TRƯỜNG ĐẠI HỌC CÔNG NGHỆ - ĐHQGHN
VIỆN TRÍ TUỆ NHÂN TẠO

———————— * ————————

HỌC PHẦN: HỌC MÁY
BÁO CÁO BÀI TẬP CUỐI KỲ

Đề tài:
Cài đặt mô hình Logistic Regression
Ứng dụng trong bài toán phân loại tin tức

Mã LHP: INT3405# 5
Sinh viên: Vũ Hoàng Diệu Linh
Mã sinh viên: 24022387

Hà Nội, 1/2026

MỤC LỤC

LỜI NÓI ĐẦU 3
PHẦN 1: GIỚI THIỆU 4
1.1. Mô tả bài toán 4
1.2. Mục tiêu của bài tập 4
PHẦN 2: CƠ SỞ LÝ THUYẾT 5
2.1. Thuật toán Hồi quy Logistic (Logistic Regression) 5
2.2. Công thức toán học của Logistic Regression 6
2.3. Cơ sở lý thuyết của các phương pháp tối ưu hóa 7
2.4. Cơ sở lý thuyết của phương pháp xử lý dạng văn bản TF-IDF (phát triển) 8
2.5. Cơ sở lý thuyết của chiến lược xử lý bài toán phân loại nhiều hơn 2 lớp One-vs-Rest 8
PHẦN 3: PHƯƠNG PHÁP THỰC HIỆN 8
3.1. Kiến trúc mô hình 8
3.2. Dữ liệu huấn luyện và dữ liệu kiểm thử (Training and testing data) 8
3.3. Quy trình huấn luyện 10
PHẦN 4: THỰC NGHIỆM 17
4.1. Thiết lập thí nghiệm 17
4.2. Tham số huấn luyện 17
4.3. Kết quả định lượng 17
PHẦN 5: SO SÁNH VÀ PHÂN TÍCH 19
5.1. Bản baseline cài đặt bằng sự hỗ trợ của thư viện 19
5.2. So sánh mô hình tự cài đặt (Scratch) và Baseline (Scikit-learn) 20
5.3. Phân tích 21
PHẦN 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 22
LỜI CẢM ƠN 22
NGUỒN DỮ LIỆU, TÀI LIỆU THAM KHẢO 23

LỜI NÓI ĐẦU

- **Github:** github

Lời đầu tiên, em xin gửi lời cảm ơn chân thành đến thầy Trần Quốc Long, giảng dạy môn Học máy, đã tận tình hướng dẫn, truyền đạt kiến thức và tạo điều kiện để chúng em có cơ hội được vận dụng các kiến thức đã học vào bài tập này.

Trong bối cảnh Trí tuệ Nhân tạo phát triển mạnh mẽ, các mô hình Học máy – cơ sở của Trí tuệ nhân tạo càng có vai trò quan trọng hơn. Hiện nay, việc tìm hiểu và vận dụng các thuật toán vào dữ liệu thực tế được các ngôn ngữ lập trình như Python hỗ trợ phần nhiều qua các thư viện, nhưng để thực sự làm chủ một mô hình nào đó không có cách nào tốt hơn việc có thể cài đặt lại mà chỉ dùng các thư viện cơ bản. Ở bài tập này, em lựa chọn cài đặt lại thuật toán Logistic Regression (Hồi quy Logistic), ứng dụng vào bài toán cụ thể.

Em kỳ vọng qua bài tập này em có thể củng cố vững chắc kiến thức về Logistic Regression nói riêng, là nền tảng định hướng cho em cách tìm hiểu, nghiên cứu các thuật toán khác nói chung và có thể ứng dụng chúng một cách hiệu quả vào dữ liệu thực tế. Một lần nữa, em xin trân trọng cảm ơn thầy và các bạn đã đồng hành và hỗ trợ trong suốt quá trình học tập, tìm hiểu, nghiên cứu và hoàn thành bài tập.

PHẦN 1: GIỚI THIỆU

1.1. Mô tả bài toán

** Về bài toán ứng dụng*

Trong kỷ nguyên số hiện nay, lượng thông tin dạng văn bản (text data) được tạo ra mỗi ngày là vô cùng khổng lồ. Trong bài tập này, yêu cầu được đặt ra là phân loại các bài báo từ tập dữ liệu AG News vào 4 chủ đề chính: Thế giới (World), Thể thao (Sports), Kinh doanh (Business) và Khoa học/Công nghệ (Sci/Tech). Đây là bài toán phân loại đa lớp (Multi-class Classification) dựa trên đặc trưng nội dung của tiêu đề và phần mô tả bài báo.

** Về kỹ thuật cài đặt*

Với bài toán ứng dụng phân loại tin tức ở trên, chúng ta có rất nhiều phương pháp và thư viện hỗ trợ mạnh mẽ. Tuy nhiên, trọng tâm của bài tập này không nằm ở việc sử dụng các công cụ có sẵn, mà là việc tái cấu trúc và cài đặt lại thuật toán Hồi quy Logistic (Logistic Regression) từ những nền tảng toán học cơ bản nhất. Việc tự xây dựng mô hình giúp hiểu rõ cách thức xử lý dữ liệu thực tế và cơ chế tối ưu hóa bài toán phân loại đa lớp trong môi trường dữ liệu thực tế như trên.

1.2. Mục tiêu của bài tập

Với bài toán được mô tả như trên, mục tiêu trọng tâm của bài tập này bao gồm:

1. Nắm vững cơ chế hoạt động của thuật toán Logistic Regression, các hàm tối ưu (Loss Function) và phương pháp tối ưu hóa (Gradient Descent).

2. Tự cài đặt thuật toán từ đầu bằng ngôn ngữ Python và các thư viện tính toán cơ bản mà không phụ thuộc vào các thư viện học máy có sẵn (như Scikit-learn). Củng cố khả năng thực hiện các phép toán trên ma trận, đảm bảo sự ổn định trên tập dữ liệu có số lượng đặc trưng lớn (nhiều chiều – high dimensional).

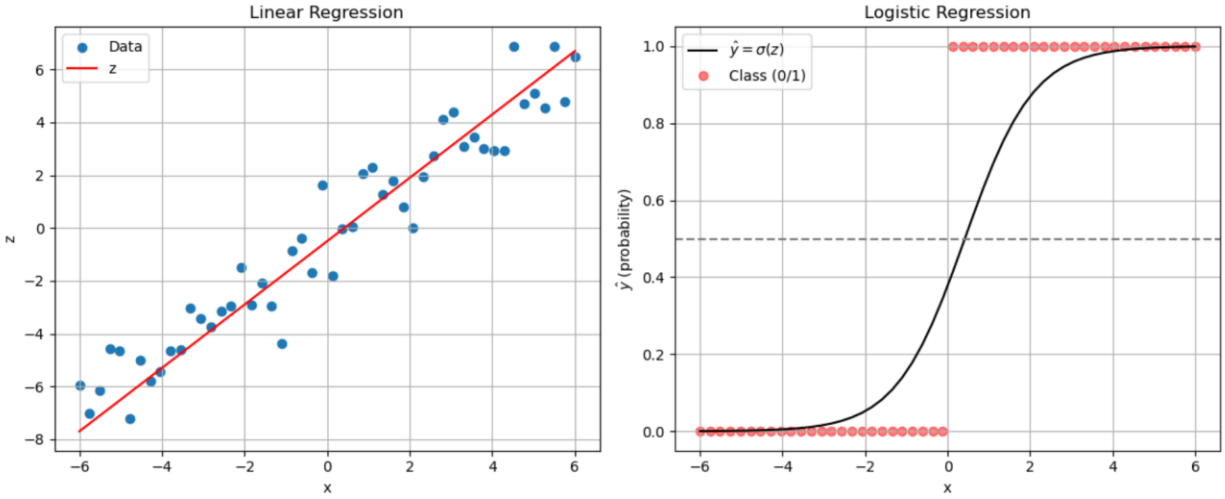
3. Sử dụng được các kỹ thuật tiền xử lý dữ liệu (NLP Preprocessing), trích xuất đặc trưng (TF-IDF) để chuyển đổi dữ liệu tin tức thô thành định dạng vector phù hợp với mô hình.

4. Tự đánh giá, kiểm chứng độ chính xác của thuật toán tự cài đặt so với việc sử dụng các thư viện chuẩn nhằm khẳng định tính đúng đắn của logic thuật toán và kỹ năng lập trình.

PHẦN 2: CƠ SỞ LÝ THUYẾT

2.1. Thuật toán Hồi quy Logistic (Logistic Regression)

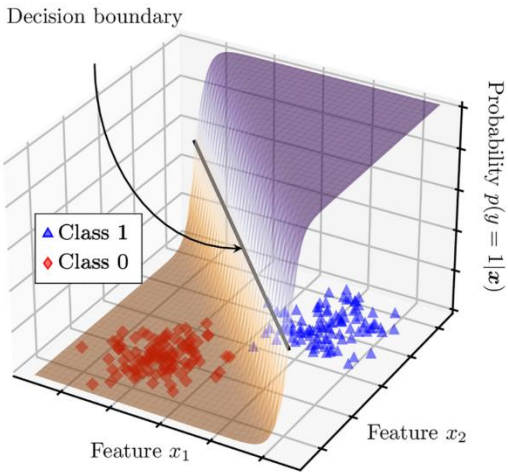
- Hồi quy logistic là một thuật toán được sử dụng để phân loại các quan sát theo các danh mục rời rạc. Thay vì có đầu ra là các giá trị liên tục như thuật toán hồi quy tuyến tính (linear regression), ý tưởng chính là dự đoán xác suất một mẫu dữ liệu thuộc về một lớp nhất định bằng cách sử dụng hàm *Sigmoid* ($\sigma(z)$) để ánh xạ giá trị đầu ra $z \in (-\infty, +\infty)$ về $\hat{y} = \sigma(z), \hat{y} \in [0, 1]$.



Hình 1. Minh họa Linear Regression và Logistic Regression

Với dữ liệu bài toán có nhiều đặc trưng (features), khi đó ta có:

- $x = (x_1, x_2, \ldots, x_n)$là một vector nhiều chiều.
- Giá trị z là một hàm tuyến tính và xác định một siêu phẳng trong không gian đặc trưng.
- Ranh giới phân loại (decision boundary) được xác định bởi phương trình $z = 0$.
- Xác suất dự đoán được tính bởi $\hat{y} = \sigma(z)$, tạo thành một bề mặt *sigmoid* trên không gian đặc trưng.



Hình 2. Minh họa LR với 2 đặc trưng

Với dữ liệu đầu vào nằm trong không gian hai chiều, hàm *sigmoid* có dạng thác nước như hình bên (Hình 2.)

2.2. Công thức toán học của Logistic Regression

Trong phần *2.1*, ta đã giới thiệu về các hàm $z, \hat{y} = \sigma(z)$, ... Để cụ thể, chúng ta cùng làm rõ qua các công thức toán học và cách cài đặt chúng trong code.

a) Giá trị z

$$z = w^{(T)}X + b$$

Trong đó:

X là vector chứa các đặc trưng của bài toán
(khi $dimX > 1, z = w^T X + b$);
w, b là các tham số (cụ thể: w là vector trọng số, b là bias);
$z \in (-\infty, +\infty)$.

b) Hàm sigmoid ($\sigma(z)$)

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \sigma(z) \in [0, 1]$$

Hàm này có nhiệm vụ ánh xạ giá trị đầu ra z về dạng đại diện cho xác suất, là mấu chốt của mô hình Logistic Regression.

c) Hàm mất mát (Loss function - $J(w, b)$)

$$J(w, b) = -\frac{1}{m}\sum_{1}^{m} y_i.\log(\hat{y_i}) + (1 - y_i).\log(1 - \hat{y_i})$$

Trong đó:

y_i: lớp thực tế của phần tử thứ i;
$\hat{y_i} = \sigma(z_i)$: lớp mà mô hình dự đoán của phần tử thứ i;
$y_i, \hat{y_i} \in [0, 1]$.

Công thức này được suy ra từ Nguyên lý ước lượng hợp lý cực đại (MLE –Maximum Likelihood Estimation) của Xác suất và thống kê.

Hàm này đo lường sự sai khác giữa xác suất lớp dự đoán $\hat{y}$ và lớp thực tế y với m là kích thước của dữ liệu huấn luyện. Mục tiêu của thuật toán là:

$$minimize_{w,b}\, J(w, b)$$

2.3. Cơ sở lý thuyết của các phương pháp tối ưu hóa

a) Phương pháp cập nhật trọng số Gradient Descent

Mục đích cuối cùng là cần tìm ra w, b tối ưu nhất. Sau mỗi vòng lặp, Gradient Descent cập nhật w, b theo công thức đạo hàm như sau:

$$w = w - \alpha.\frac{1}{m}.X^T.(\hat{y} - y)$$

$$b = b\ - \alpha.\frac{1}{m}.\sum \hat{y} - y$$

Với α là tốc độ học (learning rate).

b) Phương pháp giảm dần tốc độ học - Learning rate Decay

Việc lựa chọn tốc độ học η có thể quá lớn, quá nhỏ gây ra mô hình hoạt động không ổn định, dẫn đến tốn thời gian, tài nguyên mà lại ảnh hưởng cả về độ chính xác. Để giải quyết vấn đề này, ta có phương pháp giảm dần tốc độ học với công thức như sau:

$$\eta_t = \frac{\eta_0}{1 + k.t}$$

Trong đó:

η_t là tốc độ học ở epoch thứ t

η_0 là tốc độ học ban đầu

k là hệ số suy giảm – `decay_rate`

Kỹ thuật này đóng vai trò điều tiết tốc độ học của mô hình, học nhanh ở những epoch đầu tiên và giảm dần về sau để chỉnh các tham số, trọng số một cách chi tiết để đạt độ chính xác cao nhất.

c) Mini-batch Gradient Descent

Trong huấn luyện mô hình, thay vì sử dụng toàn bộ dữ liệu, Mini-batch Gradient Descent chia tập dữ liệu huấn luyện thành các nhóm nhỏ (batches) có kích thước K (thường là 32, 64, ..., 1024).

Công thức:

$$w = w - \eta.\frac{1}{K}.\sum_{i\ \in B} \nabla J(w, x_i, y_i)$$

với mỗi batch B chứa K mẫu, trọng số w được cập nhật theo công thức trên, trong đó η là tốc độ học.

Việc chia nhỏ tập dữ liệu thành các batch giúp tăng tốc độ hội tụ, tính ổn định và chính xác vì tận dụng được khả năng tính toán song song của CPU hiệu quả hơn với những thiết bị không có GPU mạnh mẽ.

Ở lần cải tiến thứ 4 trong quá trình huấn luyện, em lấy batch size $K = 1024$.

2.4. Cơ sở lý thuyết của phương pháp xử lý dạng văn bản TF-IDF (phát triển)

2.5. Cơ sở lý thuyết của chiến lược xử lý bài toán phân loại nhiều hơn 2 lớp One-vs-Rest

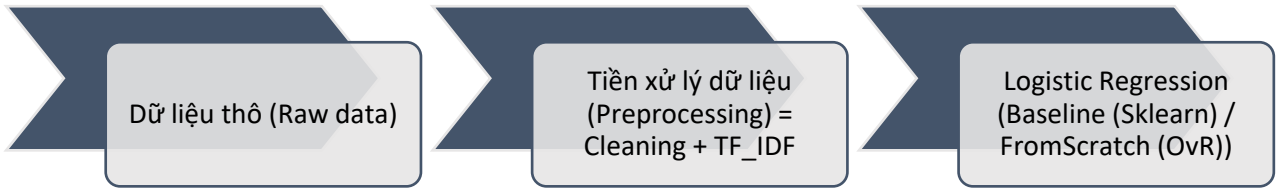
Chiến lược One-vs-Rest là một phương pháp cho phép sử dụng các thuật toán phân loại nhị phân (điển hình như Logistic Regression) để giải quyết bài toán có nhiều hơn 2 lớp ($N > 2$).

Nguyên lý: Chia bài toán phân loại N lớp thành N bài toán phân loại nhị phân riêng biệt.

Cách hoạt động: Với mỗi lớp i, ta huấn luyện một bộ phân loại nhị phân để dự đoán lớp i hiện tại (lớp positive) với tất cả $N - 1$ lớp còn lại gộp chung (lớp negative). Khi cần dự đoán một điểm dữ liệu mới, ta đưa nó qua cả N mô hình. Mỗi mô hình sẽ trả về một xác suất (probability). Lớp nào có xác suất cao nhất sẽ được chọn làm kết quả dự đoán cuối cùng.

PHẦN 3: PHƯƠNG PHÁP THỰC HIỆN

3.1. Kiến trúc mô hình



3.2. Dữ liệu huấn luyện và dữ liệu kiểm thử (Training and testing data)

a) Nguồn dữ liệu: Kaggle

b) Tiền xử lý (Preprocessing)

** Quan sát dữ liệu (Observing data):*

1. Nhập và quan sát kích thước, các đặc trưng của 2 tập dữ liệu (huấn luyện và kiểm thử).

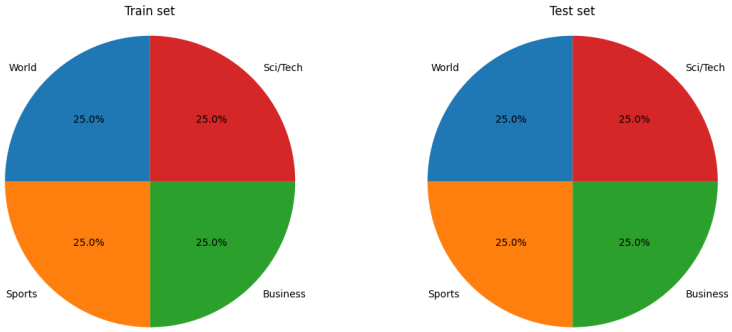
```
train_df = pd.read_csv("./data/train.csv")
test_df = pd.read_csv("./data/test.csv")

print("Train shape:", train_df.shape) # -> Train shape: (120000, 3)
print("Test shape:", test_df.shape) # -> Test shape: (7600, 3)
```

Hình 3. Cài đặt việc nhập dữ liệu

Class Index	Title	Description
3	Wall St. Bears Claw Back Into the Black (Reuters)	Reuters - Short-sellers, Wall Street's dwindling\band of...
3	Carlyle Looks Toward Commercial Aerospace (Reuters)	Reuters - Private investment firm Carlyle Group,\which...
3	Oil and Economy Cloud Stocks' Outlook (Reuters)	Reuters - Soaring crude prices plus worries\about the e...

Hình 4. Minh họa các đặc trưng của dữ liệu (train_df)

2. Vẽ biểu đồ phân bố các đầu ra của dữ liệu:



Hình 5. Biểu đồ thể hiện phân bố các lớp trong tập huấn luyện (Train set) và tập kiểm tra (Test set)

=> Nhận xét: Mỗi lớp (World, Sports, Business, Sci/Tech) chiếm 25% tổng số mẫu. Vậy cả 2 tập huấn luyện và kiểm thử đều là tập dữ liệu cân bằng nhãn (balanced dataset), điều này giúp giảm thiểu nguy cơ sai lệch dữ liệu (data bias), đồng thời đảm bảo rằng các chỉ số đánh giá của mô hình Logistic Regression phản ánh chính xác khả năng tổng quát hóa của mô hình.

** Tiền xử lý dữ liệu (Preprocessing data):*

- Từ các quan sát trên, ta có thể tách đầu vào *X* và đầu ra *y* như sau: Gộp [“Title] và [“Description”] để trở thành dạng văn bản => là bước tiền đề để chuyển đầu vào dạng vector.

```
X = train_df["Title"] + " " + train_df["Description"]
y = train_df["Class Index"]
```

Hình 6. Minh họa đầu vào X, đầu ra y

- Chia dữ liệu huấn luyện: 80% train + 20% test:

```
X_train_sub, X_val, y_train_sub, y_val = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

Hình 7. Minh họa cài đặt chia dữ liệu

- Xử lý dữ liệu dạng văn bản (mấu chốt của bài tập dạng này):

Phiên bản 1: Sử dụng `TfidfVectorizer` được thư viện `sklearn` hỗ trợ:

```python
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(
    ngram_range=(1, 3),
    max_features=20000,
    stop_words='english'
)

X_train_tfidf = vectorizer.fit_transform(X_train_sub)
X_val_tfidf   = vectorizer.transform(X_val)
X_test_tfidf  = vectorizer.transform(
    test_df["Title"] + " " + test_df["Description"]
)
```

Hình 8.

Phiên bản 2: Tự cài đặt TF_IDF dựa trên cơ sở lý thuyết (phát triển)

3.3. Quy trình huấn luyện: Sử dụng phương pháp cải tiến lũy kế - thử nhiều lần để tìm ra hướng đi tốt hơn để có 1 mô hình tốt lên dần dần qua các lần thử. Ở phần này có được so sánh luôn với bản baseline sử dụng chung đầu vào

** Lần 1:*

- Sử dụng phiên bản 1 của TF_IDF với tham số như minh họa bên:

```python
vectorizer = TfidfVectorizer(
    max_features=5000,
    stop_words='english'
)
```

Hình 9. Cài đặt TF_IDF version 1

- Class LRFromScratch (cài đặt Logistic Regression) và chiến lược One-vs-Rest được cài đặt lần lượt dưới đây.

```python
class LRfromScratch:
    def __init__(self, lr=0.1, iterations=1000):
        self.lr = lr
        self.iterations = iterations
        self.weights = None
        self.bias = None
        self.loss_history = []


    def _sigmoid(self, z):
        # Clipping z để tránh lỗi overflow khi tính exp(-z)
        z = np.clip(z, -500, 500)
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y, batch_size=1024):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        old_loss = float('inf')
        for i in range(self.iterations):
            # 1. Tính z và y_hat
            z = X.dot(self.weights) + self.bias
            y_hat = self._sigmoid(z)

            # 2. Gradient descent
            dw = (1 / n_samples) * X.T.dot(y_hat - y)
            db = (1 / n_samples) * np.sum(y_hat - y)

            self.weights -= self.lr * dw
            self.bias -= self.lr * db

            # Lưu loss để vẽ biểu đồ
            if i % 100 == 0:
                loss = -np.mean(y * np.log(y_hat + 1e-15) + (1 - y) * np.log(1 - y_hat + 1e-15))
                self.loss_history.append(loss)
                print(f"Epoch {i} - Loss: {loss:.4f} - LR: {self.lr:.5f}")

    def predict_proba(self, X):
        z = X.dot(self.weights) + self.bias
        return self._sigmoid(z)

    def predict(self, X):
        probabilities = self.predict_proba(X)
        return [1 if i > 0.5 else 0 for i in probabilities]
```

Hình 10. Class LRFromScratch version 1

```python
models = {}
classes = [1, 2, 3, 4]

for c in classes:
    print(f"Đang huấn luyện bộ phân loại cho lớp {c}...")
    # Biến bài toán thành nhị phân: Lớp c là 1, các lớp khác là 0
    y_train_binary = np.where(y_train_sub == c, 1, 0)

    model = LRfromScratch(lr=0.8, iterations=500)
    model.fit(X_train_tfidf, y_train_binary, batch_size=1024)
    models[c] = model

final_probs = []
for c in classes:
    final_probs.append(models[c].predict_proba(X_val_tfidf))

final_probs = np.array(final_probs).T
val_pred_scratch = np.argmax(final_probs, axis=1) + 1
```

Hình 11. One-vs-Rest version 1

Mô hình lần 1 này khi chạy với dữ liệu kiểm thử (80% train data) được kết quả ở bảng báo cáo như hình bên:

```
Accuracy (From Scratch): 0.859625

Classification Report (From Scratch):
              precision    recall  f1-score   support

           1       0.87      0.87      0.87      6000
           2       0.86      0.97      0.91      6000
           3       0.86      0.79      0.83      6000
           4       0.85      0.81      0.83      6000

    accuracy                           0.86     24000
   macro avg       0.86      0.86      0.86     24000
weighted avg       0.86      0.86      0.86     24000
```

** Lần 2: Cải tiến tham số của vectorizer:*

N-grams (1, 2): Giúp mô hình hiểu được các cụm từ có nghĩa (vd: "stock market", "world cup").

Max Features: Tăng từ 5,000 lên 15,000 để bắt được nhiều từ khóa hơn.

```python
vectorizer = TfidfVectorizer(
    ngram_range=(1, 2),
    max_features=15000,
    stop_words='english'
)

```

Hình 13. Vectorizer version 2

Các phần còn lại giống với mô hình ở lần thử thứ nhất, ở lần này ta được kết quả như bảng sau:

```
Accuracy (From Scratch): 0.870542

              precision    recall  f1-score   support

1                  0.88      0.88      0.88      6000
2                  0.87      0.97      0.92      6000
3                  0.87      0.81      0.84      6000
4                  0.86      0.83      0.84      6000

accuracy                               0.87     24000
macro avg          0.87      0.87      0.87     24000
weighted avg       0.87      0.87      0.87     24000
```

Hình 14. Bảng báo cáo các chỉ số của version 2

** Lần 3: Tối ưu hóa Learning rate Decay*

Sử dụng cơ sở lý thuyết ở phần trên, ta tối ưu vào hàm `fit (X, y)` của Class `LrfromScratch` như hình bên, với `decay_rate = 0.001`.

```
# cải tiến lần 3:
cur_lr = self.lr / (1 + self.decay_rate*i)
self.weights -= cur_lr * dw
self.bias -= cur_lr * db
```

Hình 15. Learning rate decay ở version 3

Các phần còn lại giống với mô hình ở lần thử thứ hai, ở lần này ta được kết quả như bảng sau:

```
Accuracy (From Scratch): 0.875458

              precision    recall  f1-score   support

1                  0.89      0.88      0.88      6000
2                  0.88      0.97      0.92      6000
3                  0.87      0.82      0.85      6000
4                  0.86      0.83      0.85      6000

accuracy                               0.88     24000
macro avg          0.88      0.88      0.87     24000
weighted avg       0.88      0.88      0.87     24000
```

Hình 16. Bảng báo cáo các chỉ số của version 3

** Lần 4: Mini-batch*

Là bước cải tiến quyết định, giảm thời gian chạy rất đáng kể (từ hơn 40 phút xuống 5-10 phút) và hiệu quả accuracy cao hơn hẳn những phiên bản trước (91%) nhờ cơ sở lý thuyết vững chắc ở phần trên.

Cụ thể quy trình thực hiện mỗi epoch như sau:

1. Xáo trộn ngẫu nhiên dữ liệu ở đầu mỗi epoch, đảm bảo tính khách quan và tránh mô hình học theo thứ tự dữ liệu.

2. Chia dữ liệu vào các ma trận thưa (sparse matrix)

3. Cập nhật các trọng số 1 cách liên tục theo công thức từ cơ sở lý thuyết.

```python
cur_lr = self.lr / (1 + self.decay_rate*i)
# cải tiến lần 4: mini batch
indices = np.random.permutation(n_samples)

for j in range(0, n_samples, batch_size):
    batch_idx = indices[j:j+batch_size]

    X_batch = X[batch_idx]
    y_batch = y[batch_idx]

    z = X_batch.dot(self.weights) + self.bias
    y_hat = self._sigmoid(z)

    diff = y_hat - y_batch
    dw = (1/len(batch_idx)) * X_batch.T.dot(diff)
    db = (1/len(batch_idx)) * np.sum(diff)

    self.weights -= cur_lr * dw
    self.bias -= cur_lr * db
```

Hình 17. Cài đặt Mini-batch ở version 4

Các phần còn lại giống với mô hình ở lần thử thứ ba, ở lần này ta được kết quả như bảng sau:

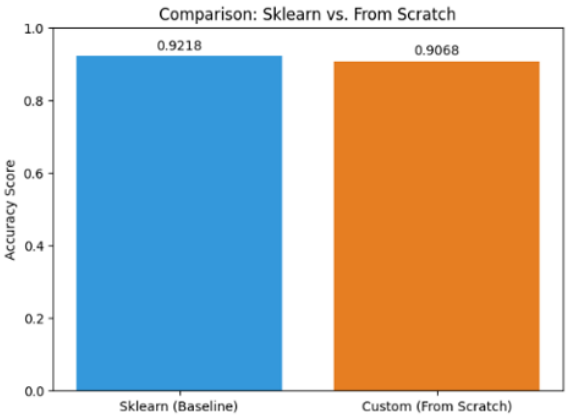
```
Accuracy (From Scratch): 0.906833

              precision    recall  f1-score   support

1                  0.92      0.90      0.91      6000
2                  0.94      0.98      0.96      6000
3                  0.89      0.87      0.88      6000
4                  0.89      0.88      0.88      6000

accuracy                               0.91     24000
macro avg          0.91      0.91      0.91     24000
weighted avg       0.91      0.91      0.91     24000
```

Hình 18. Bảng báo cáo các chỉ số của version 4

** Lần 5: Tăng các tham số của Vectorizer*

Nhận thấy, ở lần 4 mô hình có kết quả dự đoán khá cao (90%) => Mô hình đã khá hiệu quả, ta mang đi so sánh *accuracy* với bản baseline xem có chênh lệch nhiều không, để đưa ra hướng tối ưu tiếp theo phù hợp.

Biểu đồ bên (Hình 19) biểu diễn sự chênh lệch accuracy giữa mô hình baseline và mô hình tự cài đặt.



Hình 19. Sự chênh lệch accuracy baseline vs fromscratch version 5

** Lần 5: Tăng các tham số của Vectorizer*

=> Chênh lệch khoảng 15%, để cải thiện dần ta tăng các tham số của Vectorizer để đòi hỏi mô hình phải học nhiều và kỹ hơn các đặc trưng của dữ liệu đầu vào X. Cụ thể như hình bên, ta được kết quả như bảng dưới đây:

```python
vectorizer = TfidfVectorizer(
    ngram_range=(1, 3),
    max_features=20000,
    stop_words='english'
)
```

Hình 20

```
Accuracy (From Scratch): 0.913542

              precision  recall     f1-score   support

1             0.92       0.90       0.91       6000
2             0.95       0.98       0.96       6000
3             0.89       0.88       0.89       6000
4             0.90       0.89       0.89       6000

accuracy                            0.91       24000
macro avg     0.91       0.91       0.91       24000
weighted avg  0.91       0.91       0.91       24000
```

Hình 21. Bảng báo cáo các chỉ số của version 5

** Lần 6: Tăng epoch*

Ta tiếp tục đòi hỏi mô hình học nhiều lần bằng cách tăng epoch từ lên 2000, dừng sớm nếu $|loss_i - loss_{i-1}| < 10^{-6}$ (dừng khi 2 epoch liền nhau không thay đổi đáng kể). Ta cập nhật biến `iterations = 2000` và thêm 1 `if` như hình bên vào vòng lặp trong hàm `fit` của class `LRFromScratch`.

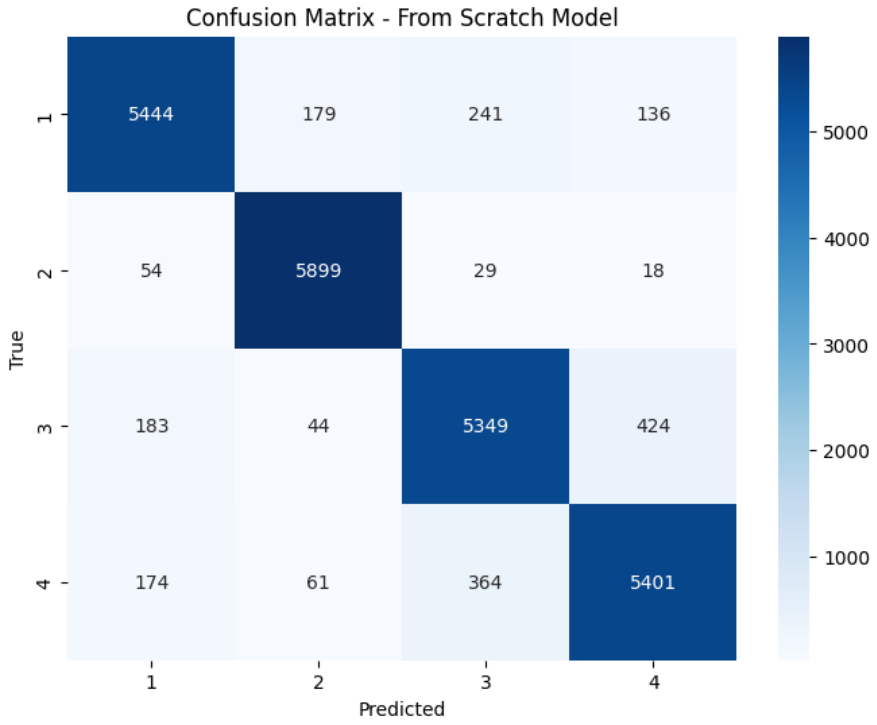
```python
if abs(old_loss - loss) < 1e-6:
    print(f"Dừng sớm tại Epoch {i}")
    break;
old_loss = loss
```

Hình 22

Ta được bảng kết quả và ma trận nhiệt như dưới đây:

```
Accuracy (From Scratch): 0.920542

              precision  recall     f1-score   support

1             0.93       0.91       0.92       6000
2             0.95       0.98       0.97       6000
3             0.89       0.89       0.89       6000
4             0.90       0.90       0.90       6000

accuracy                            0.92       24000
macro avg     0.92       0.92       0.92       24000
weighted avg  0.92       0.92       0.92       24000
```

Hình 23. Bảng báo cáo các chỉ số của version 6



Hình 24. Ma trận nhiệt biểu diễn kết quả của From Scratch Model

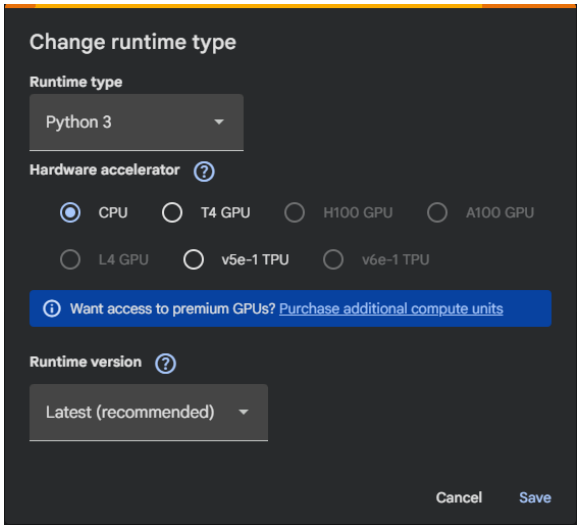
PHẦN 4: THỰC NGHIỆM

4.1. Thiết lập thí nghiệm

Thí nghiệm được tiến hành nhằm đánh giá khả năng phân loại của mô hình Logistic Regression tự triển khai so với baseline sử dụng thư viện trong việc giải quyết bài toán phân loại tin tức.

Môi trường:

- Google Colab CPU, RAM 12GB
- Ngôn ngữ: Python 3



Hình 25.
Thiết lập môi trường ở Google Colab

4.2. Tham số huấn luyện

Việc lựa chọn tham số có ảnh hưởng đến khả năng hội tụ của hàm mất mát. Trong bài tập này, các tham số đã được thực nghiệm và chọn lọc:

- Tốc độ học $\eta = 0.8$. Giá trị này cao vừa đủ để mô hình hội tụ nhanh và hoạt động ổn định.
- Số epoch: 2000. Số lần mô hình duyệt qua toàn bộ tập dữ liệu. Con số 2000 là đủ, không chạy quá lâu và cho kết quả chính xác cao.
- Batch_size = 1024. Số lượng mẫu dữ liệu được xử lý trong mỗi lần cập nhật trọng số (Mini-batch Gradient Descent)
- Hàm kích hoạt: σ

4.3. Kết quả định lượng

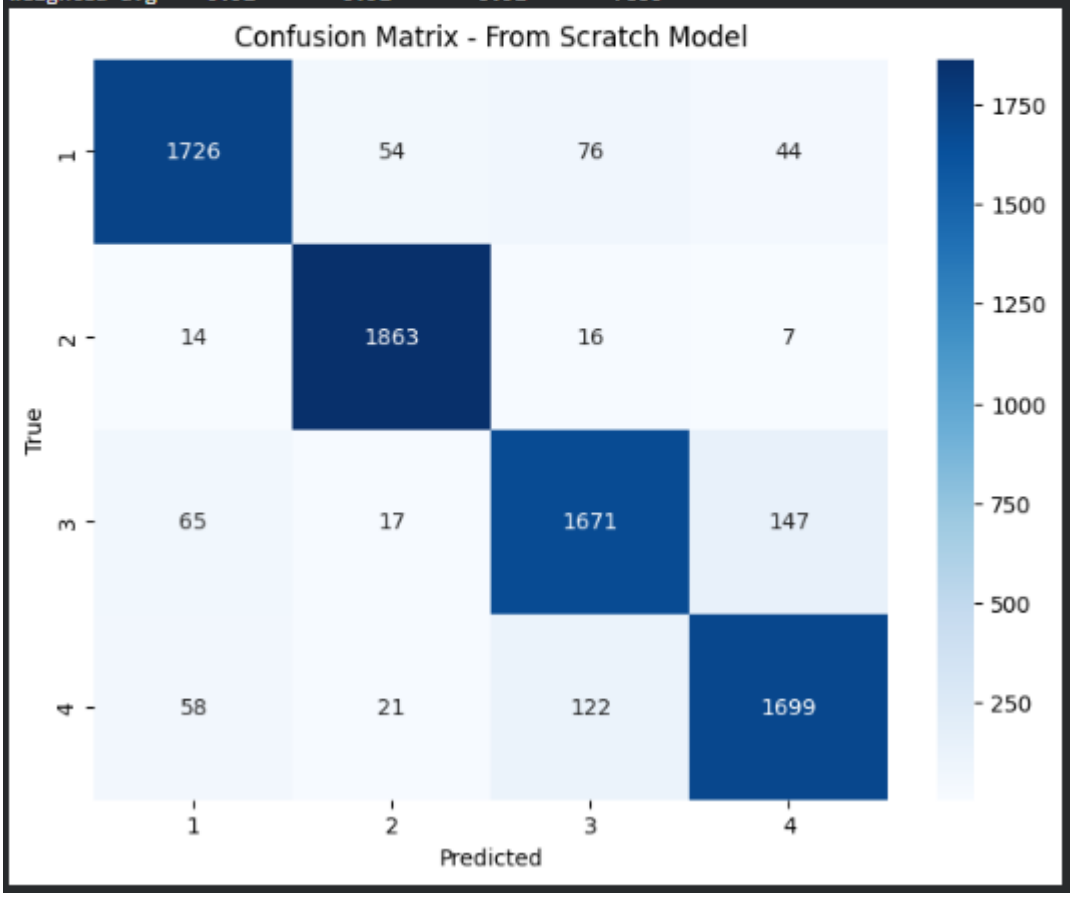
- Test ở 20% của dữ liệu huấn luyện thu được kết quả ở phần trước, version 6 của quy trình huấn luyện.

- Test ở dữ liệu kiểm thử thu được bảng báo cáo kết quả phân loại và ma trận nhiệt như hình dưới đây:

Hình 26. Bảng báo cáo kết quả và ma trận nhiệt của mô hình FromScratch trên tập dữ liệu kiểm thử

Accuracy (From Scratch): 0.915658

	precision	recall	f1-score	support
1	0.93	0.91	0.92	1900
2	0.95	0.98	0.97	1900
3	0.89	0.88	0.88	1900
4	0.90	0.89	0.89	1900
accuracy			0.92	7600
macro avg	0.92	0.92	0.92	7600
weighted avg	0.92	0.92	0.92	7600



PHẦN 5: SO SÁNH VÀ PHÂN TÍCH

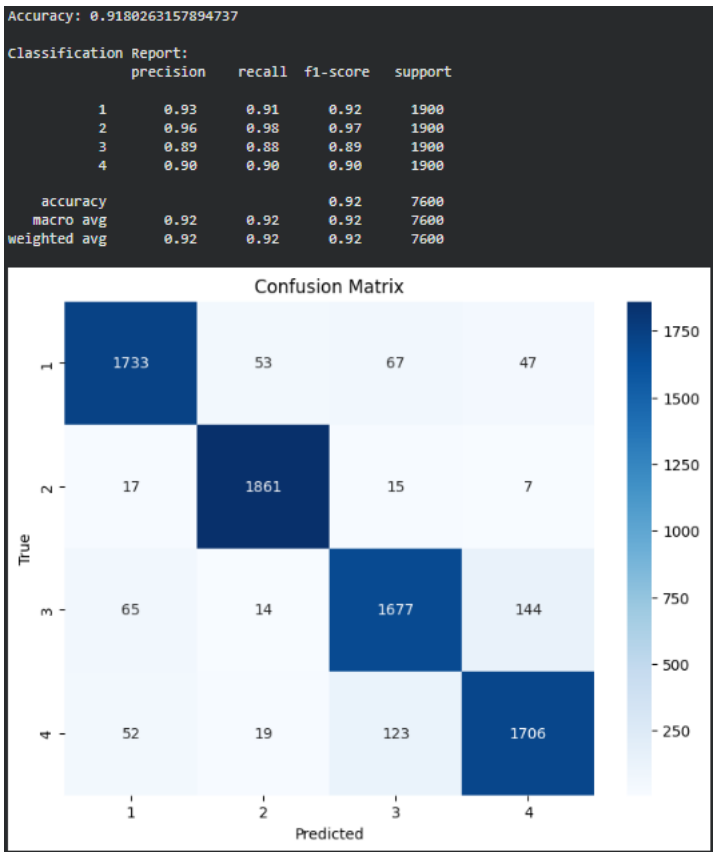
5.1. Bản baseline cài đặt bằng sự hỗ trợ của thư viện

Để đánh giá tính hiệu quả và độ chính xác của mô hình Logistic Regression tự cài đặt, việc xây dựng một hệ thống cơ sở (Baseline) là điều bắt buộc. Trong bài tập này, em thực hiện sử dụng thư viện Scikit-learn – một trong những thư viện học máy phổ biến và tối ưu nhất hiện nay cho ngôn ngữ Python. Cụ thể:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

baseline_model = LogisticRegression(max_iter=2000)
baseline_model.fit(X_train_tfidf, y_train_sub)
```

Hình 27. Cài đặt baseline bằng sự hỗ trợ của thư viện

Với baseline, Test ở dữ liệu kiểm thử thu được bảng báo cáo kết quả phân loại và ma trận nhiệt như hình bên đây:



Hình 28. Bảng báo cáo kết quả và ma trận nhiệt của mô hình baseline trên tập dữ liệu kiểm thử

5.2. So sánh mô hình tự cài đặt (Scratch) và Baseline (Scikit-learn)

Sau khi tiến hành thực nghiệm trên tập dữ liệu kiểm thử (Test Set) gồm 7.600 mẫu tin tức, nhóm đã thu được các chỉ số định lượng cụ thể. Dưới đây là bảng tổng hợp đối chiếu hiệu năng giữa hai phương pháp tiếp cận:

a) Bảng so sánh tổng quan:

Chỉ số	Mô hình Scratch	Mô hình Baseline	Chênh lệch
Accuracy (Độ chính xác tổng)	0.9157	0.9180	-0.0023
Macro Average Precision	0.9200	0.9200	0.0000
Macro Average Recall	0.9200	0.9200	0.0000
Macro Average F1-Score	0.9200	0.9200	0.0000

b) Bảng so sánh chi tiết trên từng lớp dữ liệu (theo F1-score, đại diện cho sự cân bằng giữa Precision và Recall):

Lớp	Mô hình Scratch (F1-Score)	Mô hình Baseline (F1-Score)	Đánh giá mức độ hội tụ
World	0.92	0.92	Tương đương
Sports	0.97	0.97	Tương đương (rất cao)
Business	0.88	0.89	Scratch thấp hơn 1%
Sci/Tech	0.89	0.90	Scratch thấp hơn 1%

5.3. Phân tích

Dựa trên kết quả thu được từ các bảng số liệu trên, ta có thể rút ra những đánh giá chi tiết như sau:

** Ưu điểm:*

- Độ chính xác tiệm cận tối ưu: Độ chính xác 91,57% (thấp hơn baseline khoảng 0,23%) của mô hình Scratch chỉ ra rằng cấu trúc toán học là hoàn toàn tin cậy. Và thuật toán Gradient Descent kết hợp với One-vs-Rest hoạt động ổn định và phù hợp với không gian vector thưa của TF_IDF.
- Tính kiểm soát được: Việc tự cài đặt giúp dễ dàng tùy chỉnh các hàm mất mát và các kỹ thuật tối ưu và phát triển trong tương lai hơn.
- Khả năng phân loại đồng đều: So với baseline, chênh lệch F1-Score giữa các lớp là rất nhỏ => Thuật toán không bị bias vào một nhóm dữ liệu nào.

** Nhược điểm:*

- Tốc độ hội tụ chưa cao: Cần đến hơn 2000 epochs để đạt được độ chính xác tương đương Baseline. Nguyên nhân là do sử dụng Gradient Descent cơ bản, đây cũng là một lưu ý để em phát triển thêm trong tương lai.
- Phụ thuộc nhiều vào các siêu tham số: Cụ thể là chọn tốc độ học η. Nếu chọn quá lớn, mô hình dao động mạnh dẫn đến đẩy các tham số đi xa làm ảnh hưởng độ chính xác; còn nếu chọn quá nhỏ thì mô hình chắc là sẽ không kịp hội tụ trong 2000 epochs (? :v). Em mất nhiều thời gian cho việc lựa chọn $\eta = 0.8$.
- Chi phí tính toán của OvR: Ta phải huấn luyện 4 lớp độc lập => Khối lượng tính toán tăng 4 lần, nếu dữ liệu lớn hơn hoặc phức tạp hơn (nhiều đặc trưng hơn,…) thì mô hình sẽ gặp vấn đề về thời gian cũng như chi phí huấn luyện. Phần này có thể khắc phục được phần nào bằng phương pháp lập trình song song (ta có thêm 1 hướng để phát triển).
- Có sự nhầm lẫn giữa các lớp tương đồng: Nhận thấy ở lớp Business và Sci/Tech, F1-Score thấp hơn so với 2 lớp còn lại. Đây là nhược điểm của Logistic Regression khi xử lý các văn bản có bộ từ tương tự nhau. Ví dụ: Một bài báo về "Giá cổ phiếu của tập đoàn công nghệ Apple" sẽ chứa các từ khóa thuộc cả hai lĩnh vực, khiến mô hình dễ đưa ra dự đoán nhầm lẫn.

=> Nhận xét: Tuy còn nhiều nhược điểm nhưng mô hình Logistic Regression (Scratch) khá hiệu quả cho bài toán phân loại tin tức AG News. Lý do là vì, các ma trận thưa từ TF_IDF vốn là thế mạnh của các mô hình tuyến tính; chi phí triển khai thấp và không yêu cầu GPU quá mạnh mẽ.

PHẦN 6: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Thông qua bài tập lớn này, em đã xây dựng thành công mô hình phân loại tin tức dựa trên thuật toán Logistic Regression được cài đặt thủ công hoàn toàn từ đầu (From Scratch). Việc tự cài đặt giúp em hiểu chắc hơn về quy trình vận hành của một thuật toán Machine Learning.

Để khắc phục những nhược điểm còn tồn tại và nâng cao hiệu suất của mô hình, em dự kiến tập trung phát triển bài toán theo các hướng sau:

1. Tự cài đặt lại quy trình chuyển dạng văn bản sang dạng vector (phiên bản 2 được trình bày trong bài báo cáo này). Mục tiêu trước hết là để nâng cao kỹ năng lập trình và hiểu vững chắc về phương pháp, sau đó là cố gắng tìm cách tối ưu để có thể phát triển tăng độ chính xác của mô hình.
2. Cải thiện các thuật toán tối ưu: Hướng đến triển khai các thuật toán tối ưu hóa bậc cao hơn như SGD (Stochastic Gradient Descent) thay vì chỉ dùng Gradient Descent cơ bản. Kỹ thuật này giúp mô hình hội tụ nhanh hơn, chính xác hơn.
3. Áp dụng phương pháp lập trình song song đối với chiến lược One-vs-Rest.
4. Ngoài phương pháp xử lý TF_IDF, em sẽ nghiên cứu thêm về các không gian vector dày đặc (dense vector). Hy vọng điều này sẽ khắc phục được tình trạng mô hình nhầm lẫn giữa các lớp có từ ngữ tương đồng.

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành đến thầy và các bạn trong lớp là những người đã giúp đỡ, đồng hành cùng em trong suốt quá trình học tập. Sự động viên và đóng góp của mọi người là nguồn động lực lớn để em có thể hoàn thành bài tập này. Bài báo cáo chắc chắn vẫn còn nhiều thiếu sót em rất mong nhận được sự góp ý từ Thầy để em có thể tiếp tục phát triển bài tập này nói riêng và phát triển bản thân trong các dự án học thuật tương lai nói chung.

Em xin kính chúc thầy nhiều sức khỏe, công tác tốt và luôn thành công trong hành trình nghiên cứu và giảng dạy cao quý của mình!

NGUỒN DỮ LIỆU, TÀI LIỆU THAM KHẢO

[1] A. Rai, "AG News Classification Dataset," *Kaggle*, 2021. [Online]. Available: https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset. [Accessed: Jan. 14, 2026].

[2] Trần Quốc Long, Giáo trình Học máy thống kê, Nxb Đại học Quốc gia Hà Nội, 2020.

[3] Machine Learning Cơ Bản, "Bài 10: Logistic Regression," 2017. [Online]. Available: https://machinelearningcoban.com/2017/01/27/logisticregression/. [Accessed: 14-Jan-2026].

[4] GeeksforGeeks, "Understanding TF-IDF (Term Frequency-Inverse Document Frequency)," 2025. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/understanding-tf-idf-term-frequency-inverse-document-frequency/. [Accessed: 14-Jan-2026].

[5] Lưu.name.vn, "Phương pháp One-vs-One và One-vs-Rest cho các bài toán phân loại đa lớp (Multi-Class Classification)," 2022. [Online]. Available: https://luu.name.vn/phuong-phap-one-vs-one-va-one-vs-rest-cho-cac-bai-toan-phan-loai-da-lop-multi-class-classification/. [Accessed: 14-Jan-2026].

[6] Machine Learning Cơ Bản, "Bài 12: Gradient Descent," 2017. [Online]. Available: https://machinelearningcoban.com/2017/01/12/gradientdescent/. [Accessed: 14-Jan-2026].

[7] GeeksforGeeks, "Machine Learning – Learning Rate Decay," 2026. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/learning-rate-decay/. [Accessed: 14-Jan-2026].

[8] Q. Tran, "Mini-batch Gradient Descent," 2020. [Online]. Available: https://tnquangblog.wordpress.com/2020/10/12/mini-batch-gradient-descent/. [Accessed: 14-Jan-2026]