ của các trường dữ liệu (title, author, ISBN) và tránh trùng lặp ISBN. Nếu thành công, dữ liệu sách được ghi vào danh sách và cập nhật vào tệp lưu trữ.
- **remove\_book**: Xoá một quyển sách dựa trên ISBN. Nếu không tìm thấy sách phù hợp, hàm trả về thông báo lỗi. Nếu xoá thành công, cập nhật lại dữ liệu lưu trữ.

- `get_num_books`: Trả về tổng số sách hiện có trong thư viện.
- `get_all_books`: Trả về toàn bộ danh sách sách dưới dạng một danh sách Python.
- `get_book_by_index`: Lấy thông tin một quyển sách theo chỉ số (index) trong danh sách. Nếu chỉ số không hợp lệ, trả về thông báo lỗi.
- `get_book_by_isbn`: Trả về thông tin của một quyển sách dựa trên ISBN. Nếu không tìm thấy, trả về thông báo lỗi.
- `get_suggesting_random_book_prompt`: Trả về một chuỗi lệnh nhắc (prompt) yêu cầu gợi ý một quyển sách ngẫu nhiên từ thư viện.
- `get_suggesting_book_title_by_abstract_prompt`: Nhận vào một đoạn abstract của quyển sách và tạo ra một prompt để gợi ý tiêu đề sách phù hợp, mang tính mô tả và ấn tượng.
- `get_analyzing_book_messages`: Tạo ra một danh sách các tin nhắn (theo cấu trúc trò chuyện giữa user và assistant) nhằm phục vụ phân tích một quyển sách thông qua mô hình hội thoại. Cấu trúc tin nhắn phù hợp với định dạng sử dụng trong các ứng dụng như ChatGPT hay các hệ thống trò chuyện thông minh.

Dựa vào ứng dụng trên, ta sẽ thiết kế một MCP server phục vụ việc quản lý sách trong thư viện. Đi qua mỗi core component của MCP server, chúng ta sẽ điền dần vào đoạn code dưới đây:

#### MCP Server implementation mà chúng ta sẽ điền vào

```
1 import json
2 from pathlib import Path
3 from typing import Dict, Sequence
4
5 from mcp.server.lowlevel import Server
6 from mcp.server.stdio import stdio_server
7 from mcp.types import (
8 EmbeddedResource,
9 GetPromptResult,
10 ImageContent,
11 Prompt,
12 PromptArgument,
```

```
13 PromptMessage,
14 Resource,
15 ResourceTemplate,
16 TextContent,
17 Tool,
18)
19 from pydantic import BaseModel, Field
20 from pydantic.networks import AnyUrl
21
22 class LibraryManagement:
23 ...
24
25 async def serve() -> None:
26 books_path = Path("books.json")
27 library = LibraryManagement(books_path)
28
29 server = Server("mcp-library")
30
31 ##### TOOLS #####
32 @server.list_tools()
33 async def list_tools() -> list[Tool]:
34 """List all available tools for the library management
35 system."""
36
37 ...
38
39 @server.call_tool()
40 async def call_tool(
41 name: str, arguments: dict
42) -> Sequence[TextContent | ImageContent | EmbeddedResource]:
43 """Call a specific tool by name with the provided arguments.
44
45 """
46
47 ...
48
49
50 @server.list_resources()
51 async def list_resources() -> list[Resource]:
52 """List all available resources."""
53
54 ...
```

```

55 @server.read_resource()
56 async def read_resource(uri: AnyUrl) -> str:
57 """Read a resource by its URI."""
58 ...
59
60 ##### PROMPTS #####
61 @server.list_prompts()
62 async def list_prompts() -> list[Prompt]:
63 """List all available prompts."""
64 ...
65
66 @server.get_prompt()
67 async def get_prompt(name: str, arguments: Dict[str, str] | None
68) -> GetPromptResult:
69 """Get a specific prompt by name."""
70 ...
71
72 options = server.create_initialization_options()
73 print(f"Server is running with options: {options}")
74
75 async with stdio_server() as (read_stream, write_stream):
76 await server.run(read_stream, write_stream, options)
77
78 if __name__ == "__main__":
79 import asyncio
80
81 asyncio.run(serve())

```

## Tools

**Tools trong MCP Server** là một thành phần cốt lõi, cho phép các mô hình AI thực hiện các hành động thực tế thông qua việc tương tác với các hàm được thiết kế để thực thi ở phía server.

Giống như resources, tools được định danh bằng các unique name và có thể kèm theo mô tả để hướng dẫn mô hình AI chọn đúng để thực thi. Tuy nhiên, khác với resources, tools đại diện cho các thao tác động thông qua việc chúng có khả năng thay đổi trạng thái, thông tin của MCP server hoặc tương tác với các hệ thống bên ngoài khác.

Chúng ta có các ví dụ thực tế về tool ở các MCP server sau:

- [Google Drive MCP Server](#): cung cấp tool `search(query)` để tìm kiếm các file tương ứng với query trong Google Drive,
- [Git MCP Server](#): có các tool quản lý và tương tác với các Git repo như `git_init(repo_path)` nhằm khởi tạo repo, `git_status(repo_path)` để hiển thị trạng thái repo hiện tại (giống lệnh `git status`), `git_commit(repo_path, message)` phục vụ việc lưu lại các thay đổi vào kho Git, `git_add(repo_path, files)` hỗ trợ việc đưa nội dung các tệp vào khu vực staging, và `git_checkout(repo_path, branch_name)` để chuyển sang một branch khác. Tuy nhiên, MCP Server này không cung cấp Resource hay Prompts,
- [Time MCP Server](#): lấy thông tin thời gian hiện tại và chuyển đổi múi giờ thông qua hai tool sau: `get_current_time(timezone)` để lấy thời gian hiện tại tại một múi giờ cụ thể hoặc tại múi giờ hệ thống, và `convert_time(source_timezone, time, target_timezone)` để chuyển đổi thời gian từ một múi giờ này sang múi giờ khác,
- [FileSystem MCP Server](#): hỗ trợ thao tác hệ thống tệp như đọc/ghi file và liệt kê thư mục với các tool: `read_file(path)` để đọc nội dung tệp, `write_file(path, content)` để ghi nội dung vào tệp, và `list_directory(path)` để liệt kê các mục trong thư mục. Tất cả thao tác đều bị giới hạn trong các thư mục được cấu hình trước.
- [PostgreSQL MCP Server](#): cung cấp quyền truy cập chỉ đọc vào cơ sở dữ liệu PostgreSQL và hỗ trợ tool `query(sql)` để thực thi các truy vấn SQL.

Tool trong MCP server hoạt động tương tự function calling, cho phép mô hình AI kích hoạt tính toán, điều chỉnh trạng thái, tích hợp với API bên ngoài hoặc thực hiện các hoạt động hệ thống nằm ngoài khả năng cơ bản của LLM. Điểm đặc trưng của tools trong MCP là được điều khiển bởi mô hình, nhưng chịu sự giám sát từ server và người dùng. Mô hình AI có thể tự động đề xuất việc sử dụng tool dựa trên sự hiểu biết của nó về nhiệm vụ đang xử lý (ví dụ như tìm file trong Google Drive sẽ dùng tool `search(query)` thông qua Google Drive MCP Server, đọc file từ local với tool `readFile(filename)`). Tuy nhiên, mọi yêu cầu sử dụng tool đều phải được thực thi bởi server và khi cần thiết (ví dụ như quyền truy cập vào Google Drive, quyền truy cập file, quyền thực thi command trong terminal) thì phải có sự đồng ý từ người

dùng. Quy trình này đảm bảo sự kết hợp giữa sức mạnh và tính kiểm soát: mô hình AI được trao quyền mở rộng năng lực tính toán và tương tác với môi trường, nhưng những hành động quan trọng hoặc nhạy cảm sẽ luôn được giám sát chặt chẽ bởi người dùng.

Các đặc trưng của Tool trong MCP server có thể được tóm gọn như sau:

- **Khám phá (Discovery):** MCP client có thể truy xuất danh sách các công cụ hiện có bằng cách gửi request đến endpoint `tools/list`.
- **Kích hoạt (Invocation):** Các công cụ được gọi thông qua yêu cầu `tools/call`, trong đó server sẽ thực hiện thao tác được yêu cầu và trả về kết quả.
- **Tính linh hoạt (Flexibility):** Các công cụ có thể rất đa dạng, từ các phép tính đơn giản cho đến những tương tác phức tạp với API.

**Luồng hoạt động của Tool** Quá trình gọi một Tool và lấy kết quả của mô hình AI với MCP như sau:

1. LLM lựa chọn một tool cùng với arguments để chạy tool đó (ví dụ: tool `get_weather` với tham số là `{"location": "San Francisco"}`).
2. MCP client đóng vai trò trung gian, có thể yêu cầu người dùng xác nhận hành động thực thi tool và thực hiện lệnh gọi tool đến MCP server.
3. MCP server thực thi tool đó và trả về output dưới dạng có cấu trúc (structured response).
4. MCP client chuyển kết quả trở lại cho mô hình AI, giúp quá trình suy luận và giải quyết bài toán tiếp tục một cách liền mạch.

**Định nghĩa Tool** Mỗi tool được định nghĩa theo cấu trúc sau:

#### Cấu trúc định nghĩa tool trong MCP server

```
{
 name: string ,
```

```
description?: string,
inputSchema: {
 type: "object",
 properties: { ... }
},
annotations?: {
 title?: string,
 readOnlyHint?: boolean,
 destructiveHint?: boolean,
 idempotentHint?: boolean,
 openWorldHint?: boolean
}
```

- **name:** Tên của tool và không được trùng lặp trong cùng MCP server
- **description (Optional):** Mô tả về tool
- **inputSchema:** JSON Schema định nghĩa các tham số đầu vào của tool
  - **type:** Luôn được set giá trị "object"
  - **properties:** Các tham số cụ thể của tool
- **annotations** (tùy chọn): Gợi ý về hành vi của tool
  - **title:** Tiêu đề dễ hiểu dành cho con người
  - **readOnlyHint:** Nếu giá trị là **true**, tool không thay đổi môi trường/dữ liệu
  - **destructiveHint:** Nếu giá trị là **true**, tool có thể thực hiện các thay đổi mang tính phá hủy
  - **idempotentHint:** Nếu giá trị là **true**, gọi lặp lại với cùng tham số sẽ không gây thêm hiệu ứng
  - **openWorldHint:** Nếu giá trị là **true**, tool tương tác với các object khác ở bên ngoài

Giả sử chúng ta có hàm `read_txt_file` để đọc một file .txt từ phía local storage:

### Hàm read\_txt\_file để đọc file .txt

```

1 def read_txt_file(file_path: str):
2 """
3 Read the contents of a text file.
4
5 Args:
6 file_path (str): Path to the text file.
7
8 Returns:
9 dict: Contents of the text file.
10 """
11 try:
12 with open(file_path, 'r') as file:
13 content = file.read()
14 return {"content": content}
15 except Exception as e:
16 return {"error": str(e)}

```

chúng ta định nghĩa tool `read_txt_file` trong MCP server tương ứng với hàm trên như sau:

```
{
 "name": "read_txt_file",
 "title": "File Reader",
 "description": "Read the contents of a text file",
 "inputSchema": {
 "type": "object",
 "properties": {"file_path": {"type": "string",
 "description": "Path to the text file"}},
 "required": ["file_path"],
 },
 "outputSchema": {
 "type": "object",
 "properties": {"content": {"type": "string",
 "description": "Contents of the text file"}},
 "required": ["content"],
 },
}
```

**Xây dựng Tools cho Library MCP Server** Để xây dựng tools cho Library MCP Server, chúng ta viết hai methods sau:

- `list_tools()` dùng để liệt kê các tools mà MCP server hỗ trợ
- `call_tool(name, arguments)` để MCP server thực thi tool khi MCP client gọi tool đó.

Chúng ta sẽ thêm 3 tools sau đây vào Library MCP Server:

- `add_book` để thêm sách vào thư viện
- `remove_book`: xóa một quyển sách khỏi thư viện
- `get_num_books`: lấy thông tin về số sách hiện có trong thư viện

Với hàm `list_tools`, ta sử dụng decorator `@server.list_tools()` và định nghĩa từng tool trong hàm như sau:

### Thiết kế hàm `list_tools`

```

1 class Book(BaseModel):
2 title: str = Field(..., description="The title of the book")
3 author: str = Field(..., description="The author of the book")
4 isbn: str = Field(..., description="The ISBN of the book")
5 tags: list[str] = Field(default_factory=list, description="Tags
6 associated with the book")
7
7 class BookISBNInput(BaseModel):
8 isbn: str = Field(..., description="The ISBN of the book to be
9 removed or retrieved")
10
10 class AddBookInput(Book):
11 pass
12
13 ...
14
15 @server.list_tools()
16 async def list_tools() -> list[Tool]:
17 return [
18 Tool(
19 name="add_book",
20 description="Add a book to the library",

```

```

21 inputSchema={
22 "type": "object",
23 "required": ["title", "author", "isbn"],
24 "properties": {
25 "title": {
26 "description": "The title of the book",
27 "type": "string",
28 },
29 "author": {
30 "description": "The author of the book",
31 "type": "string",
32 },
33 "isbn": {
34 "description": "The ISBN of the book",
35 "type": "string",
36 },
37 "tags": {
38 "description": "Tags associated with the
39 book",
40 "items": {"type": "string"},
41 "type": "array",
42 },
43 },
44),
45 Tool(
46 name="remove_book",
47 description="Remove a book by its ISBN",
48 inputSchema=BookISBNInput.model_json_schema(),
49),
50 Tool(
51 name="get_num_books",
52 description="Get the total number of books",
53 inputSchema={"type": "object", "properties": {}},
54),
55]

```

Trong đó, các tool được định nghĩa bằng class Tool với arguments `name`, `description`, và `inputSchema`. Đối với tool `add_book`, input shema là thông tin một quyển sách (gồm tên, author, ISBN, và tags) được viết dưới dạng JSON object. Các field của JSON object này được định nghĩa giống như cách định nghĩa field ở MCP server đã đề cập ở phần [64.2.4](#). Đối với tool

`remove_book`, ta có input schema là data class `BookISBNInput`. Với tool `get_num_books`, ta không có input arguments nên input schema rỗng.

Chúng ta tiếp tục implement hàm `call_tool` với decorator `@server.call_tool()` như sau:

### Thiết kế hàm call\_tool

```

1 @server.call_tool()
2 async def call_tool(
3 name: str, arguments: dict
4) -> Sequence[TextContent | ImageContent | EmbeddedResource]:
5 try:
6 match name:
7 case "add_book":
8 book = AddBookInput(**arguments)
9 result = library.add_book(book)
10 return [TextContent(type="text", text=result)]
11
12 case "remove_book":
13 data = BookISBNInput(**arguments)
14 result = library.remove_book(data.isbn)
15 return [TextContent(type="text", text=result)]
16
17 case "get_num_books":
18 result = library.get_num_books()
19 return [TextContent(type="text", text=str(result))]
20
21 case _:
22 raise ValueError(f"Unknown tool: {name}")
23
24 except Exception as e:
25 raise ValueError(f"LibraryServer Error: {str(e)}")

```

Chúng ta có thể thấy rằng hàm `call_tool` đóng vai trò như một cầu nối giữa các lệnh được gọi từ phía MCP client với các hàm của hệ thống quản lý thư viện. Cụ thể như sau:

- Sau khi match được tên tool, ta giải mã input arguments về dạng schema đã định nghĩa (`AddBookInput`, `BookISBNInput`, v.v.), sau đó truyền các giá trị này vào hàm tương ứng của class `LibraryManagement`.
- Kết quả trả về được chuẩn hoá dưới dạng một danh sách các object

**TextContent.** Đây là cấu trúc được định nghĩa để trả lại phản hồi dưới dạng văn bản.

## Resources

**Resources** trong MCP server là các tài nguyên dữ liệu chỉ đọc mà server cung cấp cho client, ví dụ như các file code, nhật ký (logs), bản ghi cơ sở dữ liệu, hình ảnh, video, audio, hoặc phản hồi từ API. Đối với các mô hình AI, resources đóng vai trò như ngữ cảnh bổ sung kiến thức. Chúng là những thông tin hữu ích có thể được truy xuất từ MCP server và thêm vào prompt trong quá trình giao tiếp với mô hình AI, nhưng resource không bao giờ bị chỉnh sửa/thay đổi.

Không giống như tool hay function call sẽ thực thi các đoạn mã và có thể tạo ra tác động đến server, resources được thiết kế hoàn toàn để phục vụ việc truy xuất thông tin, giúp chúng trở nên an toàn và dễ dự đoán kết quả nào được trả về trong quá trình tìm kiếm. Resource đặc biệt phù hợp trong những tình huống mà mô hình cần hiểu ngữ cảnh hơn là thực thi một hành động cụ thể. Ví dụ, khi người dùng nói: "Hiển thị số liệu doanh thu mới nhất," client phía Host có thể truy xuất một tệp CSV hoặc bản chụp dữ liệu từ cơ sở dữ liệu thông qua một Resource và chèn nó vào đầu prompt, nhưng sẽ không thực thi bất kỳ hành động nào để tránh việc tác động vào dữ liệu ở server.

## Key Features

- Kiểm soát bởi ứng dụng:** MCP client có toàn quyền quyết định khi nào và bằng cách nào để truy xuất resources. Một MCP client có thể cho phép người dùng lựa chọn resource từ danh sách, trong khi một MCP client khác có thể tự động chọn resources dựa trên các keywords trong messages của người dùng.
- Đặt tên linh hoạt và dễ dàng truy xuất:** Các resources được định danh thông qua URI với các giao thức tùy chỉnh (custom protocol) như `file://`, `mysql://`, `screen://`, v.v.. Client có thể truy xuất danh sách resource thông qua endpoint `resources/list` và sử dụng các mẫu URI (URI templates) để thêm các arguments (ví dụ: `log://{{date}}.log` có argument là `date`, phục vụ cho việc truy xuất log theo ngày).

3. **Đa dạng loại nội dung:** Resources của MCP server có hai dạng, là dạng văn bản (mã hóa UTF-8) như source code, file config, file logs, dữ liệu JSON/XML, text thuần túy, hoặc dữ liệu dưới dạng mã nhị phân (được mã hóa dưới dạng base64) như hình ảnh, âm thanh, video, PDFs, v.v..
4. **Cập nhật theo thời gian thực:** Client có thể đăng ký nhận thông báo khi danh sách resources bị thay đổi thông qua endpoint `notifications/resources/list-changed`, hoặc là nội dung của một resource cụ thể được cập nhật (thông qua endpoint `resources/subscribe` và endpoint `notifications/resources/update`)

**Resource URIs** Resources được định danh bằng các URI theo định dạng sau:

[protocol]://[host]/[path]

trong đó, protocol và path được xác định bởi cách triển khai của MCP . Các server có thể tự định nghĩa các scheme URI tùy chỉnh của riêng mình. Ta có các ví dụ URI của Resource như sau:

- csv://datasets/sales/2025.csv
- mysql://database/customers/schema
- log://datetime.log

và các resource được lấy ở các MCP server thực tế như sau:

- [Google Drive MCP Server](#): resource `gdrive:///<file_id>` hỗ trợ client truy cập file ở Google Drive với `file_id`.
- [SQLite MCP Server](#): cung cấp resource `memo://insights` như một bản ghi chú tổng hợp thông tin chi tiết kinh doanh được cập nhật liên tục trong quá trình phân tích của server.
- [n8n MCP Server](#): là một MCP server cho phép mô hình AI tương tác trực tiếp với hệ thống workflow của n8n. n8n MCP Server cung cấp các resource sau: `n8n://workflows/list` để liệt kê toàn bộ workflows, `n8n://workflow/{id}` để truy vấn thông tin chi tiết của một workflow

cụ thể, `n8n://executions/{workflowId}` để lấy danh sách các lần thực thi của một workflow, và `n8n://execution/{id}` để xem chi tiết một lần thực thi.

- **Everything MCP server:** cung cấp 100 tài nguyên mẫu để kiểm thử khả năng xử lý resource trong MCP client. Các resource có URI theo dạng `test://static/resource/{id}`.

**Định nghĩa Resource trong MCP Server** Resource trong MCP server được định nghĩa như sau:

#### Định nghĩa resource trong MCP server

```
{
 uri: string,
 name: string,
 description?: string,
 mimeType?: string,
 size?: number
}
```

trong đó,

- `uri`: Unique name (định danh) duy nhất của resource
- `name`: Tên dễ hiểu
- `description` (tùy chọn): Mô tả bổ sung
- `mimeType` (tùy chọn): Loại MIME (kiểu nội dung) của resource
- `size` (tùy chọn): Kích thước resource tính bằng byte

Ví dụ như với file PDF có url <https://docs.google.com/document/d/1ve33Kex-cJKAKBToy—lddE6BLw-LA6bBPtbTw4F8o/edit> trên Google Drive, ta có định nghĩa của file ở Google Drive MCP server như sau:

```
{
 "uri":
 "gdrive:///1ve33Kex-cJKAKBToy---lddE6BLw-LA6bBPtbTw4F8o",
```

```

 "name": "Example Resource",
 "description": "This is an example resource for
 → demonstration purposes.",
 "mimeType": "application/pdf",
 "size": 4096,
}

```

**Xây dựng Resource cho Library MCP Server** Đối với Library MCP Server của chúng ta, chúng ta sẽ xây dựng cả static resource (resource tĩnh) và dynamic resource (resource động).

Đối với static resource, chúng ta viết hàm `list_resources()` và dùng decorator `@server.list_resources()` như sau:

#### Xây dựng hàm `list_resources()`

```

1 async def list_resources() -> list[Resource]:
2 return [
3 Resource(
4 name="all_books",
5 title="All Books",
6 uri=AnyUrl("books://all"),
7 description="Get all books in the library",
8),
9]

```

Đây là nơi chúng ta định nghĩa các resource tĩnh với các fields như `name`, `title`, `uri`, và `description`. Resource `all_books` ("books://all") được định nghĩa để MCP client có thể lấy dữ liệu tất cả các quyển sách trong MCP .

Đối với dynamic resource, chúng ta implement hàm `list_resource_templates()` với decorator `@server.list_resource_templates()`:

#### Xây dựng hàm `list_resource_templates()`

```

1 @server.list_resource_templates()
2 async def list_resource_templates() -> list[ResourceTemplate]:
3 return [
4 ResourceTemplate(

```

```

5 name="book_by_index",
6 title="Book by Index",
7 uriTemplate="books://index/{index}",
8 description="Get a book by its index in the library",
9),
10 ResourceTemplate(
11 name="book_by_isbn",
12 title="Book by ISBN",
13 uriTemplate="books://isbn/{isbn}",
14 description="Get a book by its ISBN",
15),
16]

```

trong đó, `index` là argument của URI `books://index/index` để lấy thông tin quyển sách theo index trong database, và URI `books://isbn/isbn` với argument `isbn` để tìm sách theo mã ISBN. URI của dynamic resource được định nghĩa bằng field `uriTemplate` thay vì field `uri` như static resource.

Cuối cùng, chúng ta viết hàm `read_resource(uri)` với decorator `@server.read_resource()` để trả về resource dựa trên URI tương ứng:

### Xây dựng hàm `read_resource(uri)`

```

1 @server.read_resource()
2 async def read_resource(uri: AnyUrl) -> str:
3 uri_str = str(uri)
4 if uri_str == "books://all":
5 books = library.get_all_books()
6 return json.dumps(books, indent=4)
7 elif uri_str.startswith("books://index/"):
8 index_str = uri_str.split("/")[-1]
9 try:
10 index = int(index_str)
11 book = library.get_book_by_index(index)
12 if "error" in book:
13 raise ValueError(book["error"])
14 return json.dumps(book, indent=4)
15 except ValueError:
16 raise ValueError(f"Invalid index: {index_str}")
17 elif uri_str.startswith("books://isbn/"):
18 isbn = uri_str.split("/")[-1]
19 book = library.get_book_by_isbn(isbn)

```

```

20 if "error" in book:
21 raise ValueError(book["error"])
22 return json.dumps(book, indent=4)
23 else:
24 raise ValueError(f"Resource '{uri}' not found.")

```

Đây là phần xử lý thực thi khi có yêu cầu truy xuất một resource cụ thể, dựa trên URI truyền vào. Hàm `read_resource` sẽ phân tích URI và gọi hàm tương ứng từ class `LibraryManagement` để lấy resource được định nghĩa ở hai hàm liệt kê trên, sau đó tuân tự hóa thành định dạng JSON để trả về cho MCP client.

## Prompts

**Prompts trong MCP Server** là các prompt template được định nghĩa sẵn hoặc mẫu cuộc trò chuyện mà MCP server cung cấp cho MCP client. Đây thực chất là các đoạn hội thoại hoặc các query template được lưu trữ trong hệ thống mà chúng ta hoặc các hệ thống tự động hoặc hệ thống tự động có thể gọi ra dùng để đơn giản hóa quá trình tương tác với mô hình AI.

Ví dụ, một server có thể cung cấp một prompt tên là `find_bug` (xem code bên dưới), chứa các lệnh hướng dẫn mô hình AI đọc và tìm bug dựa trên error message và các chỗ trống để chèn nội dung động.

```

1 @mcp.prompt()
2 def code_debug(code: str, error_message: str) -> List[Dict[str, str]]:
3 return [
4 {"role": "system", "content": f"You are a helpful assistant that
5 helps debug code."},
6 {
7 "role": "user",
8 "content": f"Here is the code that has an error: {code}\nError
9 message: {error_message}",
10 },
11]

```

Các prompt này sẽ hiển thị trong giao diện người dùng hoặc dropdown menu, cho phép người dùng lựa chọn, tùy chỉnh và chèn vào luồng hội thoại một cách linh hoạt.

**Lợi ích của prompt khi đặt ở MCP Server** Không giống như tools, vốn được LLM tự động gọi để thực hiện hành động, prompts là những chỉ dẫn do người dùng hoặc lập trình viên chủ động chọn lựa nhằm định hình hành vi của mô hình. Chúng bao gồm các template được thiết kế để hướng dẫn mô hình AI hoạt động - ví dụ: "Bạn là một developer chuyên debug các đoạn code lỗi ... và có thể được áp dụng trên nhiều MCP với quy mô lớn. Việc MCP server công khai các prompt mang lại các lợi ích như sau:

- Tính nhất quán: Cung cấp định dạng chuẩn cho các tác vụ lặp lại như tóm tắt, kiểm tra mã, hay gỡ lỗi.
- Khả năng tái sử dụng: Client có thể gọi server (prompts/list, prompts/get) để lấy các mẫu có cấu trúc và tích hợp vào quy trình làm việc của người dùng.
- Tính mở rộng: Prompt có thể bao gồm arguments, trả đến resources, hoặc nhiều messages - hỗ trợ nội dung động và multi-turn conversations,
- Việc tách biệt template ra khỏi client cho phép duy trì, chia sẻ và cập nhật các prompt tốt nhất tại MCP server mà không cần thay đổi code ở phía client.

Chúng ta có nêu vài prompts trong các MCP server thực tế như sau:

- [Everything MCP Server](#): để test MCP client, MCP Server này cung cấp các prompts như `simple_prompt` và prompt cơ bản, `complex_prompt(temperature, style?)` để mô phỏng cuộc đối thoại nhiều lượt với hai tham số đầu vào, và `resource_prompt(resourceId)` là prompt minh họa cho việc nhúng resource vào nội dung prompt.
- [Fetch MCP server](#): hỗ trợ prompt `fetch(url)` để truy xuất nội dung từ trang web và chuyển đổi HTML thành markdown. Thích hợp cho các tác vụ đọc hiểu web, với khả năng phân trang nội dung thông qua `start_index`.
- [Sentry MCP server](#): cung cấp prompt `sentry-issue(issue_id_or_url)` để truy xuất chi tiết lỗi từ Sentry.io dưới dạng hội thoại. Kết quả bao gồm tiêu đề lỗi, mức độ nghiêm trọng, thời gian xuất hiện, số lượng sự kiện và toàn bộ stacktrace.

- **SQLite MCP server:** cung cấp prompt `mcp-demo(topic)` để hướng dẫn người dùng thực hiện phân tích dữ liệu kinh doanh. Prompt này tạo schema và dữ liệu mẫu phù hợp, đồng thời tương tác với resource `memo://insights` (bản ghi động cập nhật liên tục các kết luận phân tích).
- **Deep Research MCP server:** cung cấp prompt `deep-research` dành cho các nhiệm vụ nghiên cứu chuyên sâu., hỗ trợ toàn bộ quy trình nghiên cứu từ mở rộng câu hỏi, tạo các tiêu đề tài, tìm kiếm nguồn đáng tin cậy, phân tích nội dung cho đến tổng hợp thành báo cáo đầy đủ, có trích dẫn và định dạng chuyên nghiệp.

**Định nghĩa Prompt trong MCP Server** Prompt trong MCP server được định nghĩa như sau:

#### Định nghĩa prompt trong MCP server

```
{
 name: string,
 description?: string,
 arguments?: [
 {
 name: string,
 description?: string,
 required?: boolean
 }
]
}
```

- **name:** Tên duy nhất của prompt (không được trùng)
- **description** (tùy chọn): Mô tả dễ hiểu về prompt dành cho con người
- **arguments** (tùy chọn): Danh sách các arguments mà prompt có thể chấp nhận
  - **name:** Tên của argument
  - **description** (tùy chọn): Mô tả về argument

- **required** (tùy chọn): Xác định liệu argument có bắt buộc hay không

Ví dụ như đối với prompt `code_debug` ở trên, ta có định nghĩa về prompt đó trong MCP server như sau:

```
{
 "name": "code_debug",
 "description": "A tool to help debug code by
 → providing the code and the error message.",
 "arguments": [
 {
 "name": "code",
 "description": "The code that has an
 → error.",
 "required": True,
 },
 {
 "name": "error_message",
 "description": "The error message that was
 → raised.",
 "required": True,
 },
],
}
```

**Xây dựng Prompts cho Library MCP Server** Chúng ta sẽ viết các prompts cho Library MCP Server với `@app.prompt()` decorator. Decorator `@prompt` có các arguments sau:

- **name**: Tên tùy chọn cho prompt (mặc định là tên của hàm)
- **title**: Tiêu đề hiển thị dễ hiểu dành cho con người (tùy chọn)
- **description**: Mô tả tùy chọn về chức năng hoặc mục đích của prompt. Nếu field này bị bỏ trống, phần docstring của hàm sẽ được sử dụng thay thế.

Tiếp theo, chúng ta implement các loại prompts sau:

- Static prompt: prompt không nhận bất kì tham số/biến nào. Trong MCP server của chúng ta, đó là prompt `suggest_random_book`.
  - Dynamic prompt: prompt điều chỉnh theo arguments mà MCP client cung cấp. Bao gồm các prompts sau:
    - `suggest_book_title_by_abstract(abstract)`: nhận vào argument `abstract` để gợi ý tên sách dựa trên abstract.
    - `analyze_book(book, query)`: nhận vào hai arguments là `book` và `query` để phân tích sách dựa trên nội dung và câu hỏi của người dùng.

Tương tự như tools và resources, chúng ta cần implement hai hàm `list_prompts()` và `get_prompt(name, arguments)` để trả về danh sách các prompts và lấy một prompt cụ thể dựa trên tên prompt cần lấy cùng với các arguments tương ứng. Các hàm này sẽ trả về các đối tượng `Prompt` và `GetPromptResult` tương ứng với định nghĩa của MCP.

Đối với hàm `list_prompts()`, chúng ta dùng decorator `@server.list_prompts()` và implement như sau:

```
1 @server.list_prompts()
2 async def list_prompts() -> list[Prompt]:
3 return [
4 Prompt(
5 name="suggest_random_book",
6 description="Suggest a random book from the library. The
7 suggestion should include the title,
8 author, and a brief description.",
9),
10 Prompt(
11 name="suggest_book_title_by_abstract",
12 description="Suggest a memorable, descriptive title for a book
13 based on the following abstract.",
14 arguments=[
15 PromptArgument(
16 name="abstract",
17 description="The abstract of the book.",
18 required=True,
19)
20],
21),
22 Prompt(
23 name="suggest_chapter_summary",
24 description="Provide a brief summary of a specific chapter or section
25 from a given book."),
26]
```

```

20 name="analyze_book",
21 description="Analyze a book based on its content and user
22 query.",
23 arguments=[
24 PromptArgument(name="book", description="The book to
25 analyze.", required=True),
26 PromptArgument(
27 name="query",
28 description="The query for analysis.",
29 required=True,
30),
31],
32]

```

Chúng ta sẽ định nghĩa các prompts bao gồm các fields `name` là tên prompt, `description` chứa miêu tả về prompt cũng như cách sử dụng prompt, và `arguments` là các tham số của prompt (nếu có). Các arguments sẽ là một danh sách các object thuộc class `PromptArgument` với các fields `name` là tên của argument, `description` là miêu tả về argument, và `required` để xác định liệu argument đó có bắt buộc hay không.

Đối với hàm `get_prompt(name, arguments)` để trả về một prompt cụ thể, chúng ta dùng decorator `@server.get_prompt()` và implement như sau:

```

1 @server.get_prompt()
2 async def get_prompt(name: str, arguments: Dict[str, str] | None) ->
3 GetPromptResult:
4 prompts = await list_prompts()
5 for prompt in prompts:
6 if prompt.name == name:
7 if arguments is None:
8 arguments = {}
9 if prompt.arguments:
10 for arg in prompt.arguments:
11 if arg.name not in arguments and arg.required:
12 raise ValueError(f"Missing required argument: {arg
13 .name}")
14 break
15 else:
16 raise ValueError(f"Prompt '{name}' not found.")
17
18 if name == "suggest_random_book":
19 prompt_result = library.get_suggesting_random_book_prompt()
20 return GetPromptResult(

```

```

19 description=prompt.description,
20 messages=[
21 PromptMessage(
22 role="user",
23 content=TextContent(type="text", text=prompt_result),
24)
25],
26)
27 elif name == "suggest_book_title_by_abstract":
28 prompt_result = library.
29 get_suggesting_book_title_by_abstract_prompt
30 (**arguments)
31 return GetPromptResult(
32 description=prompt.description,
33 messages=[
34 PromptMessage(
35 role="user",
36 content=TextContent(type="text", text=prompt_result),
37)
38],
39)
40 elif name == "analyze_book":
41 book, query = arguments["book"], arguments["query"]
42 messages = library.get_analyzing_book_messages(book, query)
43 return GetPromptResult(
44 description=prompt.description,
45 messages=[
46 PromptMessage(
47 role=m["role"],
48 content=TextContent(type="text", text=m["content"]),
49)
50 for m in messages
51],
52)
53 else:
54 raise ValueError(f"Prompt '{name}' is not implemented.")

```

Hàm `get_prompt` sẽ nhận vào tên của prompt (`name`) và các arguments tương ứng (`arguments`). Hàm sẽ tìm kiếm trong danh sách các prompts đã được định nghĩa, kiểm tra xem tên prompt có tồn tại hay không, và nếu có thì kiểm tra các arguments có hợp lệ hay không. Nếu hợp lệ, hàm sẽ trả về một đối tượng `GetPromptResult` chứa prompt hoặc các messages để người dùng có thể sử dụng trong cuộc hội thoại với mô hình AI.

### Tóm tắt các thành phần cơ bản của MCP Server

Thông tin tóm tắt về từng primitive được tóm tắt trong [Bảng 64.1](#).

Primitive	Điều khiển bởi	Mô tả	Ví dụ
Prompts	Người dùng kiểm soát	Mẫu tương tác được kích hoạt theo lựa chọn của người dùng	Lệnh gạch chéo, tùy chọn trong menu
Resources	Ứng dụng kiểm soát	Dữ liệu ngữ cảnh được đính kèm và quản lý bởi client	Nội dung file PDF, lịch sử git
Tools	Mô hình kiểm soát	Các hàm được mở ra cho LLM để thực hiện hành động	POST request đến external API, ghi file vào local

Bảng 64.1: Cấu trúc phân cấp điều khiển của các primitive

#### 64.2.5 Các cơ Chế Truyền Tải trong MCP

Cơ chế truyền tải (transport) trong MCP xác định cách thức mà MCP client và MCP server trao đổi tin tức/thông điệp (các JSON-RPC messages) với nhau. Các cơ chế truyền tải này xử lý cách đóng gói, truyền tải và nhận thông điệp - đồng thời đảm bảo kết nối đáng tin cậy và dễ tương tác giữa các hệ thống. Hiện tại, MCP hỗ trợ hai cơ chế truyền tải mặc định: stdio và Streamable HTTP. Ngoài ra, giao thức còn cho phép developer cài đặt các cơ chế truyền tải tùy chỉnh, mang lại sự linh hoạt trong việc tích hợp với các môi trường khác nhau.

##### Standard Input/Output (stdio) Transport

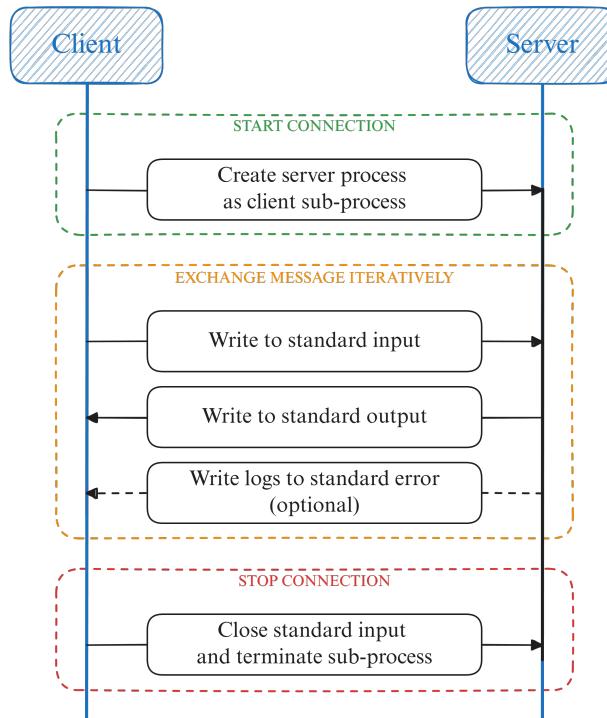
Transport stdio cho phép giao tiếp thông qua các luồng input và output theo tiêu chuẩn. Chúng ta lựa chọn stdio transport cho các môi trường cục bộ như terminal, shell scripts, hoặc bất kỳ kịch tình huống mà MCP client và MCP server hoạt động trên cùng một máy.

Cơ chế hoạt động của stdio transport gồm những bước sau (minh họa trong [Hình 64.13](#)):

- Client khởi chạy MCP server dưới dạng một tiến trình con (sub-process) của client process.

2. Các thông điệp được MCP client truyền qua standard input của MCP server và gửi đi từ MCP server qua standard output của MCP server. Quá trình này được lặp đi lặp lại trong suốt chu kì kết nối.
3. MCP server có thể ghi thông tin lỗi vào stderr, và MCP client có thể lựa chọn sử dụng log đó hoặc bỏ qua.
4. Khi kết thúc, MCP client đóng standard input và dừng tiến trình con.

Transport này phù hợp nhất cho các tích hợp đơn giản trong môi trường cục bộ, với thiết lập tối giản và hiệu suất cao mà không cần kết nối internet giữa MCP client và MCP server.



Hình 64.13: Stdio Transport Sequence Diagram

### Streamable HTTP Transport

Transport Streamable HTTP được thiết kế cho các môi trường web hoặc nhiều client. Cơ chế này sử dụng HTTP POST để MCP client gửi thông điệp

JSON-RPC tới MCP server, và tùy chọn sử dụng Server-Sent Events (SSE) để MCP server truyền thông điệp ngược lại theo kiểu streaming. Những đặc điểm nổi bật của cơ chế truyền tải này bao gồm:

- **Session có chứa trạng thái (Stateful Session):** MCP client và MCP server có thể trao đổi ID phiên thông qua header `Mcp-Session-Id` tùy chỉnh nhằm duy trì ngữ cảnh xuyên suốt nhiều requests. Session bắt đầu khi khởi tạo và có thể được kết thúc rõ ràng qua method HTTP `DELETE`.
- **Streaming và khả năng tiếp tục khi bị gián đoạn:** MCP server có thể nhúng event ID trong phản hồi SSE để MCP client có thể tiếp tục các luồng bị gián đoạn bằng cách gửi `Last-Event-ID` trong các GET requests kế tiếp.
- **Xử lý thông điệp linh hoạt:** MCP client gửi JSON-RPC message qua POST request. MCP server có thể phản hồi bằng một JSON duy nhất hoặc khởi tạo luồng phản hồi liên tục thông qua SSE. Ngoài ra, MCP client cũng có thể mở các luồng SSE GET method để nhận các yêu cầu hoặc thông báo từ MCP server.
- **Yêu cầu về bảo mật:** Các quy tắc bao gồm kiểm tra header `origin`, giới hạn binding ở localhost cho triển khai cục bộ, bắt buộc sử dụng TLS trong môi trường production, và quản lý session ID một cách an toàn.

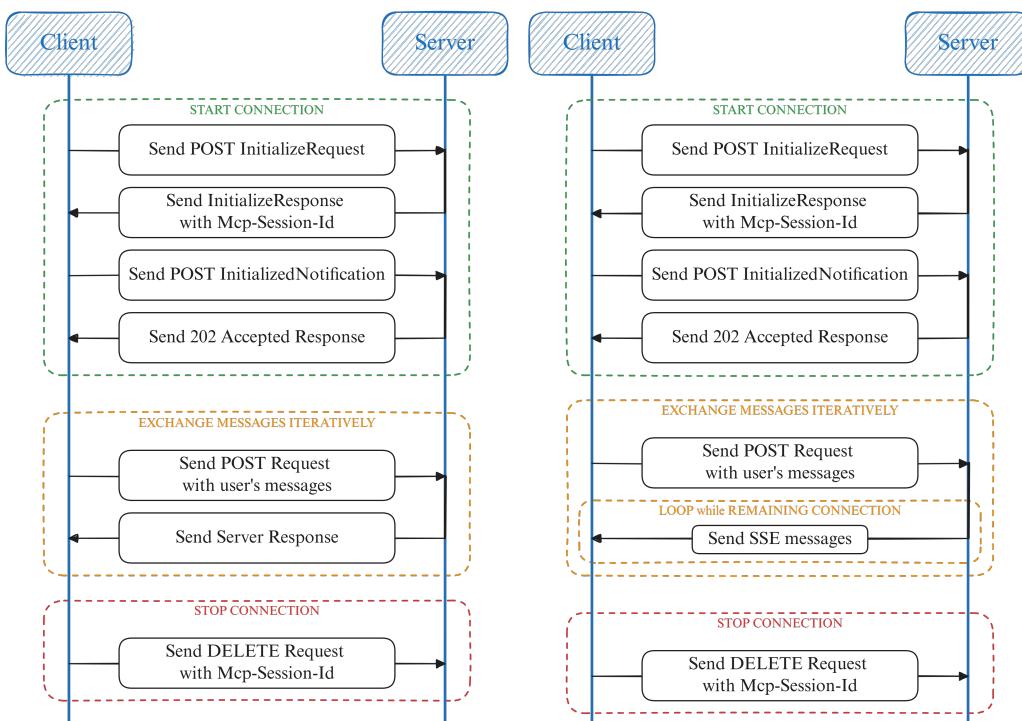
Streamable HTTP Transport hỗ trợ tốt kiến trúc client-server, cho phép nhiều client hoạt động đồng thời tại Host, có thể phục hồi kết nối bị gián đoạn, và khả năng truyền thông điệp đa dạng hơn stdio transport.

Chu trình hoạt động của Streamable HTTP transport được liệt kê như sau (xem Hình 64.14a):

1. Khi bắt đầu kết nối, MCP client gửi POST request đến server để bắt đầu kết nối. Nếu MCP server chấp nhận kết nối, MCP server sẽ gửi về `Mcp-Session-Id` để phục vụ mục đích theo dõi và hỗ trợ phiên kết nối (reconnect, close, v.v.).
2. Trong quá trình sử dụng, MCP client sẽ gửi POST request chứa nội dung của người dùng hoặc là tool cần chạy, resource cần lấy, v.v. và

MCP server sẽ xử lý nội dung và trả về trong một request tương tự. Đối với SSE transport (phiên bản cũ), response từ MCP server có thể được chia nhỏ thành nhiều messages và gửi đi lần lượt đến MCP client (xem Hình 64.14b).

3. Cuối cùng, để đóng kết nối, MCP client gửi DELETE request kèm theo Mcp-Session-Id trong header và server sẽ đóng kết nối với MCP client này.



(a) HTTP Transport Sequence Diagram (b) SSE Transport Sequence Diagram

Hình 64.14: Streamable HTTP Transport Sequence Diagram

## SSE Transport

Trước đây, MCP hỗ trợ server-sent events (SSE) độc lập kết hợp với HTTP POST từ phía client, nhưng cơ chế này đã bị ngừng hỗ trợ kể từ ngày 5 tháng 11 năm 2024. Transport Streamable HTTP vẫn giữ khả năng hỗ trợ

SSE nhưng theo một kiến trúc tiêu chuẩn và đầy đủ tính năng hơn. Các hệ thống MCP client và MCP server vẫn có thể duy trì tính tương thích với các phiên bản thư viện MCP mới thông qua các cơ chế hỗ trợ ngược (backward-compatibility)

### Custom Transports

Bên cạnh hai cơ chế truyền tải stdio và Streamable HTTP, MCP không phụ thuộc vào transport, từ đó cho developer triển khai các lớp transport tùy biến theo nhu cầu. Điều này tạo điều kiện tích hợp trong các môi trường chuyên biệt - như message queues, framework điều khiển từ xa (remote procedure call frameworks - RPC), hoặc các kênh trên thiết bị di động. Các transport đều có thể được sử dụng với MCP, miễn là transport đó tuân thủ định dạng truyền tải thông điệp JSON-RPC của MCP, bao gồm quy tắc đóng gói thông điệp và đảm bảo vòng đời kết nối.

### Implement MCP Client with Different Transport Mechanisms

Trong phần này, chúng ta sẽ implement các cơ chế truyền tải cho Library MCP Server vừa xây dựng với hai loại cơ chế Standard Input/Output và Streamable HTTP.

**Standard Input/Output (stdio) Transport** Trong đoạn code [64.2.4](#), Library MCP Server đang sử dụng stdio transport (định nghĩa ở dòng 74-75) như sau:

#### Library MCP Server với stdio transport

```
1 options = server.create_initialization_options()
2 async with stdio_server() as (read_stream, write_stream):
3 await server.run(read_stream, write_stream, options)
```

Trong đó, các câu lệnh được giải thích như sau:

- `server.create_initialization_options()`: tạo ra một tập hợp các tùy chọn khởi tạo ban đầu cho MCP server. Các tùy chọn này có thể bao gồm thông tin về tool, resource, prompt, hoặc metadata,

- `async with stdio_server() as (read_stream, write_stream):`: Thiết lập một cơ chế truyền tải thông qua standard input/output với hai luồng `read_stream` để đọc các thông điệp từ MCP client gửi đến (ví dụ: các yêu cầu JSON-RPC) và `write_stream` để gửi thông điệp đến MCP client. Bên cạnh đó, thời nhở vào nhở vào cú pháp `async` của Python context manager, hệ thống tự động xử lý việc mở và đóng kết nối I/O một cách an toàn,
- `await server.run(read_stream, write_stream, options)` kích hoạt vòng lặp xử lý chính của MCP server. Keyword `await` đảm bảo coroutine này được thực thi bất đồng bộ, không chặn main process và không kết thúc cho đến khi MCP server bị tắt. Coroutine này đọc yêu cầu từ `read_stream`, xử lý thông điệp dựa trên cấu hình trong `options`, và gửi phản hồi qua `write_stream`.

Ở phía MCP client, chúng ta thiết lập stdio transport cho MCP client để mở một session như sau:

### Client dùng stdio transport

```

1 async def test_mcp_server_with_stdio_transport():
2 async with stdio_client(server_params) as (read, write):
3 async with ClientSession(read, write) as session:
4 # Initialize the connection
5 await session.initialize()
6 ...

```

Trong đó, các câu lệnh

- `async with stdio_client(server_params) as (read, write):`: Khởi chạy một process phụ đại diện cho MCP server, thiết lập luồng đọc/ghi dựa trên các luồng standard input và standard output, đóng vai trò như kênh giao tiếp chính,
- `async with ClientSession(read, write) as session:` bao bọc các luồng transport trong một lớp giao thức MCP ở tầng cao hơn, đồng thời cung cấp các method như `initialize()`, `list_tools()`, `call_tool()`, v.v.,

- `await session.initialize()` hoàn tất quá trình handshake với MCP server, bao gồm trao đổi metadata, các tool, prompt và resource hiện có của server, đồng thời chuẩn bị phiên làm việc để thực thi các lệnh liên quan đến tool và resource thông qua kênh kết nối đã thiết lập.

Sau khi đã khởi tạo kết nối thành công đến MCP server với lệnh `await session.initialize()`, chúng ta có thể khám phá năng lực của server thông qua các lệnh như `await session.list_tools()` và `await session.list_resources()`, đọc resource `await session.read_resource("resource://...")`, hay gọi tool để thực thi một task nào đó `result = await session.call_tool("tool_name", args=...)`

**Streamable HTTP Transport** Đối với cơ chế này, chúng ta thiết lập MCP server như sau:

### Cài đặt Streamable HTTP MCP Server

```

1 session_manager = StreamableHTTPSessionManager(
2 app=server,
3 event_store=None,
4 json_response=(
5 True if transport == "sse" else False
6), # Use JSON response for SSE, otherwise standard HTTP
7 stateless=True,
8)
9
10 # Handle HTTP requests with the session manager
11 async def handle_streamable_http(scope: Scope, receive: Receive,
12 send: Send) -> None:
13 await session_manager.handle_request(scope, receive, send)
14
15 @contextlib.asynccontextmanager
16 async def lifespan(app: Starlette) -> AsyncIterator[None]:
17 async with session_manager.run():
18 logger.info("Application started with StreamableHTTP session
19 manager!")
20 try:
21 yield
22 finally:
23 logger.info("Application shutting down...")
24
25 # Create an ASGI application using the transport

```

```

24 starlette_app = Starlette(
25 debug=True,
26 routes=[
27 Mount("/mcp", app=handle_streamable_http),
28],
29 lifespan=lifespan,
30)
31
32 uvicorn.run(starlette_app, host="127.0.0.1", port=port, log_level=
33 log_level.lower())

```

Đoạn code này được giải thích như sau:

1. Khởi tạo Session Manager với class `StreamableHTTPSessionManager` và nhận vào các arguments bao gồm:
  - `app=server`: MCP server chúng ta đã gắn các tools, resources và promtps,
  - `event_store=None`: tắt tính năng lưu event để hỗ trợ kết nối lại (resume). MCP client không thể tiếp tục session nếu bị disconnect,
  - `json_response`: Khi `transport == "http"`, trả về JSON response, và khi `transport == "sse"` thì trả về theo dạng streaming,
  - `stateless=True`: Mỗi request HTTP POST là một session riêng biệt, không ràng buộc về trạng thái hay session qua các kết nối.
2. Tạo handler xử lý request đến ASGI<sup>1</sup> bằng function `handle_streamable_http(scope, receive, send)`: Function này được mount vào ASGI server (Starlette/uvicorn). Mọi request đến /mcp sẽ được chuyển qua `handle_request()`, nơi session manager sẽ parse, tạo session tạm thời mới, gọi tool-method và trả kết quả,

<sup>1</sup>ASGI (Asynchronous Server Gateway Interface) là phiên bản hiện đại hơn của WSGI (Web Server Gateway Interface). Trong khi WSGI chỉ xử lý một yêu cầu web tại một thời điểm và phải chờ hoàn tất trước khi xử lý tiếp, thì ASGI có khả năng xử lý nhiều yêu cầu đồng thời mà không cần chờ đợi - giống như việc chúng ta có thể vừa trò chuyện vừa lướt web một cách mượt mà trong cùng một phiên làm việc. Framework Flask chỉ hỗ trợ WSGI, trong khi các Python web framework mới nhất hỗ trợ ASGI như FastAPI, Quart, Django (>=4), v.v.

3. Quản lý lifecycle của ứng dụng với hàm `lifespan(app)`: Trong hàm này, câu lệnh `session_manager.run()` mở một task group nội bộ để quản lý nhiều kết nối song song. Khi ứng dụng được khởi động, Starlette gọi hàm `lifespan(app)`, nhảy vào context tạo bởi `session_manager.run()` và session manager sẵn sàng nhận nhiều request cùng lúc. Khi shutdown, hệ thống thoát khỏi context và thông báo dừng server,
4. Cấu hình ứng dụng Starlette với class `Starlette`: Khởi tạo ứng dụng ASGI và gắn server của chúng ta vào route `/mcp` thông qua mount route `/mcp` và handler `handle_streamable_http`,
5. Khởi chạy server với Uvicorn: Bắt đầu chạy server ASGI của chúng ta ở local với method `uvicorn.run` và các tham số như `host`, `port`, v.v.

Về phía MCP client, chúng ta cài đặt để tạo kết nối tới MCP server bằng Streamable HTTP transport như sau:

### Streamable HTTP MCP Client

```
1 async with streamablehttp_client(server_url) as (read, write,
2 get_session_id_callback):
3 async with ClientSession(read, write) as session:
4 await session.initialize()
5 ...
```

Khác với MCP client dùng stdio transport ở [64.2.5](#), câu lệnh `async with streamable_http_client(server_url) as (read, write, get_session_id_callback)`: tạo một kết nối HTTP đến MCP server tại `server_url`. Ngoài ra, `streamablehttp_client` sử dụng phương thức truyền tải hoạt động thông qua HTTP bằng gửi các thông điệp JSON-RPC bằng phương thức POST và nhận phản hồi từ MCP server (kể cả các streaming events). Ngoài ra, hàm `streamablehttp_client` còn cung cấp một callback `get_session_id_callback`, cho phép MCP client truy xuất session ID nhằm hỗ trợ quá trình reconnect khi bị mất kết nối.

#### 64.2.6 Chạy Library MCP Server và thiết kế MCP Client để kiểm tra

Trong phần này, chúng ta sẽ tổng hợp lại toàn bộ đoạn code của Library MCP Server, MCP client và cách chạy các đoạn scripts.

##### Library MCP Server Full Implementation

Đầu tiên, ta tạo một folder rỗng và đặt tên liên quan đến project (vd: `library_mcp_server_and_client`). Sau đó, ta tạo file `server.py` với nội dung ở link [này](#).

Trong implementation của Library MCP Server, bên cạnh framework `mcp`, chúng ta dùng thêm thư viện `click` để thêm arguments trong quá trình chạy script trên terminal. Chúng ta có các arguments sau:

- `--log-level`: Cho phép người dùng chỉ định mức độ log của ứng dụng. Các giá trị hợp lệ là `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`. Mặc định là `INFO`. Argument này giúp kiểm soát thông tin in ra terminal hiển thị khi server đang chạy.
- `--transport`: Tùy chọn để chọn loại transport giữa MCP server và MCP client. Các giá trị hợp lệ là `http`, `stdio`, `sse`. Mặc định là `http`. Argument này quy định cách thức trao đổi dữ liệu giữa MCP client và MCP server.
- `--port`: Cho phép người dùng chỉ định cổng (port) mà HTTP MCP server sẽ triển khai trên đó. Giá trị port mặc định là 8000 và các giá trị người dùng đặt phải là một số nguyên. Argument này rất quan trọng khi triển khai server trên các môi trường khác nhau hoặc tránh xung đột cổng.

##### MCP Client Implementation

Trong folder vừa tạo, ta tạo tiếp một file `client.py` có nội dung đặt trong link [này](#).

File `client.py` có các thành phần chính sau:

- Các hàm hỗ trợ như `display_tools()`, `display_resources()`, `display_prompts()`, `display_resource_templates()` dùng để hiển thị các thành phần chính của MCP server (tools, resources, prompts).
- Hàm kiểm thử chính trong file `test_book_library_management_mcp_server(session)` thực hiện tuần tự các tác vụ kiểm thử cơ bản sau khi MCP client đã kết nối tới MCP server với các chức năng sau
  - Liệt kê các tool, resources, resource templates, prompts có sẵn (thể hiện rõ kiến trúc 3-primitives của MCP server đã mô tả ở các phần trên).
  - Thực hiện lần lượt các thao tác để tương tác với MCP server và hệ thống quản lí thư viện như sau:
    1. Gọi tool `get_num_books` để lấy số sách hiện có.
    2. Truy xuất toàn bộ resource `books://all` và in ra danh sách sách.
    3. Thêm một quyển sách mới thông qua tool `add_book` (tạo dữ liệu ngẫu nhiên để kiểm thử khả năng hoạt động thực tế).
    4. Kiểm tra số lượng sách sau khi thêm.
    5. Truy xuất quyển sách vừa thêm bằng index và bằng ISBN qua resource template.
    6. Xóa quyển sách vừa thêm bằng tool `remove_book`.
    7. Kiểm tra số lượng sách sau khi xóa, đảm bảo tính nhất quán dữ liệu.
    8. Lấy nội dung các prompt (bao gồm prompt static, dynamic) và in ra kết quả, giúp xác thực tính năng prompts đã trình bày trong MCP .
- Tích hợp và test từng loại transport:
  - `test_mcp_server_with_stdio_transport` khởi động MCP server sử dụng standard input/output transport, kết nối với client và test bằng các hàm trên.
  - `test_mcp_server_with_http_transport` kết nối tới MCP server qua HTTP (hoặc SSE), thử nghiệm toàn bộ workflow thông qua API chuẩn hoá theo MCP (JSON-RPC).

Luồng hoạt động tổng thể của MCP client chúng ta vừa thiết kế được tóm tắt như sau:

1. Lựa chọn transport phù hợp và kết nối tới MCP server (theo đúng kiến trúc client-server chuẩn của MCP).
2. Thực hiện handshake/initialize session với server.
3. Truy vấn metadata về server (tools, resources, resource templates, prompts).
4. Thực thi tuần tự các tác vụ: đọc/thêm/xoá sách, truy xuất resource động/tĩnh, lấy prompts.
5. In ra kết quả chi tiết từng bước để kiểm tra hoạt động và minh họa cho kiến trúc primitives của MCP .

### Chạy MCP Server và Client với stdio Transport

Đối với stdio transport, chúng ta không cần chạy server. Thay vào đó, chúng ta đưa đường dẫn của file `server.py` đến cho client. Sau đó, chúng ta chạy file `client.py` từ terminal với lệnh:

```
python ./client.py --transport stdio
```

Kết quả nhận được là:

#### Kết quả chạy file client.py để test Library MCP Server

```
*** Available Tools ***
Tool: add_book
 Add a book to the library
Tool: remove_book
 Remove a book by its ISBN
Tool: get_num_books
 Get the total number of books

*** Available Resources ***
Resource: All Books (books://all)

*** Available Resource Templates ***
Resource Template: Book by Index (books://index/{index})
 Get a book by its index in the library
```

```
Resource Template: Book by ISBN (books://isbn/{isbn})
Get a book by its ISBN

*** Available Prompts ***
Prompt: suggest_random_book (suggest_random_book)
Suggest a random book from the library. The
suggestion should include the title, author, and a
brief description.
Prompt: suggest_book_title_by_abstract (
 suggest_book_title_by_abstract)
Suggest a memorable, descriptive title for a book
based on the following abstract.
Prompt: analyze_book (analyze_book)
Analyze a book based on its content and user query.

#####
*** Calling Tool: get_num_books ***
Results=2

*** Get all books ***
All Books:
[
{
 "title": "Atomic Habits",
 "author": "James Clear",
 "isbn": "9780735211292",
 "tags": [
 "self-help",
 "psychology",
 "productivity"
]
},
{
 "title": "The Pragmatic Programmer",
 "author": "Andrew Hunt, David Thomas",
 "isbn": "9780201616224",
 "tags": [
 "software engineering",
 "programming",
 "career"
]
}
]
```

```
]

*** Adding a new book ***
Response from add_book: Book 'Random Book Name 979' by
 Random Author 382 added to the library.

*** Number of Books in Library after addition ***
Number of Books in Library: 3

*** Retrieve the added book by index ***
Retrieved Book:
{
 "title": "Random Book Name 979",
 "author": "Random Author 382",
 "isbn": "9787352249742",
 "tags": [
 "biography",
 "history",
 "poetry"
]
}

*** Retrieve the added book by ISBN ***
Retrieved Book:
{
 "title": "Random Book Name 979",
 "author": "Random Author 382",
 "isbn": "9787352249742",
 "tags": [
 "biography",
 "history",
 "poetry"
]
}

*** Removing the added book ***
Response from remove_book: Book with ISBN
 '9787352249742' removed from the library.

*** Number of Books in Library after removal ***
Number of Books in Library: 2

#####
```

```
*** Get suggest_random_book prompt ***
Prompt Response: [PromptMessage(role='user', content=
 TextContent(type='text', text='Suggest a random book
from the library. The suggestion should include the
title, author, and a brief description.', annotations
=None, meta=None))]

*** Get suggest_book_title_by_abstract prompt ***
Prompt Response: [PromptMessage(role='user', content=
 TextContent(type='text', text='Suggest a memorable,
descriptive title for a book based on the following
abstract: A book about the wonders of the universe.',
annotations=None, meta=None))]

*** Get analyze_book prompt ***
Prompt Response: [
{
 "role": "user",
 "content": "This is the book I want to analyze:
\"{\\\"title\\\": \\"Random Book Name 979\\\", \\"author\\\": \\"Random Author 382\\\", \\"isbn\\\": \\"9787352249742\\\", \\"tags\\\": [\\"biography\",
\\\", \\"history\\\", \\"poetry\\\"]}\\"
},
{
 "role": "assistant",
 "content": "Sure! Let's analyze this book together.
What would you like to know?"
},
{
 "role": "user",
 "content": "What is the main theme of this book?"
}
]

#####
Test completed successfully!
```

Phân tích kết quả trả về, ta thấy được:

- Chúng ta thấy MCP client nhận được ba tools: add\_book, remove\_-

book, và get\_num\_books. Resource mà MCP client nhận được là All Books (books://all) và hai resource templates là Book by Index (books://index/index) và Book by ISBN (books://isbn/isbn). Cuối cùng, chúng ta có ba prompts là suggest\_random\_book, suggest\_book\_title\_by\_abstract, và analyze\_book.

- Trong quá trình test các tools, MCP client đã thực hiện các thao tác như lấy số lượng sách, thêm sách mới, truy xuất sách theo index và ISBN, xóa sách, và cuối cùng là lấy nội dung của các prompts. Kết quả trả về cho thấy các thao tác đều thành công và dữ liệu được xử lý chính xác.
- Việc truy xuất các prompts cũng cho thấy tính năng này hoạt động tốt, với các prompt được trả về dưới dạng danh sách các tin nhắn (messages) với vai trò người dùng và trợ lý.

### Chạy MCP Server và Client với Streamable HTTP Transport

Đối với giao thức này, chúng ta phải chạy MCP server trước với lệnh sau:

```
python ./server.py --transport http --port 8000
```

Sau đó, chúng ta chạy file client.py với lệnh:

```
python ./client.py --transport http
```

Kết quả trả về sẽ tương tự như khi chạy với stdio transport, nhưng thay vì sử dụng I/O của subprocess, nó sẽ sử dụng HTTP để giao tiếp với MCP server.

Toàn bộ source code của Library MCP Server được lưu trữ tại [đây](#).

### MCP Inspector

**MCP Inspector** là một công cụ tương tác dành cho nhà phát triển, hỗ trợ kiểm thử và gỡ lỗi các MCP server.

Để khởi động MCP Inspector, ta cần cài trước [Node.js](#) (^ 22.7.5). Sau đó, trong folder của server đã tạo ở trên (folder library\_mcp\_server\_and\_client), chúng ta chạy lệnh sau trong terminal:

```
npx @modelcontextprotocol/inspector
```

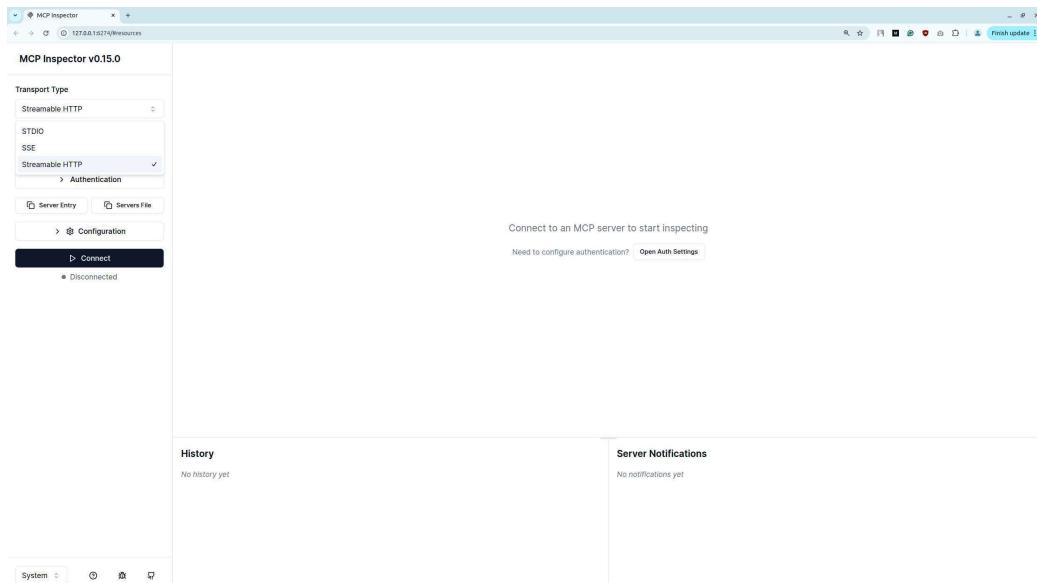
Output nhận được trong terminal là:

### Kết quả nhận được trong terminal

```
Starting MCP inspector...
Proxy server listening on 127.0.0.1:6277
Session token: ...
Use this token to authenticate requests or set DANGEROUSLY_-
OMIT_AUTH=true to disable auth

Open inspector with token pre-filled:
http://localhost:6274/?MCP_PROXY_AUTH_TOKEN=...
```

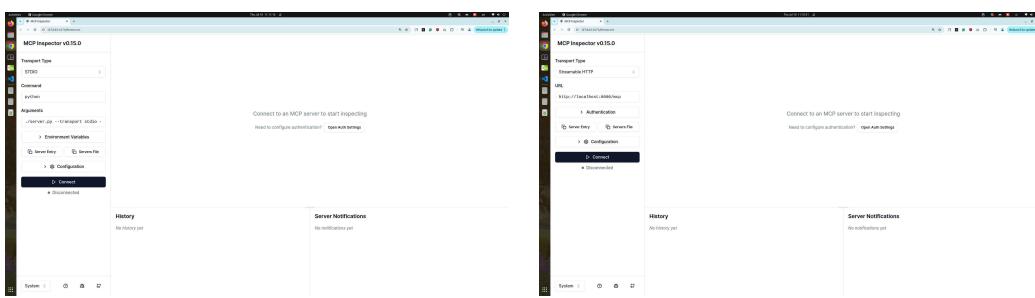
Chúng ta bật trình duyệt và truy cập vào URL có dạng: `http://localhost:6274/?MCP_PROXY_AUTH_TOKEN=...` trong output của terminal để sử dụng MCP Inspector. Giao diện của MCP Inspector giống với Hình 64.15.



Hình 64.15: MCP Inspector Homepage

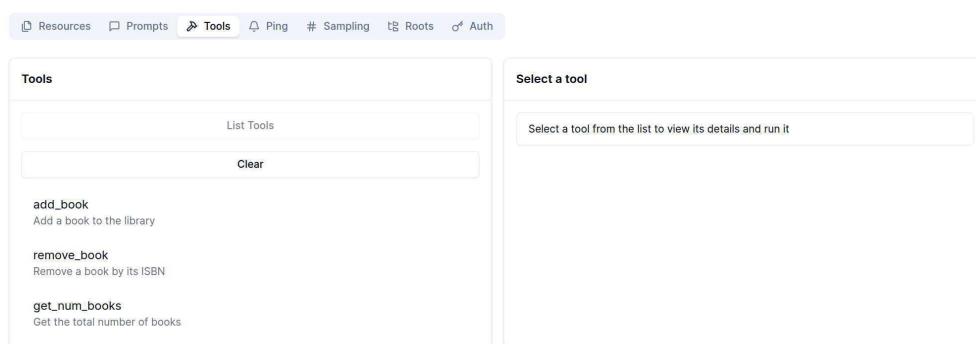
Đầu tiên, chúng ta sẽ kết nối với MCP server bằng sidebar ở bên trái (xem Hình 64.16). Đối với từng giá trị trong mục **Transport Type**, ta điền như sau:

- Nếu chúng ta chọn **STDIO** thì điền các field như sau:
  - Command: `python`,
  - Arguments: `./server.py -transport stdio -log-level error`
- Nếu chúng ta chọn **Streamable HTTP** thì chúng ta sẽ điền field URL là `http://localhost:8000/mcp`. và chúng ta ấn **Connect** để kết nối với server.



Hình 64.16: Filling Required Fields to Connect to MCP

Với MCP Inspector, ta có thể xem các tài nguyên mà MCP server cung cấp, bao gồm Tools, Resources và Prompts như Hình 64.17.



(a) Tools

The screenshot shows the MCP server Inspector interface with the 'Resources' tab selected. The top navigation bar includes links for Resources, Prompts, Tools, Ping, Sampling, Roots, and Auth.

Resources	Resource Templates	Select a resource or template
List Resources Clear all_books >	List Templates Clear book_by_index > book_by_isbn >	Select a resource or template from the list to view its contents

(b) Resources

The screenshot shows the MCP server Inspector interface with the 'Prompts' tab selected. The top navigation bar includes links for Resources, Prompts, Tools, Ping, Sampling, Roots, and Auth.

Prompts	Select a prompt
List Prompts Clear  suggest_random_book: Suggest a random book from the library. The suggestion should include the title, author, and a brief description.  suggest_book_title_by_abstract: Suggest a memorable, descriptive title for a book based on the following abstract.  analyze_book: Analyze a book based on its content and user query.	Select a prompt from the list to view and use it

(c) Promtps

Hình 64.17: Xem các core components của MCP server với Inspector

## 64.3 Cài đặt Personal Database MCP Server

### 64.3.1 Tạo thư mục project và cài thư viện

Chúng ta thực hiện những bước sau để tạo thư mục project và cài ứng dụng uv để quản lí thư viện của Python:

1. Cài uv

#### Cài uv trên Windows

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

#### Cài uv trên macOS

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

2. Tạo project folder và di chuyển vào thư mục project:

#### Tạo project folder với uv

```
uv init Personal-Database-MCP-Server
cd Personal-Database-MCP-Server/
```

Một thư mục tên **Personal-Database-MCP-Server** được tạo ra bằng uv và chúng ta sẽ di chuyển vào thư mục này để implement MCP server của chúng ta.

3. Trong folder của project , chúng ta tạo thêm các folder và file theo cấu trúc như sau (nếu file nào có sẵn do được tạo bởi uv thì ta bỏ qua):

```
personal_database_mcp_server/
|-- documents/
|-- .python-version
|-- create_vector_database.py
|-- prepare_documents.py
|-- pyproject.toml
|-- README.md
```

```
|-- retriever.py
|-- server.py
|-- uv.lock
```

4. Cài Python cho riêng project này với uv và kích hoạt Python:

#### Cài python riêng cho project trên Windows bằng uv

```
uv venv
.venv\Scripts\activate
```

#### Cài python riêng cho project trên macOS bằng uv

```
uv venv
source .venv/bin/activate
```

5. Cài thư viện:

#### Cài thư viện với uv trên Windows

```
uv add click mcp[cli] httpx langchain-community
langchain openai duckduckgo-search ddgs sentence-
transformers transformers qdrant-client
```

#### Cài thư viện với uv trên macOS

```
uv add click "mcp[cli]" httpx langchain-community
langchain openai duckduckgo-search ddgs sentence-
transformers transformers qdrant-client
```

### 64.3.2 Chuẩn bị dữ liệu và văn bản

Trong file `prepare_documents.py`, chúng ta sẽ tải về các bộ dữ liệu từ HuggingFace Datasets và lưu vào folder `documents`.

Để thuận tiện cho việc chuẩn bị data, chúng ta sẽ sử dụng các bộ dữ liệu có sẵn từ HuggingFace Datasets. Các bộ dữ liệu này chứa các định nghĩa về

keywords của từng ngành học, giúp chúng ta xây dựng một cơ sở dữ liệu cá nhân (Personal Database) phục vụ cho việc tìm kiếm và truy vấn thông tin. Danh sách các bộ dữ liệu này được lưu trong biến dataset\_ids như sau:

#### Danh sách tên các datasets dùng trong project này

```
1 dataset_ids = [
2 "burgerbee/pedagogy_textbook",
3 "burgerbee/pedagogy_wiki",
4 "burgerbee/psychology_textbook",
5 "burgerbee/psychology_wiki",
6 "burgerbee/psychiatry_textbook",
7 "burgerbee/psychiatry_wiki",
8 "burgerbee/art_and_culture_textbook",
9 "burgerbee/art_and_culture_wiki",
10 "burgerbee/medicine_textbook",
11 "burgerbee/medicine_wiki",
12 "burgerbee/chemistry_textbook",
13 "burgerbee/chemistry_wiki",
14 "burgerbee/social_studies_textbook",
15 "burgerbee/social_studies_wiki",
16 "burgerbee/religion_textbook",
17 "burgerbee/religion_wiki",
18 "burgerbee/science_studies_textbook",
19 "burgerbee/science_studies_wiki",
20 "burgerbee/history_textbook",
21 "burgerbee/history_wiki",
22 "burgerbee/philosophy_textbook",
23 "burgerbee/philosophy_wiki",
24 "burgerbee/biology_textbook",
25 "burgerbee/biology_wiki",
26 "burgerbee/physics_textbook",
27 "burgerbee/physics_wiki",
28]
```

Đây là những bộ dữ liệu chứa các định nghĩa về keywords của từng ngành.

Tiếp theo, chúng ta download dữ liệu trên về và lưu vào local trong folder documents như sau:

### Tải các datasets về local storage

```
1 from datasets import load_dataset
2 import os
3 import json
4 from datasets.utils.logging import set_verbosity_error
5 import tqdm
6
7 set_verbosity_error()
8
9 CACHE_DIR = "./cache"
10 DOCUMENT_DIR = "./documents"
11
12 dataset_ids = [
13 ... # dataset ids here (similar to dataset_ids mentioned in the
14 previous code block)
15]
16
17 for dataset_id in tqdm.tqdm(dataset_ids, desc="Downloading datasets"):
18 print(f"Downloading {dataset_id}...")
19 dataset = load_dataset(dataset_id, cache_dir=CACHE_DIR)
20 print(f"Downloaded {dataset_id} with {len(dataset)} splits.")
21
22 dataset_dir = os.path.join(DOCUMENT_DIR, dataset_id.split("/")[1])
23 os.makedirs(dataset_dir, exist_ok=True)
24
25 for split in dataset:
26 print(f" - {split}: {len(dataset[split])} examples")
27 split_data = len(dataset[split])
28
29 for i, sample in enumerate(dataset[split]):
30 title, text = sample["title"], sample["text"]
31 source = f"https://hf.co/datasets/{dataset_id}"
32 sample["source"] = source
33 sample["split"] = split
34 sample["id"] = f"{dataset_id}-{split}-{i}"
35
36 doc_path = os.path.join(dataset_dir, f"{title.replace
37 ('/', '_')}{i}.json")
38 with open(doc_path, "w", encoding="utf-8") as f:
39 json.dump(sample, f, ensure_ascii=False, indent=4)
```

```

39 print(f"Saved {split_data} samples to {dataset_dir}")
40 print(f"Finished processing {dataset_id}.\\n")
41 print("All datasets processed successfully.")

```

Dữ liệu tải về và được lưu vào máy trong các folder tương ứng với dataset name và split của nó. Ví dụ, nếu chúng ta tải về bộ dữ liệu burgerbee/art\_and\_culture\_textbook, các documents sẽ được lưu trong folder documents/art\_and\_culture\_textbook với các file json tương ứng với từng sample trong bộ dữ liệu. Cấu trúc của thư mục documents có dạng như sau:

```

1 documents/
2 |-- art_and_culture_textbook/
3 | |-- 21st century skills_18.json
4 | |-- Alternative fashion_40.json
5 | |-- ...
6 |
7 |-- history_textbook/
8 | |-- Ancient civilizations_01.json
9 | |-- World Wars_52.json
10 | |-- ...
11 |
12 |-- science_textbook/
13 | |-- Physics fundamentals_14.json
14 | |-- Chemistry basics_22.json
15 | |-- ...
16 |
17 |-- ...

```

### 64.3.3 Xây dựng Vector Store

Chúng ta sẽ chuyển đổi các documents đã tải về thành embedding vectors tương ứng, phục vụ cho việc lưu trữ và tìm kiếm sau này. Việc chuyển đổi này sẽ sử dụng một mô hình embedding đã được huấn luyện trước, giúp chúng ta tạo ra các vector đại diện cho nội dung của từng document.

Giai đoạn này sẽ được implement ở file theo các bước sau:

1. Với các documents đã tải xuống và lưu ở folder documents, chúng ta tìm tất cả vị trí của documents với hàm `discover_and_get_all_files`:

### Hàm discover\_and\_get\_all\_files

```

1 def discover_and_get_all_files(root_dir, allowed_extensions=
2 None, recursive=False):
3 if allowed_extensions is None:
4 allowed_extensions = ['.json', '.txt', '.md']
5
6 all_files = []
7 for item_name in os.listdir(root_dir):
8 item_path = os.path.join(root_dir, item_name)
9 if os.path.isfile(item_path) and any(item_path.
10 endswith(ext) for ext in
11 allowed_extensions):
12 all_files.append(item_path)
13 elif os.path.isdir(item_path) and recursive:
14 all_files.extend(discover_and_get_all_files(
15 item_path, allowed_extensions,
16 recursive))
17
18 return all_files
19
20
21 all_document_paths = discover_and_get_all_files(DOCUMENT_DIR,
22 allowed_extensions=['.json', '.txt', '.md'], recursive=True)
23 print(f"Found {len(all_document_paths)} documents in {DOCUMENT_DIR}.")
24
25 ### OUTPUT: Found 24954 documents in ./documents.

```

2. Chúng ta implements các hàm đọc documents từ file lưu trên ổ cứng như sau:

### Các hàm để đọc file lưu trên local storage

```

1 def load_json_file(file_path):
2 with open(file_path, 'r', encoding='utf-8') as f:
3 return json.load(f)
4
5 def load_txt_file(file_path):
6 with open(file_path, 'r', encoding='utf-8') as f:
7 return f.read()
8
9 def load_md_file(file_path):
10 with open(file_path, 'r', encoding='utf-8') as f:

```

```
11 return f.read()
```

Các hàm này sẽ giúp chúng ta đọc các file json, txt và md từ ổ cứng. Hàm `load_json_file` sẽ đọc file json và trả về nội dung của nó dưới dạng dictionary. Tương tự, hàm `load_txt_file` và `load_md_file` sẽ đọc file txt và md tương ứng và trả về nội dung văn bản của chúng. Những hàm này sẽ được sử dụng trong MCP server của chúng ta khi chúng ta load một văn bản nào đó để trả về dưới dạng resource.

- Khi có được, vị trí của chúng lưu tại `all_document_paths`, chúng ta sẽ load tất cả documents từ các file đã lưu trong folder `documents` và lưu chúng vào biến `documents`:

#### Load tất cả các documents lên memory

```
1 documents = []
2 with tqdm(total=len(all_document_paths), desc="Loading
 documents") as pbar:
3 for doc_path in all_document_paths:
4 if doc_path.endswith('.json'):
5 doc = load_json_file(doc_path)
6 elif doc_path.endswith('.txt'):
7 doc = {"text": load_txt_file(doc_path)}
8 elif doc_path.endswith('.md'):
9 doc = {"text": load_md_file(doc_path)}
10 else:
11 continue
12
13 # Add metadata
14 doc["path"] = doc_path
15 documents.append(doc)
16 pbar.update(1)
```

Mỗi document sẽ là một dictionary chứa nội dung văn bản của document với field `text` và các metadata khác như `path`, `source`, `split`, `id` (nếu có).

- Để chuyển đổi documents thành embedding, chúng ta dùng embedding model [Alibaba-NLP/gte-multilingual-base](#) và load model bằng thư viện `SentenceTransformers` như sau:

### Load embedding model

```

1 embedding_model_id = "Alibaba-NLP/gte-multilingual-base"
2 embedding_model = SentenceTransformer(
3 embedding_model_id,
4 trust_remote_code=True,
5 cache_folder='./cache'
6)

```

và convert các documents đó về embeddings:

### Chuyển các documents đã load về embedding

```

1 document_embeddings = embedding_model.encode(
2 [doc["text"] for doc in documents],
3 show_progress_bar=True,
4 convert_to_tensor=True,
5 normalize_embeddings=True,
6 batch_size=16,
7)

```

5. Để lưu trữ embedding của documents, chúng ta dùng [Qdrant Vector Database](#) cùng với thư viện [qdrant-client](#). Qdrant là một vector similarity search engine, cung cấp dịch vụ sẵn sàng cho production với API tiện lợi hỗ trợ việc lưu trữ, tìm kiếm và quản lý các points (vectors) với payload đi kèm. Chúng ta có thể coi payload là các thông tin bổ sung đi kèm với vector, hỗ trợ việc tìm kiếm, truy vấn và lưu trữ thông tin. Đầu tiên, chúng ta tạo Qdrant Database chạy ở local bằng một trong hai cách sau:

- (a) Dữ liệu được lưu trực tiếp vào folder trên ổ cứng:

Cách này phù hợp cho việc xây dựng ứng dụng trong quá trình phát triển và kiểm thử. Chúng ta trực tiếp tạo `qdrant_client` trong Python như sau:

### Tạo Qdrant client để kết nối tới Qdrant database ở local

```

1 client = QdrantClient(path='./qdrant_database')

```

- (b) Dữ liệu được quản lý trong Docker Container:

Cách này phù hợp khi ứng dụng đã vào giai đoạn triển khai (production). Để chạy Qdrant Database dưới dạng một Docker container, chúng ta chạy lệnh sau trong terminal:

### Chạy Qdrant database với Docker

```
docker run -p 6333:6333 -p 6334:6334 \
-v "$(pwd)/qdrant_storage:/qdrant/storage:z"
\qdrant/qdrant
```

Sau đó, chúng ta kết nối và tương tác với Qdrant container thông qua client bằng lệnh:

### Kết nối tới Qdrant database ở Docker

```
1 client = QdrantClient(url="http://localhost:6333")
```

6. Tiếp theo, chúng ta sẽ tạo một collection tên `mcp_database` để lưu trữ tất cả dữ liệu vào collection này. Một collection trong Qdrant là một tập hợp các points (vectors với payload) mà bạn có thể tìm kiếm. Mỗi vector của các point trong cùng một collection phải có cùng kích thước và được so sánh bằng một metric duy nhất (ví dụ: cosine similarity, Euclidean distance, v.v.). Các vector được đặt tên có thể được sử dụng để có nhiều vector trong một point, mỗi vector có thể có yêu cầu về kích thước và metric riêng. Chúng ta tạo collection này bằng cách sử dụng method `create_collection` của `client` như sau:

### Tạo collection `mcp_database` trong Qdrant client

```
1 client.create_collection(
2 collection_name="mcp_database",
3 vectors_config=models.VectorParams(
4 size=embedding_model.get_sentence_embedding_dimension
5 (),
6 distance=models.Distance.COSINE
7)
```

7 )

Trong đó, các arguments được giải thích như sau:

- `collection_name="mcp_database"`: Tên của collection được tạo là `mcp_database`.
- `vectors_config`: Định nghĩa cấu hình cho các vector sẽ lưu trữ trong collection này.
  - `size=embedding_model.get_sentence_embedding_dimension()`: Độ dài của mỗi vector embedding, lấy từ mô hình embedding đang sử dụng.
  - `distance=models.Distance.COSINE`: Phương thức tính toán độ tương đồng giữa hai embedding vectors. Chúng ta sử dụng cosine similarity trong bài demo này.

7. Sau khi kết nối tới Qdrant Database bằng một trong hai cách trên và tạo collection `mcp_database`, chúng ta lưu các embeddings cùng với các documents tương ứng vào database với method `upload_points`:

#### Lưu dữ liệu embedding của documents cùng với nội dung tương ứng vào Qdrant database

```
1 client.upload_points(
2 collection_name="mcp_database",
3 points=[
4 models.PointStruct(
5 id=idx, vector=document_embeddings[idx],
6 payload={
7 "text": doc["text"],
8 "path": doc["path"],
9 "source": doc.get("source", ""),
10 "split": doc.get("split", ""),
11 "id": doc.get("id", "")
12 }
13) for idx, doc in enumerate(documents)
14],
15)
```

Trong đó, mỗi point đại diện cho một document với các attribute như sau:

- **id=idx**: ID duy nhất của point, được gán bằng chỉ số của document trong danh sách.
- **vector=document\_embeddings[idx]**: Vector embedding tương ứng với document.
- **payload**: Chứa các thông tin metadata của document như:
  - **text**: Nội dung văn bản của document.
  - **path**: Đường dẫn tới file lưu trữ document trên ổ cứng.
  - **source**: Nguồn gốc của document (nếu có).
  - **split**: Phân chia dữ liệu (nếu có).
  - **id**: ID của sample trong dataset (nếu có).

#### 64.3.4 Xây dựng Retriever

Chúng ta tạo file mới `retriever.py` để định nghĩa các hàm và class cần thiết cho việc truy vấn dữ liệu từ Qdrant Database. Trong file này, chúng ta sẽ định nghĩa class `Retriever` với các hàm thêm văn bản vào database và truy vấn văn bản từ database. Class này sẽ sử dụng Qdrant Client để tương tác với Qdrant Database và thực hiện các thao tác tìm kiếm.

1. Đầu tiên, chúng ta thêm các thư viện cần thiết vào file `retriever.py`:

##### Import các thư viện cần thiết của retriever

```

1 import logging
2 import uuid
3 from typing import Any, Dict, List, Optional
4
5 from qdrant_client import QdrantClient
6 from qdrant_client.http.models import Distance, PointStruct,
 VectorParams
7 from sentence_transformers import SentenceTransformer

```

2. Tiếp theo, chúng ta định nghĩa class `Retriever` với các methods để thêm văn bản vào database và truy vấn văn bản từ database:

### Class Retriever với các methods tương tác với cơ sở dữ liệu

```

1 class Retriever:
2 def __init__(
3 self,
4 embedding_model: SentenceTransformer,
5 qdrant_path: str,
6 collection_name: str = "documents",
7 embedding_size: Optional[int] = None,
8):
9 """
10 Initializes the Retriever with the embedding model,
11 Qdrant path, collection name,
12 and embedding size.
13
14 Args:
15 embedding_model (SentenceTransformer): The
16 embedding model to use for
17 encoding documents.
18 qdrant_path (str): The path to the Qdrant database
19 (local or URL).
20 collection_name (str): The name of the collection
21 in Qdrant to store documents.
22 embedding_size (Optional[int]): The size of the
23 embedding vectors. If None, it
24 will be inferred from the model
25
26 Raises:
27 ValueError: If embedding_size is not a positive
28 integer.
29
30 self.embedding_model = embedding_model
31 self.qdrant_path = qdrant_path
32 self.collection_name = collection_name
33
34 # Connect to Qdrant local instance (using the local
35 path)
36 self.client = QdrantClient(
37 path=self.qdrant_path,
38)
39
40 # If embedding_size is not provided, attempt to infer
41 it
42 if embedding_size is None:

```

```
31 embedding_size = self.embedding_model.
32 get_sentence_embedding_dimension()
33 else:
34 if embedding_size <= 0:
35 raise ValueError("Vector size must be a
36 positive integer.")
37 self.embedding_size = embedding_size
38
39 # Ensure the collection exists
40 if not self.client.collection_exists(self.
41 collection_name):
42 self.client.create_collection(
43 collection_name=self.collection_name,
44 vectors_config=VectorParams(size=
45 embedding_size, distance=
46 Distance.COSINE),
47)
48
49 def add_documents(self, documents: list[str]) -> Dict[str,
50 str]:
51 ...
52
53 def add_document(self, document: str) -> Dict[str, str]:
54 ...
55
56 def retrieve(self, query: str, limit: int = 5) -> List[
57 dict]:
58 ...
```

Trong đó, các methods được implement như sau:

- `__init__`: Khởi tạo class Retriever với các arguments như mô hình embedding, đường dẫn tới Qdrant Database, tên collection và kích thước embedding. Nếu kích thước embedding không được cung cấp, nó sẽ được lấy từ mô hình embedding.
- `add_documents`: Thêm một danh sách các documents vào Qdrant Database. Mỗi document sẽ được chuyển đổi thành embedding và lưu trữ trong collection với một UUID duy nhất.

### Method add\_documents để thêm nhiều documents vào database

```

1 ...
2
3 def add_documents(self, documents: list[str]) -> Dict[str
4 , str]:
5 """
6 Embeds the documents and adds them to the Qdrant
7 collection.
8 Each document is assigned a unique UUID.
9 """
10
11 try:
12 embeddings = self.embedding_model.encode(
13 documents)
14
15 points = [
16 PointStruct(
17 id=str(uuid.uuid4()), # unique id for
18 each document
19 vector=embedding,
20 payload={"text": doc},
21)
22 for doc, embedding in zip(documents,
23 embeddings)
24]
25 self.client.upsert(collection_name=self.
26 collection_name, points=
27 points)
28
29 return {"status": "success", "message": "
30 Documents added successfully
31 ."}
32
33 except Exception as e:
34 logging.error(f"Error adding documents: {e}")
35 return {"status": "error", "message": str(e)}
36
37 ...
38

```

- **add\_document:** Thêm một document đơn lẻ vào Qdrant Database. Tương tự như phương thức trên nhưng chỉ xử lý một document.

### Method add\_document để thêm một document riêng lẻ vào database

```

1 ...
2
3 def add_document(self, document: str) -> Dict[str, str]:
4 """
5 Embeds a single document and adds it to the Qdrant
6 collection.
7 The document is assigned a unique UUID.
8 """
9 try:
10 embedding = self.embedding_model.encode([document
11])[0]
12 point = PointStruct(
13 id=str(uuid.uuid4()), # unique id for the
14 document
15 vector=embedding,
16 payload={"text": document},
17)
18 self.client.upsert(collection_name=self.
19 collection_name, points=[point])
20
21 return {"status": "success", "message": "Document
22 added successfully."}
23 except Exception as e:
24 logging.error(f"Error adding document: {e}")
25 return {"status": "error", "message": str(e)}
26
27 ...

```

- **retrieve:** Truy vấn các documents tương tự với query đầu vào với metric cosine similarity. Method này trả về một danh sách các dictionary chứa nội dung của document và độ tương đồng giữa query với document tương ứng.

### Method retrieve để tìm documents có nội dung tương ứng trong database

```

1 ...
2

```

```

3 def retrieve(self, query: str, limit: int = 5) -> List[dict]:
4 """
5 Retrieves documents similar to the query using cosine
6 similarity.
7 Returns a list of dictionaries containing the
8 document text and its score.
9 """
10 query_embedding = self.embedding_model.encode([query])
11 [0].tolist()
12 search_result = self.client.query_points(
13 collection_name=self.collection_name,
14 query=query_embedding,
15 limit=limit,
16)
17
18 results = [
19 {"text": point.payload["text"], "score": point.
20 score} for point in
21 search_result.points
22]
23
24 return results
25
26 ...

```

3. Để kiểm tra logic của retriever, chúng ta test các hàm này trong file retriever.py với hàm như sau:

### Kiểm tra Retriever đã build bằng hàm main trong Python

```

1 if __name__ == "__main__":
2 embedding_model = SentenceTransformer(
3 "Alibaba-NLP/gte-multilingual-base", trust_remote_code
4 =True, cache_folder=".cache"
5)
6 retriever = Retriever(
7 embedding_model=embedding_model,
8 qdrant_path=".qdrant_database",
9 collection_name="mcp_database",
10)
11
12 results = retriever.retrieve("What is organic chemistry?", ...

```

```
12 limit=5)
13 for i, result in enumerate(results):
14 print(f"Result {i + 1} (Score: {result['score']:.4f}):
15 {result['text'][:250]}...")
```

Cuối cùng, chúng ta chạy file retriever.py với Python bằng lệnh sau trong terminal:

```
python retriever.py
```

và nhận được output là các kết quả truy vấn từ Qdrant Database với các documents tương tự với query "What is organic chemistry?". Các kết quả này sẽ được sắp xếp theo điểm số tương ứng, phục vụ cho việc thiết kế các logic tìm kiếm nâng cao ở phía MCP server.

### Output khi chạy đoạn code trên

Result 1 (Score: 0.7766): Bioorganic chemistry is a scientific discipline that combines organic chemistry and biochemistry. It is that branch of life science that deals with the study of biological processes using chemical methods. Protein and enzyme function are examples of ...

Result 2 (Score: 0.7766): Bioorganic chemistry is a scientific discipline that combines organic chemistry and biochemistry. It is that branch of life science that deals with the study of biological processes using chemical methods. Protein and enzyme function are examples of ...

Result 3 (Score: 0.7766): Bioorganic chemistry is a scientific discipline that combines organic chemistry and biochemistry. It is that branch of life science that deals with the study of biological processes using chemical methods. Protein and enzyme function are examples of ...

Result 4 (Score: 0.7752): Organic Chemistry

Organic chemistry is a branch of chemistry that deals with the study of the structure, properties, and reactions of organic

compounds and materials. These compounds contain carbon atoms and are found in various forms, including liv...

Result 5 (Score: 0.7617): Organic chemistry is a subdiscipline within chemistry involving the scientific study of the structure, properties, and reactions of organic compounds and organic materials, i.e., matter in its various forms that contain carbon atoms. Study of structu...

#### 64.3.5 Xây dựng MCP Server

Cuối cùng, chúng ta tạo file `server.py` để định nghĩa MCP Server của chúng ta. Các chức năng của Personal Database MCP Server sẽ bao gồm:

- **Tools:** Cung cấp các tools để thực hiện các tác vụ sau:
  - Truy vấn các documents từ Qdrant Database dựa trên query của người dùng,
  - Tìm kiếm query của người dùng trên internet bằng công cụ DuckDuckGo Search (nếu cơ sở dữ liệu cá nhân không có thông tin cần thiết),
  - Lưu trữ một document vào cơ sở dữ liệu cá nhân (Qdrant Database) và lưu xuống local file system để sử dụng sau này,
- **Resources:** Cung cấp các resources sau đây để lấy documents từ cơ sở dữ liệu cá nhân:
  - Lấy danh sách các topics trong cơ sở dữ liệu cá nhân,
  - Lấy tất cả các documents trong một topic cụ thể,
  - Lấy danh sách các documents trong một topic cụ thể với phân trang,
- **Prompts:** Cung cấp các mẫu prompt templates để hỗ trợ việc sử dụng MCP server này hiệu quả hơn, bao gồm các prompts có chức năng như sau:
  - Prompt để truy vấn các documents từ Qdrant Database,
  - Prompt để tìm kiếm thông tin từ internet,

- Prompt để thêm một document vào cơ sở dữ liệu cá nhân,

Chúng ta sẽ sử dụng thư viện `FastMCP` được cung cấp trong framework `mcp` để xây dựng Personal Library MCP Server theo các bước dưới đây:

1. Đầu tiên, chúng ta import các packages cần thiết cho việc xây dựng MCP server:

**Import thư viện để xây dựng Personal Library MCP Server**

```

1 import logging
2 import os
3 from typing import Any, Dict, List, Optional
4
5 import click
6 from langchain_community.tools import DuckDuckGoSearchResults
7 from mcp.server.fastmcp import FastMCP
8 from mcp.server.fastmcp.prompts import base
9 from pydantic import BaseModel, Field
10 from sentence_transformers import SentenceTransformer
11 from retriever import Retriever
12 from create_vector_database import (
13 discover_and_get_all_files,
14 load_json_file,
15 load_txt_file,
16 load_md_file,
17)
18 import uuid
19 import json

```

2. Chúng ta thiết kế hàm `run_mcp_server` để khởi tạo cơ sở dữ liệu, load các documents và chạy MCP Server như sau:

**Hàm run\_mcp\_server để khởi tạo và chạy Personal Database MCP Server**

```

1 def run_mcp_server():
2 embedding_model = SentenceTransformer(
3 "Alibaba-NLP/gte-multilingual-base",
4 trust_remote_code=True,
5 cache_folder=".cache",
6)

```

```

7 qdrant_path = "./qdrant_database"
8
9 retriever = Retriever(
10 embedding_model=embedding_model,
11 qdrant_path=qdrant_path,
12 collection_name="mcp_database",
13)
14
15 DOCUMENT_DIR = "./documents"
16 DOCUMENTS_PATH_BY_TOPICS = []
17 for folder in os.listdir(DOCUMENT_DIR):
18 if os.path.isdir(os.path.join(DOCUMENT_DIR, folder)):
19 DOCUMENTS_PATH_BY_TOPICS[folder] =
20 discover_and_get_all_files(
21 os.path.join(DOCUMENT_DIR, folder),
22 allowed_extensions=[".json", ".txt", ".md"],
23 recursive=True,
24)
25
26 ALL_DOCUMENT_PATHS = []
27 for folder in DOCUMENTS_PATH_BY_TOPICS:
28 ALL_DOCUMENT_PATHS.extend(DOCUMENTS_PATH_BY_TOPICS[
29 folder])
30
31 mcp_server = create_mcp_server(retriever,
32 DOCUMENTS_PATH_BY_TOPICS,
33 ALL_DOCUMENT_PATHS)
34 mcp_server.run(transport="streamable-http")

```

Hàm `run_mcp_server` sẽ thực hiện các bước sau:

- Khởi tạo mô hình embedding `embedding_model` sử dụng thư viện `sentence_transformers` với model Alibaba-NLP/gte-multilingual-base.
- Khởi tạo Qdrant Database với đường dẫn `qdrant_database` và tên collection `mcp_database`.
- Load các document paths từ thư mục `documents` vào hai dictionary:
  - `DOCUMENTS_PATH_BY_TOPICS`: Dictionary chứa danh sách các document paths theo tên topic,
  - `ALL_DOCUMENT_PATHS`: Danh sách tất cả các document paths.
- Khởi tạo MCP server với các tools, resources và prompts bằng

hàm `create_mcp_server`.

- Chạy MCP server với transport `streamable-http`.
3. Tiếp theo, chúng ta định nghĩa các tools, resources và prompts cho MCP server trong hàm `create_mcp_server`:

### Hàm `create_mcp_server` để khởi tạo Personal Database MCP Server

```

1 def create_mcp_server(
2 retriever: Retriever,
3 documents_path_by_topics: Dict[str, List[str]],
4 all_document_paths: List[str],
5):
6 mcp = FastMCP(
7 name="Retriever MCP Server",
8 host="127.0.0.1",
9 port=2545,
10 description="A server that provides MCP-powered
11 agentic RAG capabilities.",
12)
13 ...
14
15 return mcp

```

Chúng ta sẽ đi thiết kế lần lượt các tools, resources và prompts cho MCP Server để điền vào dấu ... trong hàm `create_mcp_server`.

## 4. Tools

Để xây dựng tools cho MCP Server với FastMCP, ta sử dụng `@tool` decorator với các arguments như sau:

- `name`: Tên tùy chọn cho công cụ (mặc định sẽ dùng tên của function)
- `title`: Tiêu đề tùy chọn, dễ hiểu dành cho con người
- `description`: Mô tả tùy chọn về chức năng của công cụ
- `annotations`: ToolAnnotations tùy chọn, cung cấp thông tin bổ sung cho công cụ

- **structured\_output**: Điều khiển việc đầu ra của công cụ có được cấu trúc hay không
  - Nếu là `None`: tự động phát hiện dựa trên kiểu dữ liệu trả về được chú thích trong function
  - Nếu là `True`: luôn tạo tool với đầu ra có cấu trúc (nếu kiểu trả về cho phép)
  - Nếu là `False`: luôn tạo tool với đầu ra không có cấu trúc

(a) **Tool tìm kiếm documents từ Qdrant Database**

Chúng ta sẽ sử dụng tool này để truy vấn các documents từ Qdrant Database dựa trên query của người dùng. Input của tool này tuân theo schema sau:

**Schema QueryInput**

```

1 class QueryInput(BaseModel):
2 query: str = Field(description="The query to retrieve
3 relevant documents.")
4 num_documents: int = Field(default=5, description="
5 The number of documents to
6 retrieve for the query.")

```

Tool `retrieve_documents_from_database` được định nghĩa như sau:

**Tool retrieve\_documents\_from\_database**

```

1 @mcp.tool(
2 name="retrieve_documents_from_database",
3 title="Retrieve Documents from Database",
4 description="Retrieve documents similar to the query
5 from the database.",
6)
7 def retrieve(input: QueryInput) -> RetrievalResult:
8 retrieval_results = retriever.retrieve(input.query,
9 input.num_documents)
10 return RetrievalResult(
11 results=[
12 RetrievedDocument(text=result["text"], score=
13 result["score"]))

```

```
11 for result in retrieval_results
12]
13 }
```

Khi tool được gọi, method `retrieve` trong class `Retriever` sẽ được chạy để truy vấn các documents từ Qdrant Database. Tool này sẽ trả về kết quả theo schema `RetrievalResult`, chứa danh sách các documents tuân theo schema `RetrievedDocument` gồm nội dung của văn bản với điểm số tương ứng:

**Output schema của tool retrieve\_documents\_from\_database**

(b) Tool tìm kiếm query trên internet

Để thiết kế tool `search_query_on_internet` này, ta dùng class `DuckDuckGoSearchResults` từ thư viện `langchain_community.tools`. Đầu vào của tool này là query của người dùng và số lượng documents cần tìm kiếm giống như tool `retrieve_documents_from_database` ở trên. Tiếp theo, chúng ta implement tool này như sau:

Tool search query on internet

```
1 @mcp.tool(
2 name="search_query_on_internet",
3 title="Search the Query on the Internet".
```

```

4 description="Search for documents on the internet
 related to the query. This
 tool will be used when the
 database does not have
 relevant documents for the
 query.",
5)
6 def search_query_on_internet(input: QueryInput) ->
 SearchResult:
7 search_engine = DuckDuckGoSearchResults(
8 output_format="list", num_results=input.
9 num_documents, backend="text"
10 "
11)
12 search_results = search_engine.invoke(input.query)
13 return SearchResult(
14 results=[
15 RetrievedDocument(
16 text=f"Title: {result['title']}\nText: {result['snippet']}",
17 score=None,
18)
19]
)

```

Khi được thực thi, tool sẽ gọi method `invoke` trong class `DuckDuckGoSearchResults` để tìm kiếm query trên internet với các arguments `output_format="list"` (kết quả trả về là một list of documents) và `num_results` là số lượng documents cần tìm kiếm.

Tương tự như tool `retrieve_documents_from_database`, tool này sẽ trả về một danh sách các documents  `SearchResult` có liên quan đến query đầu vào dưới dạng danh sách các documents `RetrievedDocument` với điểm số tương ứng.

(c) **Tool lưu trữ một document vào cơ sở dữ liệu cá nhân**

Chúng ta thiết kế tool `add_document_to_database` để lưu trữ một document vào cơ sở dữ liệu cá nhân:

```

Tool add_document_to_database

1 @mcp.tool(
2 name="add_document_to_database",
3 title="Add a Document to Database",
4 description="Add a document to the database for
5 future retrieval.",
6)
7 def add_document_to_database(
8 document: str, topic_name: Optional[str] = None,
9 document_name: Optional[str]
10 = None
11) -> AddDocumentResponse:
12 doc_id = str(uuid.uuid4())
13 if topic_name is None:
14 topic_name = "default"
15 if not os.path.exists(f"./documents/{topic_name}"):
16 os.makedirs(f"./documents/{topic_name}")
17 if document_name is None:
18 document_name = doc_id
19
20 # save to the documents folder
21 with open(f"./documents/{topic_name}/{document_name}.
22 json", "w") as f:
23 doc_data = {
24 "text": document,
25 "path": f"./documents/{topic_name}/{
26 document_name}.json",
27 "source": "user-added",
28 "split": topic_name,
29 "id": doc_id,
30 }
31 json.dump(doc_data, f, indent=4)
32
33 # add to the database
34 retriever.add_document(doc_data)
35
36 return AddDocumentResponse(status="success", message=
37 "Document added to database.
38 ")

```

Hàm được xây dựng như sau:

- Tool nhận vào ba arguments:

- `document`: Nội dung của document cần lưu trữ,
- `topic_name`: Tên topic của document,
- `document_name`: Tên của document.
- Tiếp theo, chúng ta tiến hành xử lý bổ sung các thông tin của document như sau:
  - Tạo một UUID duy nhất cho document,
  - Nếu `topic_name` không được cung cấp, chúng ta sẽ sử dụng giá trị mặc định là "`default`",
  - Nếu `document_name` không được cung cấp, chúng ta sẽ sử dụng UUID làm tên của document,
  - Tạo thư mục cho topic nếu chưa tồn tại,
- Tiếp theo, chúng ta lưu document vào thư mục topic với tên là `document_name` dưới dạng file `.json`:
  - Tạo một dictionary chứa các thông tin của document,
  - Lưu dictionary này vào file `{document_name}.json` trong thư mục `{topic_name}`,
- Cuối cùng, chúng ta thêm document vào Qdrant Database với method `add_document` trong class `Retriever` và trả về kết quả.

## 5. Resources

Trong Personal Database MCP Server, resources của chúng ta là những documents được lưu trong folder `documents` với các định dạng `.json`, `.txt` và `.md` (chủ yếu là `.json`). Do đó, chúng ta sẽ thiết kế các MCP resources để lấy các resources này.

Để xây dựng resources cho MCP Server với FastMCP, ta sử dụng `@resource` decorator với các arguments như sau:

- `uri`: URI của resource (có thể chứa các arguments như `{topic_name}`, `{page_number}`), ví dụ như `document://topics` hoặc `document://topics/{topic_name}`
- `name`: Tên tùy chọn cho resource (mặc định sẽ dùng tên của function định nghĩa resource)
- `title`: Tiêu đề tùy chọn, dễ hiểu dành cho người dùng
- `description`: Mô tả tùy chọn về chức năng của resource

- `mime_type`: MIME type tùy chọn cho resource, ví dụ như `text/plain`, `application/json`, `application/pdf`, v.v.

(a) **Resources lấy danh sách các topics trong cơ sở dữ liệu cá nhân**

Chúng ta thiết kế resource `get_all_topics` với URI `document://topics` để lấy danh sách các topics trong cơ sở dữ liệu cá nhân:

**Resource document://topics**

```

1 @mcp.resource(
2 uri="document://topics",
3 name="get_all_topics",
4 title="Get All Topics in the Database",
5 description="A resource to get all topics in the
6 database.",
7)
8 def get_all_topics() -> List[str]:
9 return sorted(list(documents_path_by_topics.keys()))

```

Khi được gọi, resource này sẽ trả về danh sách các topics trong cơ sở dữ liệu cá nhân.

(b) **Resources lấy tất cả các documents trong một topic cụ thể**

Để lấy tất cả các documents trong một topic cụ thể, chúng ta sẽ sử dụng resource `get_all_documents_by_topic` với URI `document://topics/{topic_name}`:

**Resource document://topics/{topic\_name}**

```

1 @mcp.resource(
2 uri="document://topics/{topic_name}",
3 name="get_all_documents_by_topic",
4 title="Get All Documents by Topic",
5 description="A resource to get all documents by topic
6 .",
7)
8 def get_all_documents_by_topic(topic_name: str) ->
9 LocalDocumentList:
10 doc_paths = documents_path_by_topics[topic_name]

```

```
9 documents = []
10 for doc_path in doc_paths:
11 if doc_path.endswith(".json"):
12 doc = load_json_file(doc_path)
13 doc = LocalDocument(
14 text=doc["text"],
15 path=doc_path,
16 source=doc.get("source", ""),
17 split=doc.get("split", ""),
18 id=doc.get("id", ""),
19)
20 elif doc_path.endswith(".txt"):
21 doc = LocalDocument(
22 text=load_txt_file(doc_path),
23 path=doc_path,
24 source=None,
25 split=None,
26 id=None,
27)
28 elif doc_path.endswith(".md"):
29 doc = LocalDocument(
30 text=load_md_file(doc_path),
31 path=doc_path,
32 source=None,
33 split=None,
34 id=None,
35)
36 else:
37 continue
38 documents.append(doc)
39 return LocalDocumentList(documents=documents)
```

Resource `get_all_documents_by_topic` được lấy bằng URI `document://topics/{topic_name}` với argument `topic_name` là tên của topic cần lấy documents.

Khi được gọi, resource này sẽ lấy toàn bộ đường dẫn các documents trong topic cụ thể, đọc nội dung của từng document và trả về danh sách các documents `LocalDocument` với các thông tin của document, bao gồm `text`, `path`, `source`, `split` và `id`:

### Output schema của resource get\_all\_documents\_by\_topic

```

1 class LocalDocument(BaseModel):
2 text: str = Field(description="The content of the
3 local document.")
4 path: str = Field(description="The path of the local
5 document.")
6 source: Optional[str] = Field(None, description="The
7 source of the local document
8 .")
9 split: Optional[str] = Field(None, description="The
10 split of the local document.
11 ")
12 id: Optional[str] = Field(None, description="The id
13 of the local document.")

14
15 class LocalDocumentList(BaseModel):
16 documents: List[LocalDocument] = Field(description=""
17 The list of local documents.
18 ")

```

(c) Resources lấy các documents trong một topic cụ thể với cơ chế phân trang

Cơ chế phân trang được sử dụng để tránh việc lấy quá nhiều documents cùng lúc, đặc biệt là khi số lượng documents trong một topic quá nhiều.

Để lấy các documents của một topic với phân trang, chúng ta sẽ sử dụng resource `get_documents_by_topic` với URI `document://topics/{topic_name}/pages/{page_number}`:

### Resource `document://topics/{topic_name}/pages/{page_number}`

```

1 @mcp.resource(
2 uri="document://topics/{topic_name}/pages/{{
3 page_number}}",
4 name="get_documents_by_topic",
5 title="Get Documents by Topic",
6 description="A resource to get documents by topic"

```

```

6) using pagination. Each page
7 def get_documents_by_topic(topic_name: str, page_number: contains 10 documents.",

8 int) -> LocalDocumentList:
9
10 doc_paths = documents_path_by_topics[topic_name]
11 start_index, end_index = (page_number - 1) * 10,
12 page_number * 10
13
14 documents = []
15
16 for doc_path in doc_paths[start_index:end_index]:
17 if doc_path.endswith(".json"):
18 doc = load_json_file(doc_path)
19 elif doc_path.endswith(".txt"):
20 doc = LocalDocument(
21 text=load_txt_file(doc_path),
22 path=doc_path,
23 source=None,
24 split=None,
25 id=None,
26)
27 elif doc_path.endswith(".md"):
28 doc = LocalDocument(
29 text=load_md_file(doc_path),
30 path=doc_path,
31 source=None,
32 split=None,
33 id=None,
34)
35 else:
36 continue
37
38 documents.append(doc)
39
40 return LocalDocumentList(documents=documents)

```

Resource `get_documents_by_topic` được lấy bằng URI `document://topics/{topic_name}/pages/{page_number}` với argument `topic_name` là tên của topic cần lấy documents và `page_number` là vị trí trang cần lấy. Khi được gọi, resource này sẽ trả về tối đa 10 documents. Kết quả trả về tuân theo kiểu dữ liệu `LocalDocumentList`, chứa danh sách các documents `LocalDocument` với các thông tin của document, bao gồm `text`, `path`, `source`, `split` và `id` tương tự như kết quả trả về của resource `get_all_documents_by_topic`.

## 6. Prompts

Bên cạnh tools và resources, chúng ta cũng thiết kế thêm các prompts để hỗ trợ người dùng sử dụng Personal Database MCP Server.

Chúng ta sẽ viết các prompt cho MCP server của chúng ta bằng `@prompt()` decorator với các arguments sau:

- `name`: Tên tùy chọn cho prompt (mặc định là tên của hàm)
- `title`: Tiêu đề hiển thị dễ hiểu dành cho con người (tùy chọn)
- `description`: Mô tả tùy chọn về chức năng hoặc mục đích của prompt. Nếu field này bị bỏ trống, phần docstring của hàm định nghĩa prompt sẽ được sử dụng thay thế.

### (a) Prompt `retrieve_documents_from_database_prompt`

Prompt này được thiết kế để lấy các documents từ cơ sở dữ liệu cá nhân như sau:

```
Prompt retrieve_documents_from_database_prompt

1 @mcp.prompt(
2 name="retrieve_documents_from_database_prompt",
3 title="Retrieve Documents from Database Prompt",
4 description="Prompt to retrieve documents similar to
5 the query from the database.
6 ",
7)
8 def get_retrieve_document_from_database_tool(input:
9 QueryInput) -> base.
10 UserMessage:
11
12 return base.UserMessage(
13 content=f"""
14 Answer the following question.
15 Remember to use the 'retrieve_documents_from_database'
16 tool to find {input}.
17 num_documents} relevant
18 documents in the database to
19 support your answer.
20
21 Question: {input.query}
22 """
23)
```

Prompt nhận vào argument `input` là kiểu dữ liệu `QueryInput`, gồm query và số lượng documents cần lấy, và trả về một message `UserMessage` chứa nội dung prompt.

- (b) Prompt `retrieve_document_and_search_internet_prompt` để lấy các documents từ cơ sở dữ liệu cá nhân và tìm kiếm trên internet nếu cần:

**Prompt** retrieve\_document\_and\_search\_internet\_-  
prompt

```
1 @mcp.prompt(
2 name="retrieve_document_and_search_internet_prompt",
3 title="Retrieve Document and Search Internet Prompt",
4 description="Prompt to retrieve documents similar to
5 the query from the database
6 and search the internet if
7 necessary.",
8)
9
10 def get_retrieve_document_and_search_internet_tool(input:
11 QueryInput) -> base.
12 UserMessage:
13
14 return base.UserMessage(
15 content=f """Answer the following question.
16
17 Remember to use the '
18 retrieve_documents_from_database
19 ' tool to find {input.
20 num_documents} relevant
21 documents in the database to
22 support your answer. If the
23 database does not have
24 relevant documents, use the
25 ,
26 search_documents_on_internet
27 ' tool to search for
28 documents on the internet.
29
30
31 Question: {input.query}
32 """
33)
```

Tương tự như prompt `retrieve_documents_from_database_prompt`, prompt này cũng nhận vào argument `input` là kiểu dữ

liệu QueryInput, gồm query và số lượng documents cần lấy, và trả về một message UserMessage chứa nội dung prompt. Tuy nhiên, prompt này còn thêm câu hỏi nếu cơ sở dữ liệu không có documents liên quan, thì sẽ sử dụng tool `search_documents_on_internet` để tìm kiếm trên internet.

- (c) Prompt `search_query_on_internet_prompt` để tìm kiếm trên internet trực tiếp (không cần lấy từ cơ sở dữ liệu cá nhân):

**Prompt search\_query\_on\_internet\_prompt**

```

1 @mcp.prompt(
2 name="search_query_on_internet_prompt",
3 title="Search Internet Prompt",
4 description="Prompt to search the internet for
5 documents related to the
6 query.",
7)
8 def get_search_internet_tool(input: QueryInput) -> base.
9 UserMessage:
10 return base.UserMessage(
11 content=f"""
12 Answer the following question.
13 Remember to use the
14 'search_documents_on_internet'
15 'tool to find {input.
16 num_documents} relevant
17 documents on the internet to
18 support your answer.
19
20 Question: {input.query}
21 """
22)

```

Giống với prompt `retrieve_documents_from_database_prompt`, prompt này cũng nhận vào argument `input` là kiểu dữ liệu `QueryInput`, gồm query và số lượng documents cần lấy, và trả về một message `UserMessage` chứa nội dung prompt. Tuy nhiên, prompt này không sử dụng tool `retrieve_documents_from_database` để lấy documents từ cơ sở dữ liệu cá nhân, mà sẽ tìm kiếm trên internet trực tiếp.

- (d) Prompt `add_single_document_to_database_prompt` để thêm một document vào cơ sở dữ liệu cá nhân:

**Prompt add\_single\_document\_to\_database\_prompt**

```

1 @mcp.prompt(
2 name="add_single_document_to_database_prompt",
3 title="Add Single Document to Database Prompt",
4 description="Prompt to add a single document to the
5 database.",
6)
7 def get_add_single_document_to_database_tool(document:
8 str) -> base.UserMessage:
9 return base.UserMessage(
10 content=f"""Add the following document to the
11 database for future
12 retrieval using the '
13 add_single_document_to_database
14 ' tool.
15
16 Document: {document}
17 """
18)

```

Prompt nhận vào argument `document` là nội dung của document cần thêm vào cơ sở dữ liệu cá nhân, và trả về một message `UserMessage` chứa nội dung prompt.

#### 64.3.6 Chạy MCP Server và tích hợp vào Claude Desktop

**Run MCP Server** Để chạy MCP Server, chúng ta có thể thực thi một trong các lệnh sau trong terminal:

- Chạy bằng uv: `uv run server.py`
- Chạy bằng python: `python server.py`

Kết quả chạy MCP server sẽ hiển thị như sau:

### Output trong terminal khi chạy MCP server

```
INFO: Started server process [175875]
INFO: Waiting for application startup.
2025-07-13 11:18:23,876 - mcp.server.
 streamable_http_manager - INFO - StreamableHTTP
 session manager started
INFO: Application startup complete.
INFO: Uvicorn running on http://127.0.0.1:2545 (
 Press CTRL+C to quit)
```

Chúng ta sử dụng URL <http://127.0.0.1:2545> để kết nối với MCP Server ở phần tiếp theo.

**Test MCP Server** Để test MCP Server, chúng ta sử dụng tool MCP Inspector. Cách chạy tool này đã được trình bày chi tiết ở mục [64.2.6](#). Chúng ta bật trình duyệt và truy cập vào URL có dạng: [http://localhost:6274/?MCP\\_PROXY\\_AUTH\\_TOKEN=...](http://localhost:6274/?MCP_PROXY_AUTH_TOKEN=...) trong output của terminal để sử dụng MCP Inspector.

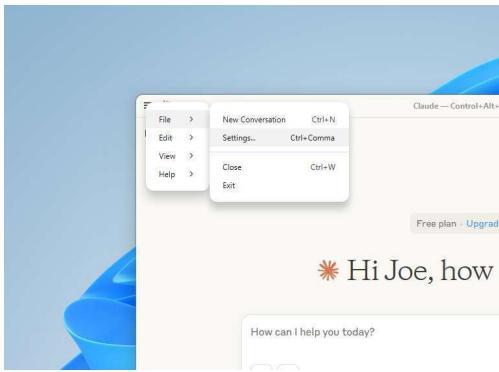
Ở thanh sidebar bên trái, chúng ta nhập vào các thông số như sau để kết nối với MCP Server:

- Transport Type: Chọn Streamable HTTP,
- URL: <http://127.0.0.1:2545>.

và ấn nút Connect để kết nối với MCP Server. Sau khi kết nối thành công, chúng ta có thể sử dụng các tool của MCP Server để test như trong hướng dẫn ở mục [64.2.6](#).

**Tích hợp MCP Server vào Claude Desktop** Để thêm MCP Server vào Claude Desktop, chúng ta thực hiện theo các bước sau:

1. Mở Claude Desktop và mở Setting (xem Hình [64.18](#)),



(a) Claude Desktop Setting on Windows

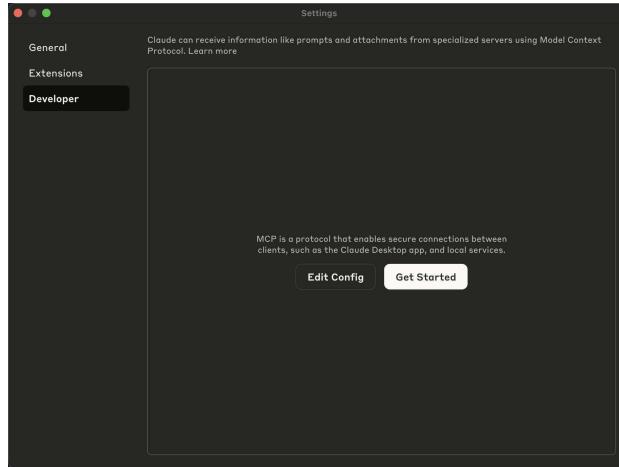


(b) Claude Desktop Setting on MacOS

Hình 64.18: Claude Desktop Setting

2. Trong tab Developer, chọn button **Edit Config** (xem Hình 64.19) để tạo file `claude_desktop_config.json` trong thư mục sau:

- Window: `%APPDATA%\Claude\claude_desktop_config.json`
- MacOS: `/Library/Application\ Support/Claude/claude_desktop_config.json`



Hình 64.19: Developer Tab in Claude Config

3. Chúng ta thêm đoạn code sau vào file `claude_desktop_config.json`:

**Nội dung file config claude\_desktop\_config.json khi MCP server dùng Streamable HTTP transport**

```
{
 "mcpServers": {
 "Personal Database MCP Server": {
 "command": "npx",
 "args": ["mcp-remote",
 "http://127.0.0.1:2545/mcp"]
 }
 }
}
```

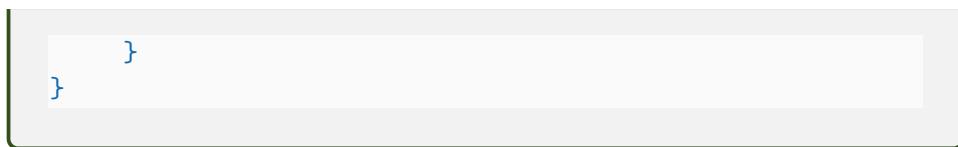
Trong đó, `http://127.0.0.1:2545/mcp` là URL đến MCP server của chúng ta. Nếu MCP server được deploy trên một máy chủ cloud, chúng ta có thể thay `127.0.0.1` bằng public IP của máy chủ cloud đó.

Ngoài ra, nếu chúng ta sử dụng stdio transport (định nghĩa ở phần code 3, dòng 33 với giá trị của argument `transport="stdio"`), chúng ta sử dụng đoạn code sau trong file config của Claude Desktop (thay `/ABSOLUTE/PATH/TO/OUR/PROJECT/DIRECTORY` thành đường dẫn đến project folder của chúng ta):

**Nội dung file config claude\_desktop\_config.json khi MCP server dùng stdio transport**

```
{
 "mcpServers": {
 "Personal Database MCP Server": {
 "command": "uv",
 "args": [
 "--directory",

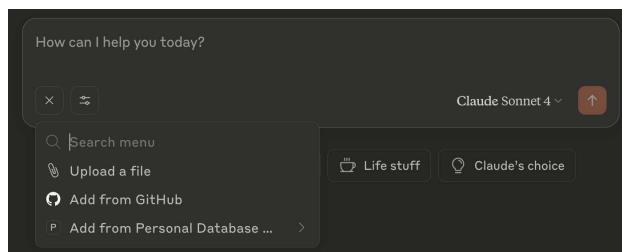
 "/ABSOLUTE/PATH/TO/OUR/PROJECT/DIRECTORY",
 "run",
 "server.py"
]
 }
 }
}
```



4. Khởi động lại Claude Desktop

#### 64.3.7 Sử dụng Prompts, Resources và Tools trên Claude Desktop

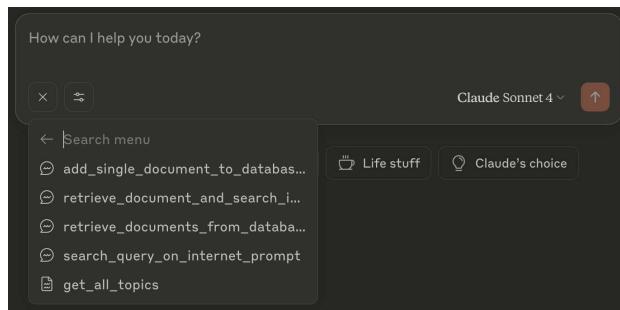
**Resources** Chúng ta xem danh sách các prompts của server bằng cách ấn vào nút + trong góc trái dưới của hộp thoại. Danh sách Resources và Prompts của từng MCP server sẽ hiển thị dưới dạng dropdown menu và phía dưới option Add from Github (xem Hình 64.20).



Hình 64.20: Danh sách các MCP Server

Sau đó, chúng ta chọn Personal Database MCP Server để xem danh sách các prompts và resources (xem Hình 64.21), bao gồm:

- Prompts: `retrieve_document_from_database`, `retrieve_document_and_search_internet`, `search_query_on_internet`, `add_single_document_to_database`
- Resources: `get_all_topics`



Hình 64.21: Danh sách Prompts và Resource của Personal Database MCP Server trong Claude Desktop

Chúng ta chọn resource `get_all_topics` để lấy danh sách các topics trong cơ sở dữ liệu cá nhân. Kết quả trả về sẽ được đính kèm trong hộp thoại chat của Claude Desktop như Hình 64.22a. Khi ấn vào file, nội dung sẽ được hiển thị như Hình 64.22b.

(a) Screenshot of the Claude Desktop chat interface. A user has entered the prompt `get_all_topics`. The response is shown in a separate window titled 'get\_all\_topics' containing a JSON array of topic names.

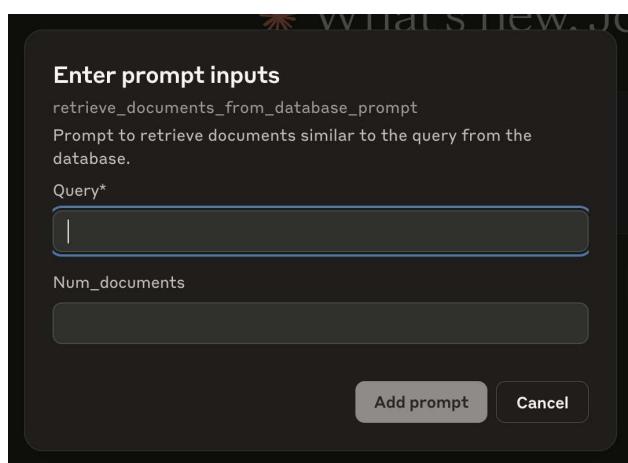
```
["art_and_culture_textbook",
 "art_and_culture_wiki",
 "biology_textbook",
 "biology_wiki",
 "chemistry_textbook",
 "chemistry_wiki",
 "default",
 "history_textbook",
 "history_wiki",
 "medicine_textbook",
 "medicine_wiki",
 "pedagogy_textbook",
 "pedagogy_wiki",
 "philosophy_textbook",
 "philosophy_wiki",
 "physics_textbook",
 "physics_wiki",
 "psychiatry_textbook",
 "psychiatry_wiki",
 "psychology_textbook",
 "psychology_wiki",
 "religion_textbook",
 "religion_wiki",
 "science_studies_textbook",
 "science_studies_wiki",
 "social_studies_textbook",
 "social_studies_wiki"
]
```

(b) Screenshot of the Claude Desktop interface showing the expanded content of the 'get\_all\_topics' resource. The JSON array is displayed as a list of topic names.

(a) Kết quả trả về khi lấy resource được đính kèm trong hộp thoại  
 (b) Xem nội dung của resource được trả về

Hình 64.22: Kết quả resource trả về từ MCP server được đính vào trong hộp thoại chat với Claude

**Prompt** Chúng ta chọn prompt `retrieve_document_from_database` và một popup hiển thị để chúng ta nhập vào các arguments của prompt đó (xem Hình 64.23).



Hình 64.23: Popup window để nhập arguments của prompt

Chúng ta nhập các field như sau:

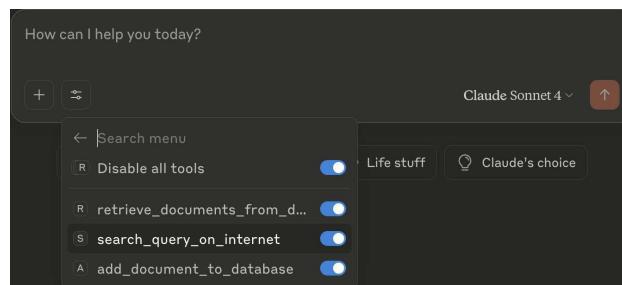
- Query: What is organic chemistry?
- Num\_documents: 5

Sau khi nhập xong, ta ấn Add prompt. Prompt trả về được đính kèm vào ô hộp thoại tương tự như khi thêm resource. Nội dung của prompt được trả về được hiển thị trong Hình 64.24.



Hình 64.24: Nội dung của prompt khi được đọc từ ô hộp thoại

**Tools** Sau khi có prompt `retrieve_document_from_database` được trả về trong hộp thoại, chúng ta gửi đoạn chat cho Claude. Lưu ý rằng chúng ta phải bật tool `retrieve_document_from_database` trong danh sách tools của Personal Database MCP Server (xem Hình 64.25).



Hình 64.25: Danh sách các tools của Personal Database MCP Server

Lúc này, Claude sẽ xử lí message của chúng ta như sau:

- Đầu tiên, mô hình AI sẽ ra tín hiệu gọi tool `retrieve_document_from_database` trong quá trình xử lí message của chúng ta.
- Claude Desktop (AI application) xin quyền được gọi tool từ người dùng (xem Hình 64.26a). Trong popup xin quyền gọi tool, các arguments để gọi tool cũng được đính kèm.
- Sau khi được cấp quyền sử dụng tool, MCP server sẽ chạy tool và trả về các văn bản liên quan được đính kèm trong cuộc hội thoại (xem Hình 64.26b) để mô hình AI sinh câu trả lời cho người dùng.

(a) Claude Desktop xin quyền để gọi tool

(b) Kết quả gọi tool

Hình 64.26: Dùng tools trong Claude Desktop

Toàn bộ conversation (bao gồm arguments gọi tool và tool response) có thể được xem ở [đây](#).

## 64.4 Câu hỏi trắc nghiệm

### 64.4.1 Câu hỏi

1. Trong kiến trúc của MCP, thành phần nào chịu trách nhiệm điều phối và kiểm soát logic nghiệp vụ cấp cao?
  - (a) MCP Server.
  - (b) MCP Client.
  - (c) MCP Host.
  - (d) Transport Layer.
2. MCP client duy trì kết nối với MCP server theo mô hình nào?
  - (a) Nhiều-một.
  - (b) Một-nhiều.
  - (c) Một-một.
  - (d) Nhiều-nhiều.
3. Định dạng nào được MCP sử dụng để truyền thông điệp giữa client và server?
  - (a) gRPC.
  - (b) HTTP/2.
  - (c) JSON-RPC 2.0.
  - (d) SOAP.
4. Trong số các cơ chế truyền tải của MCP, cơ chế nào phù hợp cho môi trường local?
  - (a) WebSocket Transport.
  - (b) Streamable HTTP Transport.
  - (c) Server-Sent Events.
  - (d) Stdio Transport.
5. Vai trò chính của MCP server là gì?

- (a) Tổng hợp dữ liệu từ các MCP Client.
  - (b) Hiển thị giao diện người dùng.
  - (c) Cung cấp năng lực chuyên biệt thông qua giao thức chuẩn.
  - (d) Lưu trữ toàn bộ phiên làm việc của Host.
6. MCP server cung cấp các loại tài nguyên nào sau đây? (Chọn phương án đúng nhất)
- (a) Resources, Plugins, Queries.
  - (b) Documents, Tools, Commands.
  - (c) Files, Streams, Tasks.
  - (d) Resources, Tools, Prompts.
7. Tool trong MCP Server khác với Resource ở điểm nào?
- (a) Tools chỉ dành cho quản trị viên.
  - (b) Tools có thể thực thi và thay đổi trạng thái server.
  - (c) Resources được gọi tự động bởi LLM.
  - (d) Resources có thể bị sửa đổi bởi client.
8. Trong quá trình handshake giữa MCP Client và Server, điều gì xảy ra?
- (a) Truy vấn resources từ server.
  - (b) Gửi kết quả trả lời của LLM.
  - (c) Thương lượng các tính năng được hỗ trợ giữa hai bên.
  - (d) Tạo tệp log hệ thống.
9. Prompt trong MCP Server được dùng để:
- (a) Gửi thông điệp bảo mật giữa client và server.
  - (b) Làm mẫu hướng dẫn mô hình AI hoạt động theo ý định định sẵn.
  - (c) Tạo file output từ AI application.
  - (d) Giao tiếp giữa các MCP Servers.
10. Trong kiến trúc MCP, khi nào Host sẽ khởi tạo MCP Client mới?

- (a) Khi kết nối với mạng Internet.
- (b) Khi cần cập nhật mô hình AI.
- (c) Khi bắt đầu kết nối đến một MCP server.
- (d) Khi hệ thống gặp lỗi kết nối.

#### 64.4.2 Đáp án và giải thích

##### 1. Đáp án đúng: (C) MCP Host

*Giải thích:* MCP Host là thành phần chủ động khởi tạo kết nối tới MCP Server, kiểm soát quyền truy cập, quản lý client và điều phối toàn bộ logic nghiệp vụ cấp cao của ứng dụng AI.

##### 2. Đáp án đúng: (C) Một-một

*Giải thích:* Mỗi MCP client duy trì một kết nối 1:1 với một MCP server cụ thể để đảm bảo an toàn và không chia sẻ dữ liệu giữa các server khác nhau.

##### 3. Đáp án đúng: (C) JSON-RPC 2.0

*Giải thích:* MCP sử dụng định dạng JSON-RPC 2.0 để truyền và xử lý thông điệp giữa client và server vì tính đơn giản, tiêu chuẩn và khả năng hỗ trợ các yêu cầu có trạng thái.

##### 4. Đáp án đúng: (D) Stdio Transport

*Giải thích:* Stdio Transport sử dụng luồng chuẩn đầu vào/đầu ra để giao tiếp và không yêu cầu mạng, rất phù hợp với môi trường cục bộ như terminal.

##### 5. Đáp án đúng: (C) Cung cấp năng lực chuyên biệt thông qua giao thức chuẩn

*Giải thích:* MCP Server hoạt động như một adapter, đóng gói và công bố các năng lực chuyên biệt (như truy vấn, xử lý dữ liệu) thông qua một giao diện chuẩn.

##### 6. Đáp án đúng: (D) Resources, Tools, Prompts

*Giải thích:* Đây là ba thành phần cốt lõi (primitives) của MCP Server: Resource cung cấp dữ liệu, Tool là hành động có thể thực thi, Prompt là chỉ dẫn cho mô hình AI.

7. **Đáp án đúng: (B) Tools có thể thực thi và thay đổi trạng thái server**

*Giải thích:* Khác với resource (chỉ đọc), tool trong MCP server có khả năng thay đổi trạng thái hệ thống thông qua hành động như thêm, xoá hoặc cập nhật dữ liệu.

8. **Đáp án đúng: (C) Thương lượng các tính năng được hỗ trợ giữa hai bên**

*Giải thích:* Quá trình handshake giữa MCP client và MCP server giúp thống nhất về các tính năng như mã hoá, định dạng message, tool có sẵn,... trước khi giao tiếp chính thức bắt đầu.

9. **Đáp án đúng: (B) Làm mẫu hướng dẫn mô hình AI hoạt động theo ý định định sẵn**

*Giải thích:* Prompt là các template hoặc hướng dẫn định nghĩa sẵn nhằm định hình cách mô hình AI phản hồi, thường được sử dụng trong những tương tác có cấu trúc lặp lại.

10. **Đáp án đúng: (C) Khi cần truy cập một MCP Server mới với chức năng riêng biệt**

*Giải thích:* MCP Host sẽ tạo MCP client mới để kết nối đến mỗi MCP server riêng biệt, đảm bảo mỗi dịch vụ (như lưu trữ tệp, truy vấn dữ liệu) được tách biệt và an toàn.

# Chương 65

## Giao thức A2A

### 65.1 Giới thiệu



Hình 65.1: Giới thiệu A2A Protocol.

Sự hợp tác giữa các Agent độc lập, được xây dựng trên các nền tảng công nghệ khác nhau, là một thách thức lớn trong môi trường doanh nghiệp hiện đại. Hãy hình dung một quy trình làm việc phức tạp như sau:

- **Agent A** – Chatbot chăm sóc khách hàng, được xây dựng bằng *LangGraph*, phát hiện ra một sự cố kỹ thuật nghiêm trọng từ phía người dùng.
- Agent A cần chuyển tiếp sự cố này cho **Agent B** – một agent chẩn đoán chuyên biệt, được phát triển nội bộ bằng *CrewAI*. Agent B có khả năng phân tích log hệ thống, xác định nguyên nhân và đề xuất hướng xử lý.
- Sau khi phân tích, Agent B xác định rằng cần triển khai một bản vá phần mềm cụ thể để khắc phục lỗi.
- Cuối cùng, Agent B phải chuyển nhiệm vụ triển khai này cho **Agent C** – một agent bên thứ ba, được xây dựng bằng *LlamaIndex*, có chức năng truy cập và cập nhật trực tiếp vào môi trường triển khai của khách hàng.

Không có một tiêu chuẩn giao tiếp chung, việc tích hợp ba agent này là một quá trình tùy chỉnh phức tạp và thiếu tính ổn định. Các lập trình viên sẽ

phải tạo các lớp chuyển đổi riêng cho từng cặp agent. Bất kỳ thay đổi nào về thành phần agent hoặc mỗi lần cập nhật API đều có thể dẫn đến lỗi dây chuyền, ảnh hưởng đến tính ổn định của quy trình hợp tác liên agent. Kết quả là một hệ thống thiếu tính linh hoạt, dễ bị phân tán chức năng, gây cản trở cho việc mở rộng quy mô hoặc thích ứng với yêu cầu nghiệp vụ mới. Để hiện thực hóa khả năng hợp tác hiệu quả giữa các Agent, hệ thống cần một cơ chế chung cho phép các Agent có thể:

- **Khám phá năng lực (Capability Discovery):** Mỗi Agent phải công bố rõ các tác vụ mình hỗ trợ, định dạng dữ liệu chấp nhận và phương thức giao tiếp để các Agent khác có thể đánh giá và định tuyến phù hợp.
- **Thỏa thuận cách tương tác (UX Negotiation):** Các Agent cần thống nhất về hình thức trao đổi thông tin (văn bản, biểu mẫu, file, stream...) để đảm bảo hiểu đúng và phối hợp hiệu quả.
- **Quản lý tác vụ và trạng thái (Task and State Management):** Mỗi tác vụ phải được theo dõi rõ ràng trong suốt vòng đời (submitted → working → completed), kể cả khi qua nhiều Agent xử lý.
- **Hợp tác an toàn (Secure Collaboration):** Giao tiếp giữa các Agent phải đảm bảo xác thực, phân quyền và bảo mật, đặc biệt trong các hệ thống phân tán hoặc xuyên tổ chức.

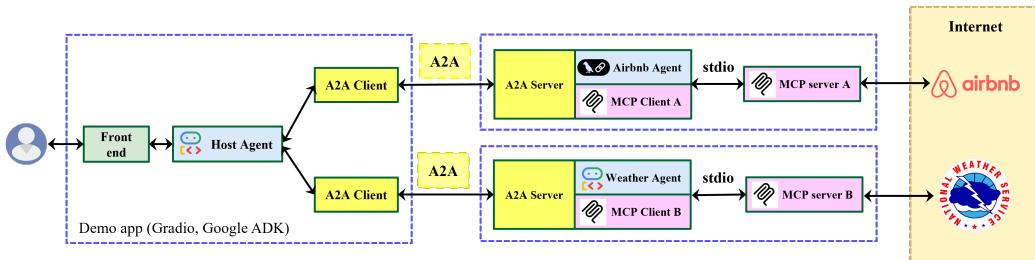
Trước thực trạng đó, cần một cách tiếp cận toàn diện để thiết lập một cơ chế giao tiếp chung, cho phép các Agent dì nền tảng có thể tương tác hiệu quả, linh hoạt và an toàn. Và giải pháp nổi bật đang được đề xuất ở đây chính là: **Agent-to-Agent (A2A) Protocol**.



Hình 65.2: Sự kết hợp của nhiều Agent là xu hướng tất yếu.

**Agent-to-Agent (A2A) Protocol** là một giao thức tiêu chuẩn, được thiết kế nhằm giải quyết toàn diện các thách thức trên trong môi trường AI đa agent. A2A định nghĩa một tập hợp quy tắc giao tiếp và định dạng thông điệp chuẩn hóa, đóng vai trò như lớp trừu tượng (abstraction layer) giữa các Agent – giúp che giấu các chi tiết về ngôn ngữ lập trình, framework, hoặc cách triển khai nội bộ của từng Agent cụ thể.

Nhờ đó, các Agent có thể liên kết, phối hợp và hoán đổi linh hoạt, bất kể sự khác biệt về công nghệ nền tảng. Việc này tăng cường khả năng tương tác liên nền tảng, thúc đẩy khả năng mở rộng, và tạo tiền đề cho sự phát triển của các hệ thống AI hợp tác phức tạp trong môi trường doanh nghiệp hiện đại.



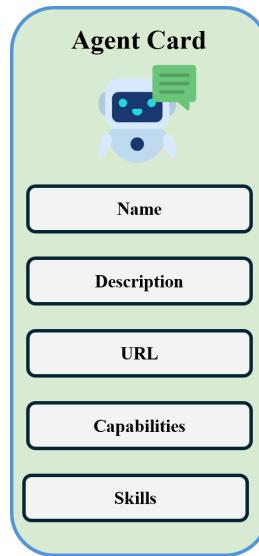
Hình 65.3: Demo sử dụng A2A và MCP.

## 65.2 Các khái niệm cốt lõi của A2A

### 65.2.1 Agent Card

Agent Card là một tệp JSON công khai đóng vai trò như "danh thiếp kỹ thuật số" của một Agent tuân thủ giao thức A2A. Nó chứa các thông tin mô tả về Agent và là một tệp JSON tiêu chuẩn, thường được lưu trữ tại đường dẫn (`/well-known/agent.json`)

**Mục đích:** Phục vụ cho việc khám phá Agent (Agent Discovery). Các client sẽ truy xuất tệp này để tìm hiểu năng lực và phương thức tương tác của agent.



Hình 65.4: Minh họa Agent Card đơn giản

#### Các trường chính:

- **name (bắt buộc):** Tên dễ hiểu, hiển thị của agent. Dùng để phân biệt trong giao diện hoặc log hệ thống.
- **description (tùy chọn):** Mô tả ngắn gọn vai trò hoặc chức năng chính của agent.

- **url (bắt buộc):** Endpoint HTTP(S) nơi agent lắng nghe và xử lý các lệnh A2A theo giao thức JSON-RPC. Thường là: <https://.../a2a>.
- **version (bắt buộc):** Phiên bản hiện tại của agent hoặc phiên bản chuẩn A2A mà agent tuân thủ (ví dụ: "1.0.0").
- **capabilities (bắt buộc):** Các năng lực giao thức mà agent hỗ trợ, bao gồm:
  - **streaming** (boolean, mặc định: `false`): Agent có hỗ trợ phản hồi dạng streaming qua `tasks/sendSubscribe` với SSE (Server-Sent Events) hay không.
  - **pushNotifications** (boolean, mặc định: `false`): Agent có khả năng gửi thông báo chủ động đến client qua webhook (khi client đã đăng ký trước) hay không.
  - **stateTransitionHistory** (boolean, mặc định: `false`): Agent có lưu và trả về lịch sử chuyển trạng thái của các Task hay không.
- **authentication (tùy chọn):** Mô tả các phương thức xác thực được agent yêu cầu trước khi cho phép tương tác:
  - **schemes (bắt buộc nếu có authentication):** Danh sách các phương thức xác thực được hỗ trợ như: "bearer", "apiKey", hoặc định dạng tùy chỉnh.
  - **credentials (tùy chọn):** Thông tin hoặc URL cung cấp gợi ý về cách lấy token/API key (không nên dùng cho thông tin nhạy cảm).
- **defaultInputModes / defaultOutputModes (tùy chọn):** Mô tả kiểu dữ liệu mà agent mặc định chấp nhận (đầu vào) hoặc tạo ra (đầu ra). Ví dụ: "text", "application/json", "image/png".
- **skills (bắt buộc):** Danh sách các kỹ năng hoặc chức năng mà agent cung cấp. Mỗi phần tử là một đối tượng gồm:
  - **id (bắt buộc):** Định danh duy nhất (string) cho skill — dùng trong giao tiếp hoặc định tuyến task.
  - **name (bắt buộc):** Tên dễ đọc cho skill (hiển thị với người dùng hoặc agent khác).

- **description (tùy chọn)**: Mô tả ngắn về mục đích của skill.
- **inputModes / outputModes (tùy chọn)**: Các loại dữ liệu mà skill chấp nhận đầu vào và tạo ra đầu ra — có thể khác với mặc định của agent.
- **examples (tùy chọn)**: Các ví dụ minh họa cho prompt hoặc use case cụ thể — giúp client hiểu và sử dụng đúng skill.

Ví dụ về cấu trúc Agent Card (mang tính minh họa):

```
1 {
2 "name": "Image Generation Agent",
3 "description": "Generates images based on text prompts.",
4 "url": "https://api.example-image-agent.com/a2a",
5 "version": "1.0.0",
6 "capabilities": {
7 "streaming": true,
8 "pushNotifications": false,
9 "stateTransitionHistory": true
10 },
11 "authentication": {
12 "schemes": ["apiKey"]
13 },
14 "defaultInputModes": ["text"],
15 "defaultOutputModes": ["image/png"],
16 "skills": [
17 {
18 "id": "generate_image",
19 "name": "Generate Image",
20 "description": "Creates an image from a textual description.",
21 "inputModes": ["text"],
22 "outputModes": ["image/png"],
23 "examples": ["Generate an image of a 'blue cat wearing a top
24 hat'"]
25 }
26]
}
```

### 65.2.2 Giao thức JSON-RPC 2.0

A2A sử dụng giao thức **JSON-RPC 2.0** làm nền tảng chuẩn để các Agent có thể giao tiếp một cách thống nhất, không phụ thuộc vào nền tảng. Giao thức này cho phép client gửi yêu cầu, theo dõi tiến trình, nhận phản hồi hoặc hủy bỏ các tác vụ (task) đang xử lý. Dưới đây là danh sách các phương thức JSON-RPC chính mà một A2A Server cần hỗ trợ:

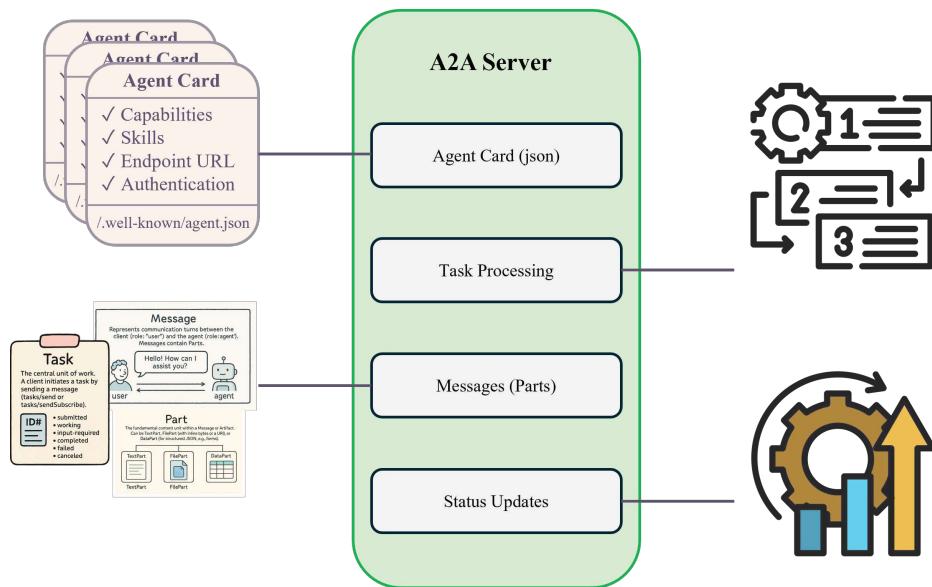
- **tasks/send**: Gửi một task mới và nhận phản hồi khi hoàn tất.
- **tasks/sendSubscribe**: Gửi task và nhận tiến trình xử lý theo thời gian thực qua SSE.
- **tasks/get**: Truy vấn trạng thái hiện tại của một task cụ thể.
- **tasks/cancel**: Hủy một task đang xử lý (nếu có thể).
- **tasks/pushNotification/set**: Cấu hình webhook để nhận thông báo chủ động từ agent.
- **tasks/pushNotification/get**: Xem lại webhook đã đăng ký.
- **tasks/resubscribe**: Khôi phục luồng SSE nếu bị ngắt kết nối.

Ví dụ minh họa cách sử dụng **tasks/send** trong thực tế:

```
1 { "jsonrpc": "2.0",
2 "method": "tasks/send",
3 "params": {
4 "taskId": "abc-123-task",
5 "message": {
6 "role": "user",
7 "parts": [
8 { "text": "Hãy tạo một hình ảnh con mèo đang cuộn rồng." }
9]
10 }
11 },
12 "id": "req-001"}
```

### 65.2.3 A2A Server

A2A Server là thành phần được triển khai kèm theo mỗi Agent chuyên biệt (ví dụ: LangGraph, CrewAI, LlamaIndex, ...), đảm nhiệm vai trò tiếp nhận và xử lý các tác vụ được gửi đến từ A2A Client thông qua giao thức **JSON-RPC**.



Hình 65.5: Các bước hoạt động của A2A Server trong hệ thống agent

A2A Server chịu trách nhiệm chính cho các hoạt động sau:

- Xử lý các phương thức JSON-RPC như:
  - `tasks/send`
  - `tasks/get`
  - `tasks/cancel`
- Quản lý vòng đời tác vụ (Task Lifecycle) từ:
  - `submitted` → `working` → `completed / failed`
- Gửi phản hồi đến client qua các cơ chế:

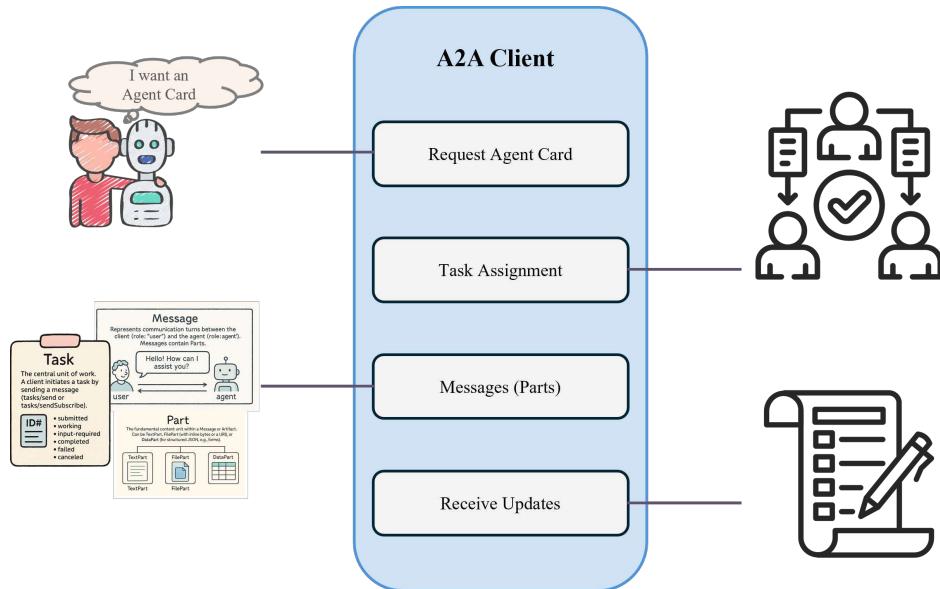
- HTTP response (phản hồi đồng bộ)
- SSE (Server-Sent Events) – để truyền dữ liệu theo thời gian thực
- Webhook – để gửi thông báo chủ động đến client
- Liên kết trực tiếp với logic xử lý nội bộ của Agent, ví dụ:
  - Phân tích log hệ thống
  - Tổng hợp dữ liệu
  - Triển khai bản vá hoặc cập nhật hệ thống

### 65.2.4 A2A Client

A2A Client là thành phần trung gian thực hiện điều phối tác vụ (task) đến các A2A Server đã đăng ký, thông qua giao thức chuẩn **JSON-RPC 2.0**.

Thực hiện:

- Gửi các lệnh JSON-RPC như: `tasks/send`, `tasks/sendSubscribe`.
- Nhận phản hồi từ Server qua:
  - HTTP response
  - SSE (stream kết quả)
  - Webhook (callback chủ động từ Agent)
- Tái kết nối hoặc đăng ký lại stream khi bị gián đoạn (`resubscribe`).
- Không cần biết công nghệ nội bộ của Agent – chỉ cần đọc Agent Card và điều phối Agent thông qua giao thức A2A.



Hình 65.6: Các bước hoạt động của A2A Client trong hệ thống agent

### 65.2.5 Task

Task là một yêu cầu công việc do client khởi tạo, được agent xử lý. Giao tiếp trong A2A luôn xoay quanh **Task**.

**Vòng đời Task (Task Lifecycle):**

- **submitted**: Client vừa gửi yêu cầu.
- **working**: Agent đang xử lý.
- **input-required**: Agent cần thêm thông tin từ client.
- **completed**: Xử lý thành công.
- **failed**: Lỗi khi xử lý.
- **canceled**: Client chủ động hủy.
- **unknown**: Không xác định.

**Thông tin chính trong một Task:**

- **id**: Mã định danh duy nhất (thường là UUID).
- **sessionId** (tuỳ chọn): Dùng để nhóm các task liên quan.
- **status**: Trạng thái hiện tại, kèm thời gian và message mới nhất.
- **artifacts** (tuỳ chọn): Kết quả tạo ra (file, dữ liệu, ảnh...).
- **history** (tuỳ chọn): Lưu lịch sử hội thoại theo lượt.
- **metadata** (tuỳ chọn): Thông tin phụ dạng key-value.

Ví dụ về Giao tiếp dựa trên Task – (Request):

```
1 {
2 "jsonrpc": "2.0",
3 "method": "tasks/send",
```

```
4 "params": {
5 "taskId": "20250615123456",
6 "message": {
7 "role": "user",
8 "parts": [
9 {
10 "text": "Find flights from New York to Miami on 2025-06-15"
11 }
12]
13 },
14 "id": "12345"
15 }
16 }
```

Ví dụ về Giao tiếp dựa trên Task – (Response)

```
1 {
2 "jsonrpc": "2.0",
3 "id": "12345",
4 "result": {
5 "taskId": "20250615123456",
6 "state": "completed",
7 "messages": [
8 {
9 "role": "user",
10 "parts": [
11 {
12 "text": "Find flights from New York to Miami on 2025-06-
13 15"
14 }
15]
16 },
17 {
18 "role": "agent",
19 "parts": [
20 {
21 "text": "I found the following flights from New York to
22 Miami
23 on June 15, 2025:\n\n1. Delta Airlines DL1234: Departs
24 JFK 08:00,..."
```

```
23]
24 }
25],
26 "artifacts": []
27 }
28 }
```

### 65.2.6 Message

Đại diện cho một lượt hội thoại trong quá trình xử lý một Task.

- **Cấu trúc (theo types.py):**

- **role** (enum: "user" hoặc "agent"): Xác định ai là người gửi.
- **parts** (bắt buộc): Danh sách các phần nội dung (xem mục Part bên dưới).
- **metadata** (tuỳ chọn): Thông tin bổ sung đi kèm thông điệp.

### 65.2.7 Part

Là đơn vị nội dung cơ bản trong một Message hoặc Artifact. Một Message có thể chứa nhiều Part với các kiểu khác nhau.

- **Các loại Part (theo types.py):**

- **TextPart**: Chứa văn bản thuần (**text**).
- **FilePart**: Đại diện cho tệp tin, có thể gồm:
  - \* **bytes**: Nội dung tệp mã hóa Base64 (dùng cho tệp nhỏ)
  - \* **uri**: Đường dẫn đến tệp (có thể kèm **name**, **mimeType**)
- **DataPart**: Dữ liệu JSON có cấu trúc (**data**), dùng cho biểu mẫu, kết quả dạng bảng, v.v.
- Mỗi Part có thể đính kèm **metadata** (tuỳ chọn).

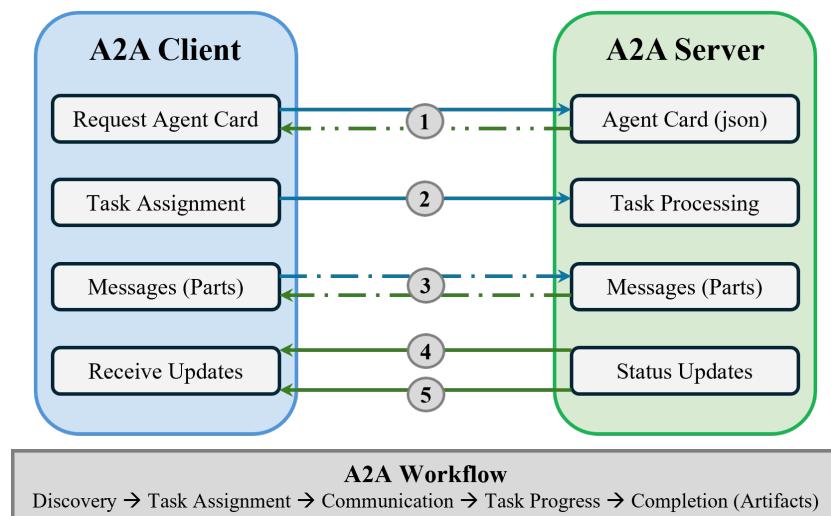
### 65.2.8 Artifact

Đại diện cho kết quả do agent tạo ra trong quá trình thực hiện task, khác biệt với hội thoại (Message). Ví dụ: mã nguồn, ảnh, tài liệu, hoặc dữ liệu có cấu trúc.

- **Cấu trúc (theo types.py):**

- `name / description` (tuỳ chọn): Tên và mô tả artifact
- `parts` (bắt buộc): Danh sách nội dung (giống như trong Message)
- `metadata` (tuỳ chọn): Thông tin đi kèm
- `index, append, lastChunk` (tuỳ chọn): Dùng khi streaming artifact lớn thành nhiều phần nhỏ

### 65.2.9 Mô hình tương tác giữa 2 Agent



Hình 65.7: Mô hình tương tác giữa 2 Agent.

Một phiên giao tiếp tiêu chuẩn giữa hai agent trong giao thức Agent2Agent (A2A) thường trải qua 5 giai đoạn chính sau:

1. **Khám phá năng lực (Discovery):**

- **Mục tiêu:** Giúp Client Agent hiểu được Remote Agent có thể làm gì trước khi gửi task.
- **Cách thức:** Client Agent thực hiện HTTP GET tới endpoint `/well-known/agent.json`.
- **Kết quả:** Nhận về *Agent Card* chứa thông tin định danh, năng lực tác vụ, endpoint API, định dạng dữ liệu hỗ trợ và cơ chế xác thực.

## 2. Giao nhiệm vụ (Task Assignment):

- **Mục tiêu:** Khởi tạo một task mới để Remote Agent xử lý.
- **Cách thức:** Client gửi yêu cầu thông qua phương thức `tasks/send` hoặc `tasks/sendSubscribe` (nếu cần nhận phản hồi theo dạng streaming).
- **Nội dung:** Task bao gồm tên tác vụ, dữ liệu đầu vào, mô tả yêu cầu và các cấu hình liên quan.
- **Chuẩn giao tiếp:** Giao tiếp tuân theo JSON-RPC 2.0.

## 3. Trao đổi dữ liệu (Communication):

- **Mục tiêu:** Hỗ trợ truyền tải dữ liệu bổ sung hoặc tương tác hai chiều trong quá trình xử lý task.
- **Cách thức:** Gửi các phần dữ liệu nhỏ (gọi là *Message Part*) qua lại giữa hai Agent.
- **Dữ liệu hỗ trợ:** văn bản, biểu mẫu có cấu trúc (form), file đính kèm, URI tham chiếu...

## 4. Cập nhật tiến trình (Task Progress):

- **Mục tiêu:** Cho phép Client theo dõi trạng thái của task theo thời gian thực.
- **Phương pháp:** Remote Agent gửi các cập nhật định kỳ hoặc theo sự kiện về trạng thái tác vụ (`submitted`, `working`, `completed`, `failed`).
- **Cơ chế truyền:** Thông qua SSE (Server-Sent Events) hoặc webhook (push).

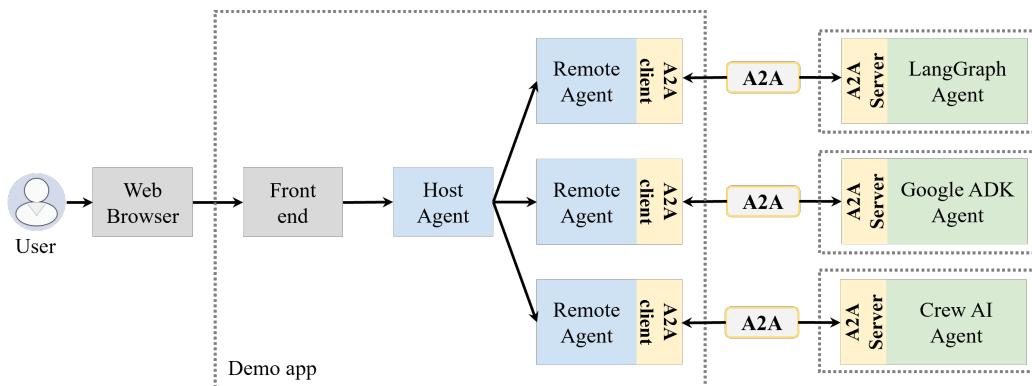
### 5. Hoàn tất và trả kết quả (Completion):

- **Mục tiêu:** Cung cấp đầu ra cuối cùng cho Client sau khi task hoàn thành.
- **Dữ liệu trả về:** Có thể là văn bản, tệp, dữ liệu JSON hoặc URI đến artifact.
- **Cách truyền:** Gửi kèm theo cập nhật trạng thái cuối, hoặc thông qua cơ chế pull (nếu chỉ định URI).

Luồng này phản ánh đầy đủ vòng đời của một tác vụ trong hệ sinh thái A2A, theo trình tự:

*Discovery → Task Assignment → Communication → Task Progress → Completion*

#### 65.2.10 Mô hình phối hợp nhiều Agent



Hình 65.8: Mô hình phối hợp nhiều Agent.

Hình trên minh họa kiến trúc một hệ thống đa agent (multi-agent system) sử dụng giao thức A2A để giao tiếp và điều phối tác vụ giữa các Agent khác nhau. Quá trình này có thể chia thành hai quy trình chính:

Quy trình 1: Khởi tạo và khám phá Agent chuyên biệt

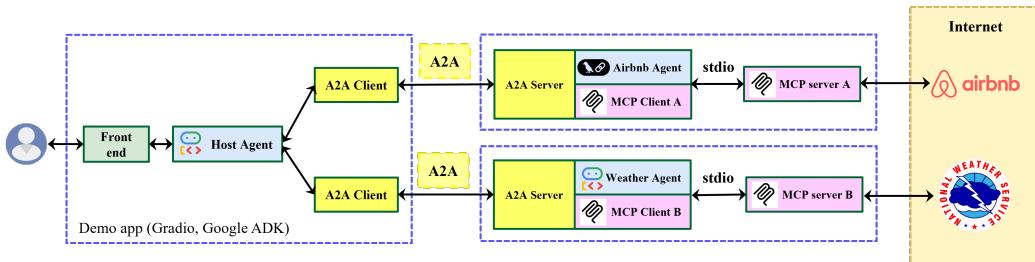
- Mỗi Agent chuyên biệt (được xây dựng trên LangGraph, CrewAI, Google ADK, v.v.) hoạt động như một **A2A Server**, và công bố năng lực của mình thông qua `/well-known/agent.json` (Agent Card).
- Host Agent chứa các **A2A Client**, và các client này chịu trách nhiệm kết nối đến các A2A Server của các agent chuyên biệt (như Weather Agent, Airbnb Agent).
- A2A Client sẽ truy xuất Agent Card để biết các khả năng, endpoint và định dạng giao tiếp mà Agent Server hỗ trợ.
- Kết quả: Hệ thống sẵn sàng phối hợp với nhiều Agent không đồng nhất, mà không cần viết tay từng lớp tích hợp riêng biệt.

Quy trình 2: Từ yêu cầu người dùng đến phản hồi cuối cùng

- Người dùng nhập yêu cầu từ giao diện web, yêu cầu này được gửi về Host Agent.
- Host Agent phân tích yêu cầu và chọn Remote Agent phù hợp, dựa trên thông tin đã khám phá từ Agent Card.
- Remote Agent gửi task đến Agent Server qua các phương thức như `tasks/send` hoặc `tasks/sendSubscribe`.
- Trong quá trình xử lý, Agent Server sẽ cập nhật tiến trình và kết quả qua luồng SSE hoặc webhook.
- Host Agent tổng hợp dữ liệu đầu ra và gửi phản hồi cuối cùng về frontend cho người dùng.

**Lưu ý:** Khái niệm “Remote Agent” trong một số tài liệu trình bày nằm ở phía client (đại diện cho các agent đã kết nối trong một host Agent), trong khi tài liệu khác đặt nó phía server. Tuy nhiên, bản chất kiến trúc của A2A là sự tương tác giữa hai vai trò: **A2A Client** và **A2A Server** dù cho Remote Agent nằm ở vị trí nào.

### 65.3 Xây dựng mô hình áp dụng A2A và MCP



Hình 65.9: Mô hình áp dụng A2A và MCP

#### 65.3.1 Tổng quan về mô hình

Hệ thống được thiết kế theo kiến trúc **đa Agent phân tán** (distributed multi-agent), trong đó các thành phần Agent tương tác thông qua **A2A Protocol** dựa trên chuẩn JSON-RPC over HTTP, cho phép phân tách rõ ràng giữa điều phối và xử lý tác vụ. Trung tâm hệ thống là **Routing Agent**

**(Host Agent)**, chịu trách nhiệm tiếp nhận truy vấn từ người dùng (qua giao diện Gradio), phân tích mục đích truy vấn và định tuyến tác vụ đến các Remote Agent chuyên biệt tương ứng.

Routing Agent không trực tiếp thực hiện truy vấn, mà chỉ đóng vai trò điều phối luồng tác vụ. Sau khi phân tích nội dung câu hỏi (bằng mô hình ngôn ngữ - LLM), agent này khởi tạo một message task và gửi đến Remote Agent tương ứng thông qua **A2A Client**. Việc gửi nhận được thực hiện qua endpoint HTTP do Remote Agent cung cấp.

**Weather Agent** là một Remote Agent chuyên xử lý các truy vấn liên quan đến điều kiện khí tượng. Khi nhận một truy vấn qua **A2A Server**, tác tử này đưa truy vấn vào mô hình ngôn ngữ để phân tích ngữ nghĩa và xác định các tham số cần thiết (ví dụ: tên thành phố, khung thời gian). Dựa trên đó, mô hình ra quyết định gọi công cụ từ MCP toolset. Công cụ này sẽ thực hiện một API call thực tế đến dịch vụ weather.gov (hệ thống thời tiết quốc gia Hoa Kỳ) – thông qua giao thức HTTP. Dữ liệu trả về dạng JSON sẽ được phân tích, tóm tắt và gửi ngược lại Host Agent thông qua A2A. (Lưu ý: do

giới hạn API, Weather Agent hiện chỉ hỗ trợ địa điểm trong lãnh thổ Hoa Kỳ.)

**Airbnb Agent** xử lý các truy vấn liên quan đến chỗ ở – chẳng hạn: "Tìm homestay ở Đà Lạt tuần này." Sau khi nhận truy vấn từ Host Agent thông qua A2A, tác tử này sử dụng mô hình ngôn ngữ để phân tích yêu cầu, trích xuất các tham số như vị trí địa lý, thời gian lưu trú, loại hình chỗ ở, v.v. Sau đó, agent gọi công cụ phù hợp từ MCP toolset. Tool này truy xuất dữ liệu từ dịch vụ Airbnb thông qua API chính thức. Kết quả được phân tích, lọc theo tiêu chí phù hợp và phản hồi lại Host Agent thông qua giao thức A2A.

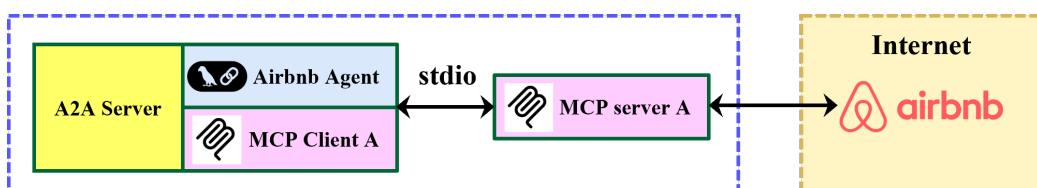
Mỗi Remote Agent được tổ chức như một đơn vị độc lập: có **A2A Server** riêng, tool xử lý riêng, và phiên làm việc ngữ nghĩa (semantic session) riêng. A2A Protocol đóng vai trò **cầu nối** giữa các thực thể độc lập này, đảm bảo sự hợp tác liên-agent mà không cần chia sẻ nội bộ về logic thực thi hay tài nguyên.

Cách tiếp cận này giúp hệ thống có **tính mở rộng cao**: việc bổ sung một agent mới chỉ cần định nghĩa AgentCard và endpoint A2A tương ứng mà không làm ảnh hưởng đến các thành phần khác trong hệ thống.

### 65.3.2 Xây dựng các A2A Server

#### Phần 1: Build Agent Airbnb (airbnb\_agent/airbnb\_agent.py)

**Mục tiêu:** Minh họa quá trình tạo ra một agent có khả năng nhận yêu cầu từ người dùng, sử dụng model ngôn ngữ (LLM) + toolset (từ MCP) để phản hồi theo định dạng chuẩn của A2A.



Hình 65.10: A2A Server - Airbnb Agent

### Import các thành phần chính:

```

1 from langchain_core.runnables.config import RunnableConfig
2 from langchain_core.messages import AIMessage, AIMessageChunk
3 from langgraph.checkpoint.memory import MemorySaver
4 from langgraph.prebuilt import create_react_agent
5 from pydantic import BaseModel
6 from langchain_google_genai import ChatGoogleGenerativeAI
7 import logging, os, httpx

```

### Giải thích các thành phần:

- **langgraph**: Thư viện giúp xây dựng agent có khả năng tương tác theo mô hình *React Agent*, tức là mô hình sử dụng công cụ (tool-augmented reasoning).
- **ChatGoogleGenerativeAI**: Giao diện kết nối đến mô hình ngôn ngữ của Google (Generative AI API) để sinh phản hồi ngôn ngữ tự nhiên.
- **MemorySaver**: Dùng để lưu trạng thái nội bộ (memory) trong quá trình tương tác – đặc biệt hữu ích khi duy trì phiên làm việc theo luồng (session-based) trong hệ thống A2A.
- **RunnableConfig**: Cho phép cấu hình tác vụ theo `thread_id`, giúp agent duy trì đúng ngữ cảnh của hội thoại khi xử lý yêu cầu.
- **AIMessageChunk**: Cho phép phân đoạn kết quả phản hồi, rất hữu ích trong chế độ phản hồi dạng *streaming* (truyền kết quả dần dần về phía người dùng).
- **httpx, logging, os**: Các thư viện hỗ trợ thường dùng cho logging, gọi HTTP và xử lý môi trường hệ thống.

### Định nghĩa format phản hồi:

```

1 class ResponseFormat(BaseModel):
2 status: Literal['input_required', 'completed', 'error'] = 'input_required'

```

```
3 | message: str
```

### Ý nghĩa trường status trong A2A:

- completed: Tác vụ đã hoàn tất thành công. A2A sẽ đóng session.
- input\_required: Agent cần thêm thông tin từ người dùng để tiếp tục.
- error: Đã xảy ra lỗi, A2A sẽ hiển thị lỗi cho người dùng.

### Lớp AirbnbAgent và hàm \_\_init\_\_:

```
1 class AirbnbAgent:
2 SYSTEM_INSTRUCTION = """You are a specialized assistant..."""
3 RESPONSE_FORMAT_INSTRUCTION = 'Select status as "completed"...'
4 SUPPORTED_CONTENT_TYPES = ['text', 'text/plain']
5
6 def __init__(self, mcp_tools: list[Any]):
7 model_name = os.getenv('GOOGLE_GENAI_MODEL')
8 self.model = ChatGoogleGenerativeAI(model=model_name)
9 self.mcp_tools = mcp_tools
```

### Giải thích:

- SYSTEM\_INSTRUCTION: chỉ dẫn hệ thống để mô hình phản hồi đúng ngữ cảnh – là một yêu cầu khi làm việc với giao thức A2A.
- RESPONSE\_FORMAT\_INSTRUCTION: hướng dẫn định dạng phản hồi – giúp model trả kết quả chuẩn JSON để A2A phân tích.
- SUPPORTED\_CONTENT\_TYPES: xác định các kiểu dữ liệu đầu vào được hỗ trợ.
- Trong \_\_init\_\_:
  - mcp\_tools: danh sách các công cụ lấy từ MCP (thường là Tool object).

- `self.model`: khởi tạo mô hình ngôn ngữ Google GenAI.
- Nếu không truyền `mcp_tools`, agent sẽ không thể xử lý tác vụ do không có công cụ đi kèm.

**Phương thức `ainvoke()`: khởi chạy Agent và nhận kết quả xử lý không-stream**

```

1 async def ainvoke(self, query: str, session_id: str) -> dict[str,
2 Any]:
3 airbnb_agent_runnable = create_react_agent(
4 self.model,
5 tools=self.mcp_tools,
6 checkpointer=memory,
7 prompt=self.SYSTEM_INSTRUCTION,
8 response_format=(self.RESPONSE_FORMAT_INSTRUCTION,
9 ResponseFormat),
10)
11
12 config = {'configurable': {'thread_id': session_id}}
13 langgraph_input = {'messages': [('user', query)]}
14
15 await airbnb_agent_runnable.ainvoke(langgraph_input, config)
16
17 return self._get_agent_response_from_state(config,
18 airbnb_agent_runnable)

```

**Giải thích:**

- Tạo Agent theo kiến trúc ReAct sử dụng:
  - `self.model`: mô hình Google GenAI.
  - `self.mcp_tools`: các công cụ lấy từ MCP.
  - `checkpointer=memory`: lưu ngữ cảnh cuộc hội thoại theo session.
  - `response_format`: yêu cầu định dạng đầu ra khớp với `ResponseFormat`.
- `session_id` được truyền vào như `thread_id` – tương ứng với `contextId` trong A2A – giúp giữ ngữ cảnh lâu dài.
- `langgraph_input` chứa thông điệp người dùng gửi.

- Kết quả cuối cùng được truy xuất qua hàm `_get_agent_response_from_state()`.

**Hàm `_get_agent_response_from_state()`:** Trích xuất kết quả cuối từ Agent

```

1 def _get_agent_response_from_state(self, config, agent_runnable) ->
2 dict:
3 current_state = agent_runnable.get_state(config)
4 structured_response = current_state.values.get('
5 structured_response')

```

Ý nghĩa:

- Hàm này lấy trạng thái hiện tại của Agent thông qua phương thức `get_state()`.
- Trạng thái Agent chứa thông tin đầu ra dưới dạng `structured_response` – đây là kết quả chính thức được A2A sử dụng.
- Nếu không có trường `structured_response`, A2A sẽ fallback sang message cuối cùng (nếu có).

Kết quả trả về chuẩn A2A:

```

1 {
2 "is_task_complete": True/False,
3 "require_user_input": True/False,
4 "content": "Kết quả trả lời"
5 }

```

**Lưu ý:** Đây là định dạng tối ưu để A2A xác định trạng thái task (hoàn tất hay chờ phản hồi tiếp), và hiển thị nội dung cho người dùng.

**Hàm `stream()`:** Phản hồi theo thời gian thực

```
1 async def stream(self, query: str, session_id: str) -> AsyncIterable
2 [Any]:
3 agent_runnable = create_react_agent(
4 self.model,
5 tools=self.mcp_tools,
6 checkpointer=memory,
7 prompt=self.SYSTEM_INSTRUCTION,
8 response_format=(self.RESPONSE_FORMAT_INSTRUCTION,
9 ResponseFormat),
10)
11 config = {'configurable': {'thread_id': session_id}}
12 langgraph_input = {'messages': [('user', query)]}
13
14 async for chunk in agent_runnable.astream_events(langgraph_input
15 , config, version='v1'):
16 event_name = chunk.get('event')
17 data = chunk.get('data', {})
18 content_to_yield = None
19
20 if event_name == 'on_tool_start':
21 content_to_yield = f"Using tool: {data.get('name', 'a
22 tool')}..."
23 elif event_name == 'on_chat_model_stream':
24 message_chunk = data.get('chunk')
25 if isinstance(message_chunk, AIMessageChunk) and
26 message_chunk.content:
27 content_to_yield = message_chunk.content
28
29 if content_to_yield:
30 yield {
31 'is_task_complete': False,
32 'require_user_input': False,
33 'content': content_to_yield,
34 }
35
36 final_response = self._get_agent_response_from_state(config,
37 agent_runnable)
38 yield final_response
```

### Phân tích:

- Hàm `stream` thực hiện tạo một `react agent` từ `model` và `toolset` có sẵn.

- Biến config định danh phiên làm việc với `thread_id` là `session_id`.
- Biến `langgraph_input` chứa nội dung hội thoại ban đầu từ người dùng.
- Với mỗi sự kiện trong `astream_events`, agent gửi ra từng phần nhỏ (`chunk`):
  - Nếu `event == 'on_tool_start'` → báo tool đang được dùng.
  - Nếu `event == 'on_chat_model_stream'` → lấy nội dung dòng ra từ mô hình.
- Mỗi phần sẽ được yield dưới dạng một dict chuẩn A2A:

```

1 {
2 'is_task_complete': False,
3 'require_user_input': False,
4 'content': 'Nội dung phản hồi từng phần'
5 }
```

- Sau khi stream xong, agent lấy trạng thái cuối bằng `_get_agent_response_from_state()` và gửi chunk cuối cùng.

### Xử lý lỗi:

- Nếu có lỗi xảy ra trong quá trình stream, trả về phản hồi dạng:

```

1 {
2 'is_task_complete': True,
3 'require_user_input': False,
4 'content': 'An error occurred during streaming: ...'
5 }
```

## Phân 2: Build AirbnbAgentExecutor (`airbnb_agent/agent_executor.py`)

**Mục tiêu:** Xây dựng lớp `AirbnbAgentExecutor` – thành phần trung gian giữa A2A Server và Agent. Nó thực hiện các nhiệm vụ chính sau:

- Nhận yêu cầu từ phía A2A Server ( thông qua `DefaultRequestHandler`).
- Gọi hàm `agent.stream(...)` để xử lý câu hỏi người dùng.
- Đẩy kết quả phản hồi về `EventQueue` theo định dạng A2A Event, gồm:
  - `TaskStatusUpdateEvent`: Thay đổi trạng thái (working, completed, ...).
  - `TaskArtifactUpdateEvent`: Gửi nội dung phản hồi (artifact) đến client.

### Import các thành phần chính:

```
1 from a2a.server.agent_execution import AgentExecutor, RequestContext
2 from a2a.server.events.event_queue import EventQueue
3 from a2a.types import (
4 TaskArtifactUpdateEvent,
5 TaskStatusUpdateEvent,
6 TaskState,
7 TaskStatus,
8)
9 from a2a.utils import (
10 new_agent_text_message,
11 new_task,
12 new_text_artifact,
13)
```

### Giải thích các thành phần:

- `AgentExecutor`: Lớp nền (base class) cần kế thừa và tuỳ biến để xử lý luồng tác vụ (task).
- `RequestContext`: Đóng gói thông tin từ client gồm `message`, `taskId`, `contextId`.
- `EventQueue`: Hàng đợi nội bộ dùng để gửi các phản hồi (event) từ agent về lại phía A2A Server.
- `TaskStatusUpdateEvent`: Sự kiện cập nhật trạng thái task như `submitted`, `working`, `completed`, `failed`.

- **TaskArtifactUpdateEvent**: Gửi dữ liệu trả lời của agent về client, dưới dạng artifact.
- **new\_task, new\_text\_artifact, new\_agent\_text\_message**: Hàm tiện ích giúp tạo nhanh các đối tượng đúng định dạng cho A2A.

### Lớp AirbnbAgentExecutor

**AirbnbAgentExecutor** là lớp kế thừa từ **AgentExecutor**, dùng để triển khai cách xử lý yêu cầu người dùng trong hệ thống A2A. Đây là cầu nối giữa A2A server và lớp logic xử lý agent.

#### Định nghĩa lớp và hàm khởi tạo:

```
1 class AirbnbAgentExecutor(AgentExecutor):
2 def __init__(self, mcp_tools: list[Any]):
3 self.agent = AirbnbAgent(mcp_tools=mcp_tools)
```

#### Giải thích:

- **AirbnbAgentExecutor**: Subclass của **AgentExecutor**, bắt buộc override lại các hàm như **execute()** hoặc **cancel()** để tùy biến luồng hoạt động.
- **mcp\_tools**: Là danh sách công cụ đã được khởi tạo từ các MCP Server. Danh sách này được truyền vào từ phần **main.py**.
- **self.agent**: Khởi tạo một instance của lớp **AirbnbAgent**, lớp này chịu trách nhiệm chính trong việc gọi mô hình LLM và sử dụng công cụ để phản hồi cho người dùng.

#### Hàm **execute(...)**: Xử lý tác vụ từ phía A2A Server

Đây là phương thức chính để xử lý một task đến từ A2A. Nó nhận đầu vào, gọi agent xử lý bằng **stream()**, và phản hồi kết quả qua **EventQueue**.

#### Định nghĩa hàm:

```

1 @override
2 async def execute(
3 self, context: RequestContext, event_queue: EventQueue
4) -> None:

```

### Ý nghĩa các tham số:

- **context**: Gói thông tin đầu vào, bao gồm **message**, **taskId**, **contextId**, ...
- **event\_queue**: Hàng đợi phản hồi, dùng để gửi các event về A2A server.

### Bước 1: Lấy câu hỏi người dùng và task hiện tại

```

1 query = context.get_user_input()
2 task = context.current_task

```

- **query**: Câu hỏi từ người dùng (VD: "Find Airbnb in Paris").
- **task**: Nếu đã tồn tại → tái sử dụng, nếu không → tạo mới ở bước sau.

### Bước 2: Tạo mới task nếu chưa có

```

1 if not task:
2 task = new_task(context.message)
3 await event_queue.enqueue_event(task)

```

- **new\_task(...)**: Tạo đối tượng task từ message gốc.
- **enqueue\_event(task)**: Gửi sự kiện tạo task về server A2A.

### Bước 3: Gọi agent xử lý bằng chế độ streaming

```
1 async for event in self.agent.stream(query, task.contextId):
```

- Gọi đến hàm `stream(...)` trong `AirbnbAgent`.
- Trả về các dict phản hồi như:
  - ‘`is_task_complete`’: Task đã hoàn tất chưa?
  - ‘`require_user_input`’: Có cần người dùng nhập thêm?
  - ‘`content`’: Nội dung phản hồi.

#### Bước 4: Gửi phản hồi về A2A theo trạng thái task

- ‘`is_task_complete`’ = `True` → gửi:
  - `TaskArtifactUpdateEvent`
  - `TaskStatusUpdateEvent(state=completed)`
- ‘`require_user_input`’ = `True` → gửi:
  - `TaskStatusUpdateEvent(state=input_required)`
- Ngược lại → phản hồi tạm thời:
  - `TaskStatusUpdateEvent(state=working)`

#### Ví dụ gửi yêu cầu nhập thêm từ user:

```
1 await event_queue.enqueue_event(
2 TaskStatusUpdateEvent(
3 status=TaskStatus(state=TaskState.input_required),
4 message=new_agent_text_message(event['content'], task.
5 agentMessageId)
6)
```

Hàm `cancel(...)`: Không hỗ trợ hủy tác vụ giữa chừng

```

1 @override
2 async def cancel(self, context, event_queue):
3 raise Exception('cancel not supported')

```

### Ý nghĩa:

- Hàm `cancel(...)` được override để xử lý logic khi phía A2A server yêu cầu huỷ một tác vụ đang thực hiện.
- Trong trường hợp này, agent chưa hỗ trợ huỷ giữa chừng nên luôn `raise Exception`.

### Phần 3: Build Airbnb A2A Server (airbnb\_agent/\_\_main\_\_.py)

**Mục tiêu:** Khởi tạo và chạy một server tuân theo chuẩn A2A. Server này chịu trách nhiệm:

- Khởi tạo `AgentExecutor` có kết nối tool từ MCP.
- Xây dựng A2A app với lifecycle đầy đủ.
- Đăng ký metadata agent (`AgentCard`) để UI A2A hiểu và tương tác đúng.

### Import các thành phần chính:

```

1 from a2a.server.apps import A2AStarletteApplication
2 from a2a.server.request_handlers import DefaultRequestHandler
3 from a2a.server.tasks import InMemoryTaskStore
4 from a2a.types import AgentCapabilities, AgentCard, AgentSkill
5 from agents.airbnb_planner_multiagent.airbnb_agent.agent_executor
6 import AirbnbAgentExecutor
7 from agents.airbnb_planner_multiagent.airbnb_agent.airbnb_agent
8 import AirbnbAgent
9 from langchain_mcp_adapters.client import MultiServerMCPClient

```

### Giải thích các thành phần:

- **A2AStarletteApplication**: Tạo ứng dụng ASGI tương thích với A2A Platform.
- **DefaultRequestHandler**: Nhận yêu cầu từ client và gọi đến AgentExecutor để xử lý.
- **AgentCard, AgentCapabilities, AgentSkill**: Metadata mô tả agent – A2A sẽ sử dụng để hiển thị giao diện.
- **AirbnbAgentExecutor**: Cầu nối giữa A2A server và logic agent.
- **MultiServerMCPClient**: Client dùng để kết nối và tải tools từ các MCP server.

### Hàm `app_lifespan(context)`:

```

1 @asynccontextmanager
2 async def app_lifespan(context: dict[str, Any]):
3 mcp_client_instance = MultiServerMCPClient(SERVER_CONFIGS)
4 mcp_tools = await mcp_client_instance.get_tools()
5 context['mcp_tools'] = mcp_tools
6
7 yield # Cho app tiếp tục chạy
8
9 await mcp_client_instance.__aexit__(None, None, None)
10 context.clear()

```

### Giải thích:

- Đây là một lifecycle hook.
- MCP client và tools được khởi tạo một lần duy nhất khi app bắt đầu.
- Dọn dẹp khi app tắt (gọi `__aexit__` và `context.clear()`).

### Hàm `get_agent_card(...)`:

```

1 def get_agent_card(host: str, port: int):
2 return AgentCard(
3 name='Airbnb Agent',
4 description='Helps with searching accommodation',
5 url=f'http://[{host}]:{port}/',
6 version='1.0.0',
7 defaultInputModes=AirbnbAgent.SUPPORTED_CONTENT_TYPES,
8 defaultOutputModes=AirbnbAgent.SUPPORTED_CONTENT_TYPES,
9 capabilities=AgentCapabilities(streaming=True,
10 pushNotifications=True),
11 skills=[
12 AgentSkill(
13 id='airbnb_search',
14 name='Search Airbnb',
15 description='Find places to stay',
16 tags=['travel'],
17 examples=['airbnb in Paris']
18)
19]
)

```

### Giải thích:

- Định nghĩa metadata agent để gửi về A2A server.
- Dùng trong hàm khởi tạo A2AStarletteApplication(...).
- Cho phép A2A UI hiển thị agent và tương tác đúng cách.

### Hàm main(...) – Khởi động A2A Server

**Mục tiêu:** Hàm main thực thi server theo chuẩn A2A, thông qua việc gọi run\_server\_async().

```

1 def main(host='localhost', port=10002, log_level='info'):
2 asyncio.run(run_server_async(host, port, log_level))

```

### Bên trong run\_server\_async(...):

Gọi `app_lifespan(...)` để khởi tạo tool:

```
1 async with app_lifespan(app_context):
2 mcp_tools = app_context.get('mcp_tools', [])
```

- Load tool từ MCP server một lần.
- Lưu vào biến toàn cục `app_context`.

Khởi tạo `AirbnbAgentExecutor`:

```
1 airbnb_agent_executor = AirbnbAgentExecutor(mcp_tools=mcp_tools)
```

- Giao tiếp giữa server và agent thông qua hàm `execute()`.

Tạo `RequestHandler`:

```
1 request_handler = DefaultRequestHandler(
2 agent_executor=airbnb_agent_executor,
3 task_store=InMemoryTaskStore(),
4)
```

- Trung gian giữa client và executor.
- Sử dụng task store nội bộ trong RAM.

Tạo A2A ASGI App:

```
1 a2a_server = A2AStarletteApplication(
2 agent_card=get_agent_card(host, port),
3 http_handler=request_handler,
4)
5 asgi_app = a2a_server.build()
```

- App này tuân theo chuẩn A2A để tương tác với UI.

Chạy bằng `uvicorn.Server(...)`

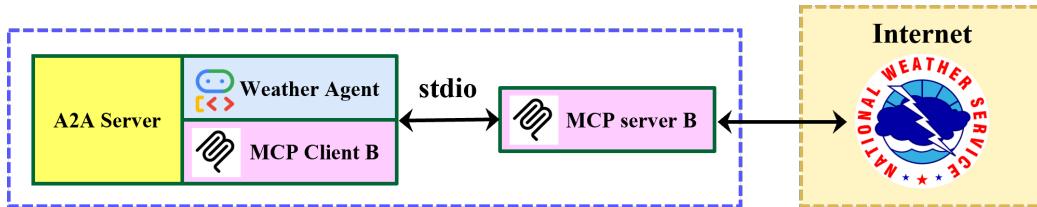
```

1 config = uvicorn.Config(
2 app=asgi_app,
3 host=host,
4 port=port,
5 log_level=log_level.lower(),
6 lifespan='auto',
7)
8 await uvicorn.Server(config).serve()

```

- Dùng Uvicorn để chạy app ASGI.
- Lifecycle hook (`lifespan='auto'`) cho phép tự động chạy `app_lifespan`.

#### Phần 4: Build Weather A2A server (weather\_agent)



Hình 65.11: A2A Server - Weather Agent

Tương tự quá trình xây dựng **Airbnb A2A Server**, một A2A server thứ hai được khởi tạo cho **weather\_agent**. Đây là một agent dựa trên mô hình ngôn ngữ lớn (LLM-based agent), được phát triển với bộ công cụ **Google ADK SDK**, có khả năng tương tác với người dùng nhằm truy vấn và cung cấp thông tin thời tiết theo thời gian thực. Quá trình khởi tạo tác tử này tuân thủ đầy đủ kiến trúc tích hợp theo chuẩn **Agent-to-Agent (A2A)**.

### Các thành phần chính:

- **Lớp khởi tạo:** LlmAgent – cung cấp kiến trúc agent tích hợp mô hình ngôn ngữ và công cụ thực thi.
- **Mô hình ngôn ngữ:** gemini-2.5-flash – tối ưu hóa cho tốc độ xử lý và chi phí inferencing.
- **Tập công cụ (toolset):** sử dụng MCPToolset, giao tiếp trực tiếp với chương trình weather\_mcp.py thông qua giao thức stdio.
- **Giao tiếp tác vụ:** tác tử sử dụng session để duy trì ngữ cảnh hội thoại, các phản hồi được phát ra theo định dạng chuẩn A2A gồm TaskStatusUpdateEvent, TaskArtifactUpdateEvent.

### Cấu hình giao diện và cổng:

- **Cổng mặc định:** 10001
- **Chuẩn giao diện:** A2A-ASGI, sử dụng lớp A2AStarletteApplication
- **Tính năng hỗ trợ:** Streaming nội dung phản hồi, push notification, và khả năng tương thích đầy đủ với nền tảng giao diện người dùng của hệ A2A.

#### 65.3.3 Xây dựng Host Agent

Phần 1: Build Remote Agent Connection(`host_agent/remote_agent_connection.py`)

**Mục tiêu:** Minh họa cách thiết lập kết nối từ xa (*remote connection*) đến các agent con trong kiến trúc Agent-to-Agent (A2A). File này đóng vai trò trung gian giữa host agent và các agent khác (ví dụ: `weather_agent`, `airbnb_agent`) thông qua giao thức A2A tiêu chuẩn.

Import các thành phần chính:

```
1 from collections.abc import Callable
2 import httpx
3
4 from a2a.client import A2AClient
5 from a2a.types import (
6 AgentCard,
7 SendMessageRequest,
8 SendMessageResponse,
9 Task,
10 TaskArtifactUpdateEvent,
11 TaskStatusUpdateEvent,
12)
```

### Giải thích các thành phần:

- **Callable** (`collections.abc`): Kiểu dữ liệu đại diện cho một đối tượng có thể gọi được (callable object), thường dùng để khai báo kiểu hàm hoặc callback.
- **httpx.AsyncClient**: HTTP client bất đồng bộ, dùng để gửi yêu cầu tới các agent từ xa.
- **A2AClient**: Lớp client thuộc SDK A2A, giúp chuẩn hóa giao tiếp với agent qua `AgentCard`.
- **AgentCard**: Thông tin định danh và mô tả của agent từ xa (gồm tên, mô hình, kỹ năng...).
- **SendMessageRequest / SendMessageResponse**: Các đối tượng dữ liệu A2A mô tả yêu cầu và phản hồi tác vụ.
- **Task, TaskArtifactUpdateEvent, TaskStatusUpdateEvent**: Các sự kiện A2A mô tả kết quả hoặc trạng thái của tác vụ.

Định nghĩa các type tiện ích:

```

1 TaskCallbackArg = Task | TaskStatusUpdateEvent |
 TaskArtifactUpdateEvent
2 TaskUpdateCallback = Callable[[TaskCallbackArg, AgentCard], Task]

```

Giải thích:

- **TaskCallbackArg**: Kiểu dữ liệu hợp nhất (Union) đại diện cho một trong ba loại phản hồi được gửi về từ agent con.
- **TaskUpdateCallback**: Một hàm callback được định nghĩa với kiểu đầu vào là **TaskCallbackArg** và **AgentCard**; trả về một **Task**.

Định nghĩa lớp **RemoteAgentConnections**

```

1 class RemoteAgentConnections:
2 """A class to hold the connections to the remote agents."""

```

Lớp bao đóng (*wrapper class*) để:

- Quản lý **A2AClient** đã được khởi tạo.
- Lưu và cung cấp thông tin từ **AgentCard**.
- Giao tiếp với agent con qua phương thức **send\_message()**.

Hàm khởi tạo **\_\_init\_\_()**:

```

1 def __init__(self, agent_card: AgentCard, agent_url: str):
2 print(f'agent_card: {agent_card}')
3 print(f'agent_url: {agent_url}')
4 self._httpx_client = httpx.AsyncClient(timeout=30)
5 self.agent_client = A2AClient(
6 self._httpx_client, agent_card, url=agent_url
7)

```

```
8 | self.card = agent_card
```

Giải thích:

- Khởi tạo HTTP client bắt đồng bộ với timeout 30 giây.
- Tạo A2AClient để thực hiện giao tiếp với agent từ xa.
- Lưu lại thông tin AgentCard để dùng về sau.

Phương thức `get_agent()`:

```
1 | def get_agent(self) -> AgentCard:
2 | return self.card
```

Trả về metadata của agent từ xa – phục vụ mục đích hiển thị hoặc ghi log.

Phương thức chính `send_message()`:

```
1 | async def send_message(self, message_request: SendMessageRequest) ->
2 | SendMessageResponse:
3 | return await self.agent_client.send_message(message_request)
```

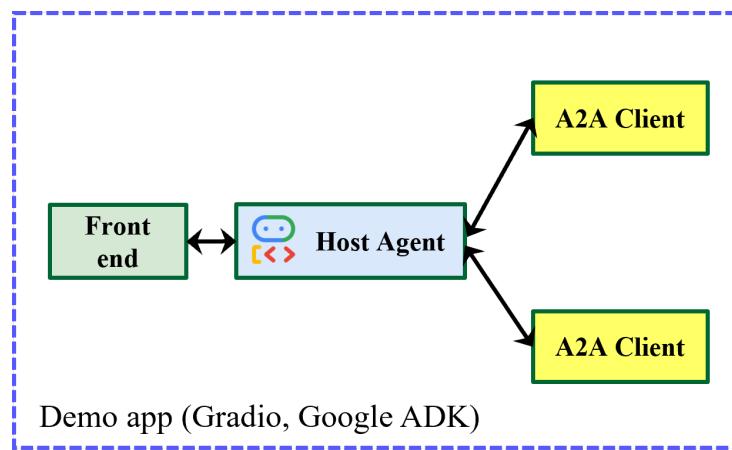
Đây là phương thức chính để giao tiếp A2A:

- Nhận đầu vào là một `SendMessageRequest` (chuẩn A2A).
- Trả về một `SendMessageResponse` chứa trạng thái và phản hồi từ agent con.

**Phần 2: Build RoutingAgent (host\_agent/routing\_agent.py)**

**Mục tiêu:** Xây dựng tác tử trung tâm (host agent) có khả năng định tuyến yêu cầu của người dùng đến các agent chuyên biệt như `airbnb_agent` hoặc `weather_agent`. Tác tử này sử dụng mô hình ngôn ngữ tự nhiên và công cụ

`send_message()` để giao tiếp với các agent con theo kiến trúc Agent-to-Agent (A2A).



Hình 65.12: Host Agent - Agent chứa các A2A Client .

### Import các thành phần chính:

```
1 import asyncio, json, os, uuid
2 from typing import Any
3
4 import httpx
5 from a2a.client import A2ACardResolver
6 from a2a.types import (
7 AgentCard,
8 SendMessageRequest,
9 SendMessageResponse,
10 ...
11)
12 from agents.airbnb_planner_multiagent.host_agent.
13 remote_agent_connection import ...
13 from google.adk import Agent
14 from google.adk.agents.callback_context import CallbackContext
15 from google.adk.agents.readonly_context import ReadonlyContext
16 from google.adk.tools.tool_context import ToolContext
```

### Giải thích các thành phần:

- **Agent:** Lớp tác tử chính thuộc SDK ADK của Google – cho phép tạo LLM Agent có công cụ và trạng thái.
- **ToolContext, ReadonlyContext, CallbackContext:** Giao diện cung cấp quyền truy cập các thành phần khác nhau trong vòng đời agent, bao gồm session, biến môi trường, bộ nhớ, phản hồi...
- **A2ACardResolver:** Công cụ giúp truy xuất **AgentCard** từ các agent con qua HTTP (gồm metadata như tên, mô hình, kỹ năng...).
- **RemoteAgentConnections:** Lớp bao đóng cho phép kết nối, gửi yêu cầu và nhận phản hồi từ các agent con thông qua chuẩn A2A.
- **SendMessageRequest, SendMessageResponse:** Định dạng dữ liệu chuẩn A2A để giao tiếp giữa các agent – mô tả yêu cầu và phản hồi tác vụ.

### Khởi tạo lớp **RoutingAgent**:

```

1 class RoutingAgent:
2 def __init__(self, task_callback=None):
3 self.task_callback = task_callback
4 self.remote_agent_connections = {}
5 self.cards = {}
6 self.agents = ""

```

Lưu trữ danh sách kết nối đến các agent con (`RemoteAgentConnections`), duy trì metadata trong `cards`, và mô tả các agent trong `self.agents`.

### Khởi tạo bất đồng bộ các thành phần:

```

1 async def _async_init_components(self, remote_agent_addresses):
2 async with httpx.AsyncClient(timeout=30) as client:
3 for address in remote_agent_addresses:
4 card = await A2ACardResolver(client, address).
5 get_agent_card()
6 remote_connection = RemoteAgentConnections(card, address)
7 self.remote_agent_connections[card.name] =
8 remote_connection
9 self.cards[card.name] = card

```

Gửi HTTP đến các địa chỉ agent con để lấy thông tin AgentCard và thiết lập kết nối.

### Tạo agent định tuyến sử dụng SDK:

```

1 def create_agent(self) -> Agent:
2 return Agent(
3 model='gemini-2.5-flash',
4 name='Routing_agent',
5 instruction=self.root_instruction,
6 tools=[self.send_message],
7 ...
8)

```

Sử dụng mô hình gemini-2.5-flash, và khai báo công cụ duy nhất là `send_message()` để tương tác với agent con.

### Sinh lệnh hành vi cho agent:

```

1 def root_instruction(self, context: ReadonlyContext) -> str:
2 current_agent = self.check_active_agent(context)
3 return f"""
4 **Role:** You are a Routing Delegator...
5 * **Task Delegation:** Use 'send_message'...
6 ...
7 * Available Agents: '{self.agents}'
8 * Currently Active Seller Agent: '{current_agent['active_agent']}
9 }'
 """

```

Tạo hướng dẫn hành vi cho LLM định tuyến, giúp phân phối đúng tác vụ, truyền ngữ cảnh phù hợp và không hỏi lại người dùng nếu không cần thiết.

### Kiểm tra agent đang xử lý tác vụ:

```

1 def check_active_agent(self, context):
2 state = context.state
3 if 'session_active' and 'active_agent' in state:
4 return {'active_agent': state['active_agent']}
5 return {'active_agent': 'None'}

```

Truy xuất agent đang xử lý hiện tại từ `state` của phiên.

### Xử lý trước khi gọi model:

```

1 def before_model_callback(self, callback_context, llm_request):
2 state = callback_context.state
3 if not state.get('session_active'):
4 state['session_id'] = uuid.uuid4()
5 state['session_active'] = True

```

Khởi tạo phiên hội thoại nếu chưa tồn tại để giữ ngữ cảnh liên tục.

**Liệt kê các agent con khả dụng:**

```

1 def list_remote_agents(self):
2 for card in self.cards.values():
3 remote_agent_info.append({'name': card.name, 'description':
4 card.description})

```

Chuẩn bị danh sách các agent để chèn vào instruction.

**Công cụ chính để giao tiếp – send\_message():**

```

1 async def send_message(self, agent_name: str, task: str,
 tool_context: ToolContext):
2 ...
3 payload = {
4 'message': {
5 'role': 'user',
6 'parts': [{'type': 'text', 'text': task}],
7 'messageId': uuid.uuid4()
8 },
9 ...
10 }
11 message_request = SendMessageRequest(...)
12 response = await client.send_message(message_request)
13 return response.root.result

```

Gửi đoạn hội thoại đến agent con đã chọn thông qua chuẩn A2A, sử dụng lớp RemoteAgentConnections.

**Hàm tiện trợ để khởi tạo đồng bộ:**

```

1 def _get_initialized_routing_agent_sync() -> Agent:
2 return asyncio.run(_async_main())

```

Cho phép khởi tạo agent một cách đồng bộ trong môi trường không hỗ trợ `async` như Gradio hoặc Streamlit.

### Phần 3: Chạy Host Agent (`host_agent/main.py`)

**Mục tiêu:** Tập tin `main.py` đóng vai trò điểm khởi chạy chính cho toàn bộ hệ thống host agent. Tại đây, ta khởi tạo phiên làm việc với `RoutingAgent` và triển khai giao diện `Gradio` để người dùng có thể tương tác trực tiếp với hệ thống thông qua trình duyệt.

Import các thành phần chính:

```
1 import asyncio, traceback
2 from collections.abc import AsyncIterator
3 from pprint import pformat
4 import gradio as gr
5
6 from agents.airbnb_planner_multiagent.host_agent.routing_agent
7 import root_agent as routing_agent
8 from google.adk.events import Event
9 from google.adk.runners import Runner
10 from google.adk.sessions import InMemorySessionService
11 from google.genai import types
```

Giải thích các thành phần:

- `routing_agent`: Được khởi tạo sẵn từ file `routing_agent.py`, đây chính là host agent trung tâm.
- `Runner`: Dùng để thực thi agent ADK như một tác tử động trong hệ thống, chịu trách nhiệm xử lý phiên và phản hồi.
- `Event`: Mô hình hoá một sự kiện dữ liệu ADK/A2A – có thể là đoạn phản hồi hoặc tín hiệu kết thúc.
- `InMemorySessionService`: Lưu trạng thái cuộc trò chuyện trong RAM, giúp duy trì ngữ cảnh theo từng phiên làm việc.

- gradio: Thư viện Python đơn giản để xây dựng giao diện người dùng tương tác web.

### Cấu hình các biến phiên mặc định:

```

1 APP_NAME = 'routing_app'
2 USER_ID = 'default_user'
3 SESSION_ID = 'default_session'
```

Đặt tên ứng dụng, người dùng, và mã phiên mặc định được sử dụng trong hệ thống để khởi tạo tương tác với agent.

### Khởi tạo Runner từ Routing Agent:

```

1 SESSION_SERVICE = InMemorySessionService()
2 ROUTING_AGENT_RUNNER = Runner(
3 agent=routing_agent,
4 app_name=APP_NAME,
5 session_service=SESSION_SERVICE,
6)
```

**Ý nghĩa:** Runner hoạt động như middleware giúp thực thi agent theo từng phiên làm việc, hỗ trợ lưu session qua InMemorySessionService.

### Xử lý phản hồi từ agent – `get_response_from_agent()`:

```

1 async def get_response_from_agent(message: str, history: list[gr.
 ChatMessage]):
2 event_iterator = ROUTING_AGENT_RUNNER.run_async(...)
3
4 async for event in event_iterator:
5 if event.content and event.content.parts:
6 for part in event.content.parts:
7 if part.function_call:
8 yield gr.ChatMessage(...)
9 elif part.function_response:
```

```

10 yield gr.ChatMessage(...)
11 if event.is_final_response():
12 yield gr.ChatMessage(...)

```

### Các loại phản hồi được xử lý:

- `function_call`: Agent đang gọi đến một công cụ.
- `function_response`: Phản hồi từ công cụ đã được agent nhận.
- `is_final_response()`: Đánh dấu kết thúc chuỗi phản hồi từ agent.

Nếu có lỗi xảy ra trong quá trình xử lý, hệ thống sẽ ghi lại `traceback` và gửi lỗi về UI cho người dùng biết.

### Hàm `main()` – Khởi tạo Gradio app:

```

1 async def main():
2 await SESSION_SERVICE.create_session(...)
3
4 with gr.Blocks(...) as demo:
5 gr.Image(...)
6 gr.ChatInterface(
7 get_response_from_agent,
8 title='A2A Host Agent',
9 description='This assistant can help you to check
 weather and find airbnb
 accommodation',
10)
11
12 demo.queue().launch(server_name='0.0.0.0', server_port=8083)

```

### Ý nghĩa:

- Giao diện Gradio sử dụng `ChatInterface` cho phép trò chuyện hai chiều.
- Hỗ trợ phản hồi dạng streaming nhờ `yield`.

- Giao diện sẽ hiển thị hình ảnh nhận diện, tiêu đề và mô tả rõ ràng.

**Khởi chạy ứng dụng:**

```
1 if __name__ == '__main__':
2 asyncio.run(main())
```

**Ý nghĩa:** Khi chạy trực tiếp tệp, hệ thống sẽ khởi tạo phiên và triển khai Gradio UI tại địa chỉ <http://localhost:8083>.

#### 65.3.4 Cloning GitHub Repo và chạy Demo

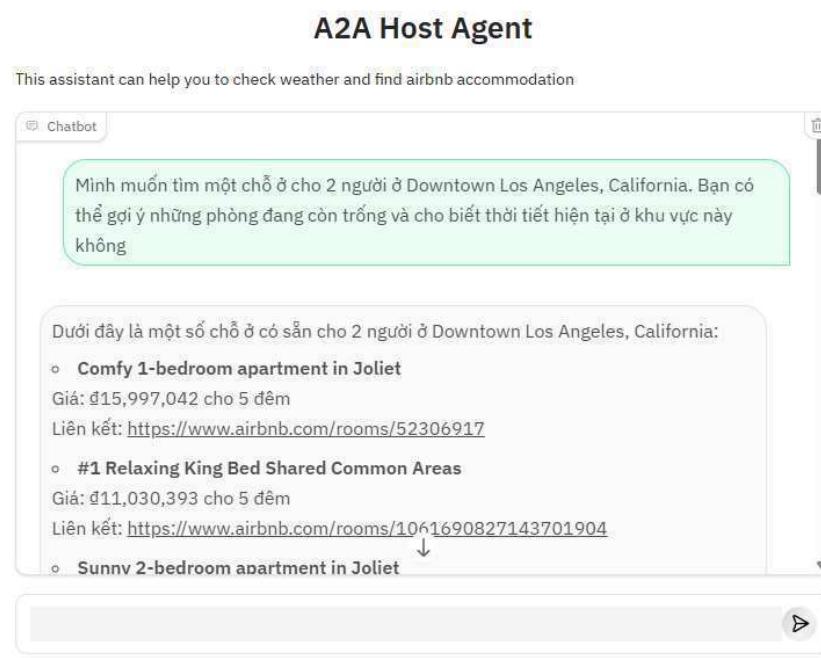
Mã nguồn mẫu được cung cấp tại: <https://github.com/quaghien/A2A-samples>.

Để chạy demo hệ thống multi-agent theo chuẩn A2A, người dùng cần chuẩn bị:

- **Python 3.13:** để chạy a2a-sdk.
- **uv:** công cụ quản lý môi trường Python.
- **Node.js (v20):** phục vụ một số tool (như Airbnb MCP server).
- **.env file:** chứa cấu hình API key và các biến môi trường khác.

Sau đó đọc hướng dẫn chi tiết ở README.md của repo để chạy thử demo.

**App UI:**



Hình 65.13: App UI khi chạy demo

## Kế hoạch cập nhật nội dung AIO cho năm 2026

