THỰC HÀNH 3: PHÂN LỚP VĂN BẢN CƠ BẢN

Mục tiêu: Xây dựng và huấn luyện một mô hình dùng cho bài toán phân loại văn bản.

Bài toán: phân tích cảm xúc bình luận phản hồi của người dùng.

Input: câu bình luận của người dùng.

Output: Nhãn cảm xúc, gồm 1 trong 3 nhãn: tích cực, tiêu cực và trung tính.

Bộ dữ liệu: **UIT-VSFC**

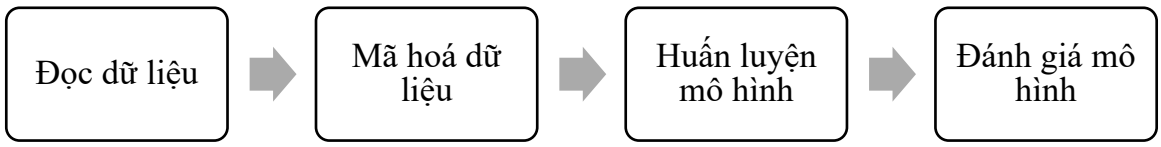
Công bố khoa học: K. V. Nguyen, V. D. Nguyen, P. X. V. Nguyen, T. T. H. Truong and N. L. Nguyen, "*UIT-VSFC: Vietnamese Students' Feedback Corpus for Sentiment Analysis*", KSE 2018.

Số lượng dữ liệu: khoảng hơn 16,000 câu.

Số lượng nhãn: 3 nhãn gồm tích cực (positive), tiêu cực (negative) và trung tính (neural).

Sử dụng cho tác vụ **sentiment-based** và **topic-based**.

Các bước thực hiện tổng quát:



1. Đọc dữ liệu

Dữ liệu được lưu trữ trên các file .txt, đã được phân chia sẵn thành các tập huấn luyện (train), phát triển (development) và kiểm thử (test).

Code đọc dữ liệu:

```
import pandas as pd

X_train = pd.read_csv('UIT-VSFC/train/sents.txt',
sep='\n', header=None, index_col=None)
y_train = pd.read_csv('UIT-VSFC/train/sentiments.txt',
sep='\n', header=None, index_col=None)

X_dev = pd.read_csv('UIT-VSFC/dev/sents.txt', sep='\n',
header=None, index_col=None)
y_dev = pd.read_csv('UIT-VSFC/dev/sentiments.txt',
sep='\n', header=None, index_col=None)

X_test = pd.read_csv('UIT-VSFC/test/sents.txt',
sep='\n', header=None, index_col=None)
```

```
y_test = pd.read_csv('UIT-VSFC/test/sentiments.txt',
sep='\n', header=None, index_col=None)

y_train = y_train.values.flatten()
y_dev = y_dev.values.flatten()
y_test = y_test.values.flatten()
```

2. Mã hoá dữ liệu

Sử dụng thư viện ***CountVectorizer*** để mã hoá văn bản thô ban đầu thành dạng vector tính tần suất xuất hiện của các từ trong văn bản.

Ngoài cách mã hoá bằng phương pháp đếm tần suất từ, còn một cách mã hoá khác được gọi là TF-IDF - đánh trọng số của một từ dựa vào mức độ quan trọng của từ đó đối với ngữ cảnh xung quanh (xem thêm về TF-IDF để biết được ý nghĩa cũng như cách đánh trọng số từ bằng TF-IDF). Để sử dụng phương pháp mã hoá theo TF-IDF, ta dùng thư viện ***TfidfVectorizertrong*** sklearn.

Ví dụ: Mã hoá văn bản ban đầu thành dạng vector tần suất xuất hiện sử dụng thư viện ***CountVectorizer*** trong sklearn:

```
from sklearn.feature_extraction.text import CountVectorizer

encoder = CountVectorizer(ngram_range=(2, 2))
encoder.fit(X_train[0])
```

Bộ mã hoá từ ***CountVectorizer*** đầu tiên sẽ xây dựng *tập từ vựng (vocabulary)* trên dữ liệu X_train, sau đó sẽ sử dụng tập từ vựng trên để tính tần suất xuất hiện của các từ trong văn bản ở các tập dữ liệu còn lại. Đoạn code phía trên đã xây dựng tập từ vựng trên dữ liệu X_train. Để lấy ra danh sách các từ vựng, ta dùng đoạn code sau:

```
encoder.vocabulary_
```

Sau khi đã xây dựng tập từ vựng, ta tiến hành mã hoá cho dữ liệu X_train, X_dev và X_test như sau:

```
X_train_encoded = encoder.transform(X_train[0])
X_dev_encoded = encoder.transform(X_dev[0])
X_test_encoded = encoder.transform(X_test[0])
```

Đối với nhãn (label / classes), ta có thể mã hoá về dạng số bằng công cụ ***LabelEncoder*** được hỗ trợ trong sklearn. Tuy nhiên, nhãn trong bộ dữ liệu **UIT-VSFC** đã được lưu

trữ dưới dạng số (đọc thêm file README trong bộ dữ liệu để biết thêm chi tiết) nên ta không cần thực hiện bước mã hoá này.

3. Huấn luyện mô hình và dự đoán

Các mô hình được chọn cho bài toán phân lớp: Naive Bayes, Logistic Regression, SVM, ...

Ví dụ: Huấn luyện mô hình Naive Bayes trên dữ liệu huấn luyện (train set).

```
from sklearn.naive_bayes import MultinomialNB

model = MultinomialNB()
model.fit(X_train_encoded, y_train)
```

Dự đoán nhãn và lưu vào biến *y_pred*:

```
y_pred = model.predict(X_test_encoded)
```

4. Đánh giá mô hình

Đánh giá khả năng dự đoán của mô hình: sử dụng các độ đo cho bài toán phân lớp như Accuracy, Precision, Recall, F1-score, ... (xem lại bài giảng về Phân lớp để biết thêm chi tiết). Trong ví dụ này sẽ sử dụng độ đo **Accuracy** và **macro F1-score** để đánh giá hiệu quả phân lớp của mô hình.

```
from sklearn.metrics import accuracy_score, f1_score

print(accuracy_score(y_test_encoded, y_pred)*100)
print(f1_score(y_test_encoded, y_pred, average='macro')*100)
```

Sử dụng **ma trận nhầm lẫn (confusion matrix)**. Xem lại bài giảng về phân lớp để biết thêm hoặc ôn lại kiến thức về ma trận nhầm lẫn.

```
from sklearn.metrics import confusion_matrix
cf = confusion_matrix(y_test_encoded, y_pred)
```

Trực quan hoá ma trận nhầm lẫn bằng thư viện seaborn

```
import seaborn as sn
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df_cm = pd.DataFrame(cf, range(3), range(3))
sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16}, fmt='d')

plt.show()
```

Từ ma trận nhầm lẫn, có thể rút ra nhận xét / đánh giá khả năng dự đoán của mô hình trên mỗi lớp cụ thể.

BÀI TẬP ÁP DỤNG

Bài 1: Thực hiện lại các bước như hướng dẫn để xây dựng mô hình phân lớp Naive Bayes cho bài toán phân loại cảm xúc người dùng dựa trên bộ dữ liệu UIT-VSFC.

Bài 2: Thay đổi cách mã hoá từ **CountVectorizer** thành **TfIdfVectorizer** ở Mục 2 (giữ nguyên n-gram). So sánh hiệu năng của mô hình Naive Bayes đối với 2 cách mã hoá (sử dụng độ đo macro F1-score và ma trận nhầm lẫn để so sánh).

Bài 3*: Sử dụng thêm mô hình Logistic Regression và SVM kết hợp với 2 phương pháp mã hoá CountVectorizer và TfIdfVectorizer để so sánh hiệu quả giữa các phương pháp. Độ đo đánh giá sử dụng: macro F1-score.

Gợi ý: Để thực hiện bài này, các bạn nên thiết kế kịch bản thử nghiệm như sau:

+ Thử nghiệm phương pháp CountVectorizer trên 3 mô hình SVM, Naive Bayes và Logistic Regression. Kết luận xem mô hình nào tốt nhất.

+ Lặp lại thử nghiệm trên với phương pháp TfidfVectorizer. So sánh với phương pháp trước đó trên từng mô hình.

+ Kết luận: với mô hình nào, và phương pháp nào thì cho ra hiệu năng mô hình cao nhất?

+ Phân tích lỗi: tìm hiểu nguyên nhân vì sao có các dự đoán sai.