lab5_rnns_text_classification

Harito ID

2025-11-06

Lab 5: Phân loại Văn bản với Mạng Nơ-ron Hồi quy (RNN/LSTM)

Mục tiêu:

- Hiểu rõ hạn chế của các mô hình phân loại văn bản truyền thống (Bag-of-Words, Word2Vec trung bình).
- Nắm vững kiến trúc và luồng hoạt động của pipeline sử dụng RNN/LSTM cho bài toán phân loại văn bản.
- Tự tay xây dựng, huấn luyện và so sánh hiệu năng giữa các mô hình:
  1. TF-IDF + Logistic Regression (Baseline 1).
  2. Word2Vec (vector trung bình) + Dense Layer (Baseline 2).
  3. Embedding Layer (pre-trained) + LSTM.
  4. Embedding Layer (học từ đầu) + LSTM.
- Phân tích và đánh giá sức mạnh của mô hình chuỗi trong việc "hiểu" ngữ cảnh của câu.

Đối tượng: Sinh viên lớp `Xử lý ngôn ngữ tự nhiên và ứng dụng` đã có kiến thức nền tảng về Word2Vec và các mô hình phân loại cơ bản.

Bộ dữ liệu: `hwu.tar.gz` (chứa các câu truy vấn người dùng và ý định tương ứng).

PHẦN 1: NỀN TẢNG LÝ THUYẾT

1. Ôn tập & Đặt vấn đề: Hạn chế của các Pipeline Cũ

Trong bài lab trước (`lab5_text_classification.pdf`), chúng ta đã tiếp cận bài toán phân loại văn bản với hai hướng chính:

- **Mô hình Bag-of-Words (Túi từ):** Biểu diễn mỗi văn bản bằng một vector tần suất từ (TF-IDF), sau đó dùng các thuật toán Machine Learning cổ điển như Logistic Regression, SVM.
- **Mô hình Word2Vec + Dense Layer:** Biểu diễn mỗi từ bằng một vector dày đặc (dense vector), sau đó tính vector trung bình cho cả câu và đưa vào một mạng nơ-ron đơn giản.

Vấn đề then chốt: Cả hai phương pháp trên đều bỏ qua một yếu tố cực kỳ quan trọng: **thứ tự và ngữ cảnh của từ**. Chúng coi một câu như một "túi" chứa các từ độc lập.

Hãy xem xét hai câu sau:

1. “Sản phẩm này chất lượng tốt, **không** hề tệ chút nào.”
2. “Sản phẩm này chất lượng **không** hề tốt, rất tệ.”

Với mô hình Bag-of-Words, hai câu này có thể có biểu diễn vector rất giống nhau vì chúng chứa cùng các từ khóa (“sản phẩm”, “chất lượng”, “tốt”, “tệ”, “không”). Mô hình rất dễ bối rối và đưa ra dự đoán sai. Đây là lúc chúng ta cần một kiến trúc có khả năng “đọc” câu theo trình tự.

2. Ý tưởng Đột phá: Mô hình Hóa Thứ tự với RNN/LSTM

RNN (Recurrent Neural Network) ra đời để giải quyết vấn đề này. Ý tưởng cốt lõi của RNN là có một “trạng thái ẩn” (hidden state) được luân chuyển qua từng bước thời gian (từng token).

Có thể hình dung lớp RNN như một vòng lặp `for`:

```
hidden_state = initial_state
for token in sequence:
  output, hidden_state = rnn_cell(token, hidden_state)
```

`hidden_state` hoạt động như một “biến tích lũy” thông minh, nó ghi nhớ thông tin từ các token đã đi qua để xử lý token hiện tại. Nhờ vậy, mô hình có thể nắm bắt được ngữ cảnh của cả câu.

Tuy nhiên, RNN cơ bản gặp vấn đề “Tiêu biến/Bùng nổ Gradient” (Vanishing/Exploding Gradient), khiến nó khó học được các phụ thuộc xa (ví dụ từ “không” ở đầu câu ảnh hưởng đến từ “tốt” ở cuối câu).

LSTM (Long Short-Term Memory) là một biến thể nâng cao của RNN, được thiết kế để giải quyết vấn đề trên. Nó sử dụng các “cổng” (gates) để kiểm soát luồng thông tin, quyết định xem thông tin nào cần được **lưu lại**, **quên đi**, hoặc **đưa ra** ở mỗi bước.

(Để hiểu sâu hơn về cơ chế hoạt động của RNN/LSTM và các cổng, sinh viên được yêu cầu xem lại bài giảng `lab5_rnn_token_classification.pdf`).

3. Triển khai Embedding Layer: Từ Lý thuyết đến Thực tế

Để đưa văn bản vào mô hình nơ-ron, chúng ta cần chuyển các token thành vector số.

- **Lý thuyết:** Mỗi token được biểu diễn bằng một vector one-hot (ví dụ, có kích thước `1 x 50000` nếu bộ từ vựng có 50000 từ). Sau đó, nhân vector này với một ma trận trọng số `W` (kích thước `50000 x 300`) để thu được vector embedding (`1 x 300`).
- **Thực tế (Lập trình):** Phép nhân ma trận với vector one-hot cực kỳ lãng phí. Thay vào đó, các thư viện như Keras/PyTorch triển khai `Embedding Layer` như một **bảng tra cứu (lookup table)**.
  1. Mỗi token được gán một chỉ số nguyên (integer index). Ví dụ: `{"PAD": 0, "UNK": 1, "Con": 2, "mèo": 3, "đen": 4}`.
  2. Với câu “Con mèo đen tên là La Tiểu Hắc”, giả sử vocab có các từ trên và số token tối đa là 10, đầu vào cho Embedding Layer sẽ là chuỗi chỉ số đã được đệm (padded): `[2, 3, 4, 1, 1, 1, 1, 1, 0, 0]`.
  3. `Embedding Layer` sẽ trực tiếp dùng các chỉ số này để “tra” vector tương ứng trong ma trận trọng số của nó.

Đầu vào của `Embedding Layer` là một ma trận số nguyên kích thước `(batch_size, max_sequence_length)`, và đầu ra là một tensor số thực kích thước `(batch_size,`

`max_sequence_length, embedding_dim)`.

Lớp này có thể được:

1. **Học từ đầu (Train from scratch):** Trọng số của lớp được khởi tạo ngẫu nhiên và cập nhật trong quá trình huấn luyện mô hình.
2. **Sử dụng Pre-trained Embeddings:** Khởi tạo trọng số bằng một ma trận embedding đã được huấn luyện trước (Word2Vec, GloVe, FastText) trên một kho dữ liệu khổng lồ.

4. Pipeline Phân loại Văn bản với LSTM

Luồng xử lý end-to-end của mô hình mới sẽ như sau:

1. **Input Text:** Một câu văn bản thô.
2. **Preprocessing:**
   - `Tokenizer`: Tách câu thành các token.
   - `Vocabulary`: Xây dựng bộ từ vựng và chuyển token thành chỉ số.
   - `Padding`: Thêm các giá trị "0" để đảm bảo mọi chuỗi chỉ số có cùng độ dài.
3. **Embedding Layer:** Chuyển chuỗi chỉ số thành chuỗi vector dày đặc.
4. **LSTM Layer:** Xử lý chuỗi vector để nắm bắt ngữ cảnh, output ra một vector biểu diễn cuối cùng cho cả câu.
5. **Dense Layer (Output):** Nhận vector biểu diễn câu từ LSTM và phân loại ra lớp tương ứng (ví dụ: dùng `softmax` cho bài toán đa lớp).

PHẦN 2: LAB THỰC HÀNH

Bước 0: Thiết lập Môi trường và Tải Dữ liệu

Đầu tiên, hãy giải nén bộ dữ liệu và tải nó bằng Pandas.

```
# Lệnh shell để giải nén file
!tar -xzvf data/hwu.tar.gz

import pandas as pd

# Dữ liệu có thể được phân tách bằng tab và không có header
df_train = pd.read_csv('hwu_train.csv', sep='\t', header=None, names=['text', 'intent'])
df_val = pd.read_csv('hwu_val.csv', sep='\t', header=None, names=['text', 'intent'])
df_test = pd.read_csv('hwu_test.csv', sep='\t', header=None, names=['text', 'intent'])

print("Train shape:", df_train.shape)
print("Validation shape:", df_val.shape)
print("Test shape:", df_test.shape)
df_train.head()
```

Bạn cũng cần tiền xử lý các nhãn (intent) để chuyển chúng thành dạng số.

```
from sklearn.preprocessing import LabelEncoder

# ... (Code để fit LabelEncoder trên toàn bộ tập intent và transform các tập train/val
```

Nhiệm vụ 1: (Warm-up Ôn bài cũ) Pipeline TF-IDF + Logistic Regression

Đây là mô hình baseline để chúng ta có một cơ sở so sánh. Hãy áp dụng lại kiến thức từ lab trước.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.metrics import classification_report

# 1. Tạo một pipeline với TfidfVectorizer và LogisticRegression
tfidf_lr_pipeline = make_pipeline(
    TfidfVectorizer(max_features=5000),
    LogisticRegression(max_iter=1000)
)

# 2. Huấn luyện pipeline trên tập train
# ...

# 3. Đánh giá trên tập test
# y_pred = tfidf_lr_pipeline.predict(...)
# print(classification_report(...))
```

Nhiệm vụ 2: (Warm-up Ôn bài cũ) Pipeline Word2Vec (Trung bình) + Dense Layer

Mô hình baseline thứ hai, sử dụng embedding nhưng chưa có khả năng xử lý chuỗi.

```
import numpy as np
from gensim.models import Word2Vec
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# 1. Huấn luyện mô hình Word2Vec trên dữ liệu text của bạn
# sentences = [text.split() for text in df_train['text']]
# w2v_model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)

# 2. Viết hàm để chuyển mỗi câu thành vector trung bình
def sentence_to_avg_vector(text, model):
    # ... (Implement logic)
    return avg_vector

# 3. Tạo dữ liệu train/val/test X_train_avg, X_val_avg, X_test_avg
# ...

# 4. Xây dựng mô hình Sequential của Keras
# model = Sequential([
#     Dense(128, activation='relu', input_shape=(w2v_model.vector_size,)),
#     Dropout(0.5),
#     Dense(num_classes, activation='softmax')
# ])
```

```
# 5. Compile, huấn luyện và đánh giá mô hình
# ...
```

Nhiệm vụ 3: Mô hình Nâng cao (Embedding Pre-trained + LSTM)

Đây là nhiệm vụ chính đầu tiên. Chúng ta sẽ sử dụng Word2Vec đã huấn luyện ở Nhiệm vụ 2 để khởi tạo trọng số cho `Embedding Layer`.

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, LSTM

# 1. Tiền xử lý cho mô hình chuỗi
# a. Tokenizer: Tạo vocab và chuyển text thành chuỗi chỉ số
# tokenizer = Tokenizer(num_words=vocab_size, oov_token="<UNK>")
# tokenizer.fit_on_texts(...)
# train_sequences = tokenizer.texts_to_sequences(...)

# b. Padding: Đảm bảo các chuỗi có cùng độ dài
# max_len = 50
# X_train_pad = pad_sequences(train_sequences, maxlen=max_len, padding='post')
# ... (Tương tự cho val và test)

# 2. Tạo ma trận trọng số cho Embedding Layer từ Word2Vec
# vocab_size = len(tokenizer.word_index) + 1
# embedding_dim = w2v_model.vector_size
# embedding_matrix = np.zeros((vocab_size, embedding_dim))
# for word, i in tokenizer.word_index.items():
#     if word in w2v_model.wv:
#         embedding_matrix[i] = w2v_model.wv[word]

# 3. Xây dựng mô hình Sequential với LSTM
# lstm_model_pretrained = Sequential([
#     Embedding(
#         input_dim=vocab_size,
#         output_dim=embedding_dim,
#         weights=[embedding_matrix],  # Khởi tạo trọng số
#         input_length=max_len,
#         trainable=False  # Đóng băng lớp Embedding
#     ),
#     LSTM(128, dropout=0.2, recurrent_dropout=0.2),
#     Dense(num_classes, activation='softmax')
# ])

# 4. Compile, huấn luyện (sử dụng EarlyStopping) và đánh giá
# ...
```

Nhiệm vụ 4: Mô hình Nâng cao (Embedding học từ đầu + LSTM)

Lần này, chúng ta sẽ để mô hình tự học lớp Embedding. Kiến trúc gần như tương tự Nhiệm vụ 3, nhưng `Embedding Layer` sẽ được học từ đầu.

```
# Dữ liệu đã được tiền xử lý (tokenized, padded) từ nhiệm vụ 3

# 1. Xây dựng mô hình
# lstm_model_scratch = Sequential([
#     Embedding(
#         input_dim=vocab_size,
#         output_dim=100,  # Chọn một chiều embedding, ví dụ 100
#         input_length=max_len
#         # Không có weights, trainable=True (mặc định)
#     ),
#     LSTM(128, dropout=0.2, recurrent_dropout=0.2),
#     Dense(num_classes, activation='softmax')
# ])

# 2. Compile, huấn luyện và đánh giá mô hình
# ...
```

Lưu ý quan trọng:

- **Tính nhất quán:** Đảm bảo các bước tiền xử lý (đặc biệt là `vocab_size` và `max_len`) được áp dụng đồng nhất cho các mô hình 3 và 4.
- **Overfitting:** Sử dụng tập validation và callback `EarlyStopping` của Keras để dừng huấn luyện khi hiệu năng trên tập validation không còn cải thiện, tránh lãng phí thời gian và overfitting.
- **Dropout:** Là một kỹ thuật regularization hiệu quả cho các mô hình nơ-ron, giúp ngăn ngừa overfitting.

Nhiệm vụ 5: Đánh giá, So sánh và Phân tích

Sau khi huấn luyện cả 4 mô hình, hãy thực hiện các bước sau:

1. So sánh định lượng: Tạo một bảng tổng hợp kết quả F1-score (macro) và loss trên tập kiểm tra (`test set`) của cả 4 pipeline. Sử dụng "macro" F1-score là rất quan trọng để đánh giá hiệu năng của mô hình trên các lớp thiểu số (ít mẫu), vì nó tính trung bình F1 của mỗi lớp mà không quan tâm đến số lượng mẫu của lớp đó.

Pipeline	F1-score (Macro)	Test Loss
TF-IDF + Logistic Regression	?	N/A
Word2Vec (Avg) + Dense	?	?
Embedding (Pre-trained) + LSTM	?	?
Embedding (Scratch) + LSTM	?	?

2. Phân tích định tính: Đây là phần quan trọng nhất để thấy sức mạnh của LSTM. Chọn ra một vài câu "khó" từ tập test, đặc biệt là những câu có yếu tố phủ định hoặc cấu trúc phức tạp.

Ví dụ các câu để kiểm tra:

- "can you remind me to not call my mom" (ý định: `reminder_create`)
- "is it going to be sunny or rainy tomorrow" (ý định: `weather_query`)
- "find a flight from new york to london but not through paris" (ý định: `flight_search`)

Với mỗi câu, hãy: a. Lấy dự đoán từ cả 4 mô hình. b. So sánh với nhãn thật. c. Nhận xét xem mô hình nào (đặc biệt là các mô hình LSTM) đưa ra dự đoán chính xác hơn và tại sao. Liệu khả năng xử lý chuỗi có giúp chúng hiểu đúng ý định trong các câu phức tạp này không?

YÊU CẦU NỘP BÀI

1. **Source Code:** Đầy đủ trên link GitHub.
2. **Báo cáo chi tiết:**
    - Link report GitHub.
    - Bảng so sánh kết quả định lượng (F1-score, loss).
    - Phần phân tích định tính cho các câu ví dụ điển hình (ví dụ 1 câu có nghĩa phụ thuộc xa), giải thích tại sao bạn nghĩ mô hình LSTM hoạt động tốt hơn (hoặc không tốt hơn) trong những trường hợp đó.
    - Nhận xét chung về ưu và nhược điểm của từng phương pháp.
    - Tham khảo các tiêu chí viết báo cáo ở các lab trước để đảm bảo chất lượng.