



# **FINAL MACHINE LEARNING PROJECT**

**PHÂN LOẠI CHỦ ĐỀ BÀI  
BÁO DỰA VÀO TIÊU ĐỀ**

Thái Hoàng Nhân -  
18521182

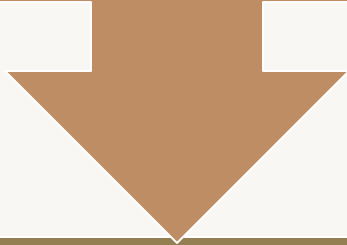
Đặt vấn đề:



Mạng xã hội ngày càng phát triển, các fanpage group xuất hiện ngày càng nhiều, với mục đích trao đổi, thảo luận, chia sẻ kinh nghiệm,... Với số lượng bài viết lớn và nhiều đề tài cần quản lý thì việc phân loại bài viết là hết sức cần thiết. Việc phân loại bài viết có thể được chính chúng ta đọc bài và phân chia vào từng loại chủ đề khác nhau, nhưng việc này tốn khá nhiều thời gian và công sức cũng như tiền bạc để chi trả cho các admin page.

## I- Mô tả bài toán:

Hướng giải quyết vấn đề:

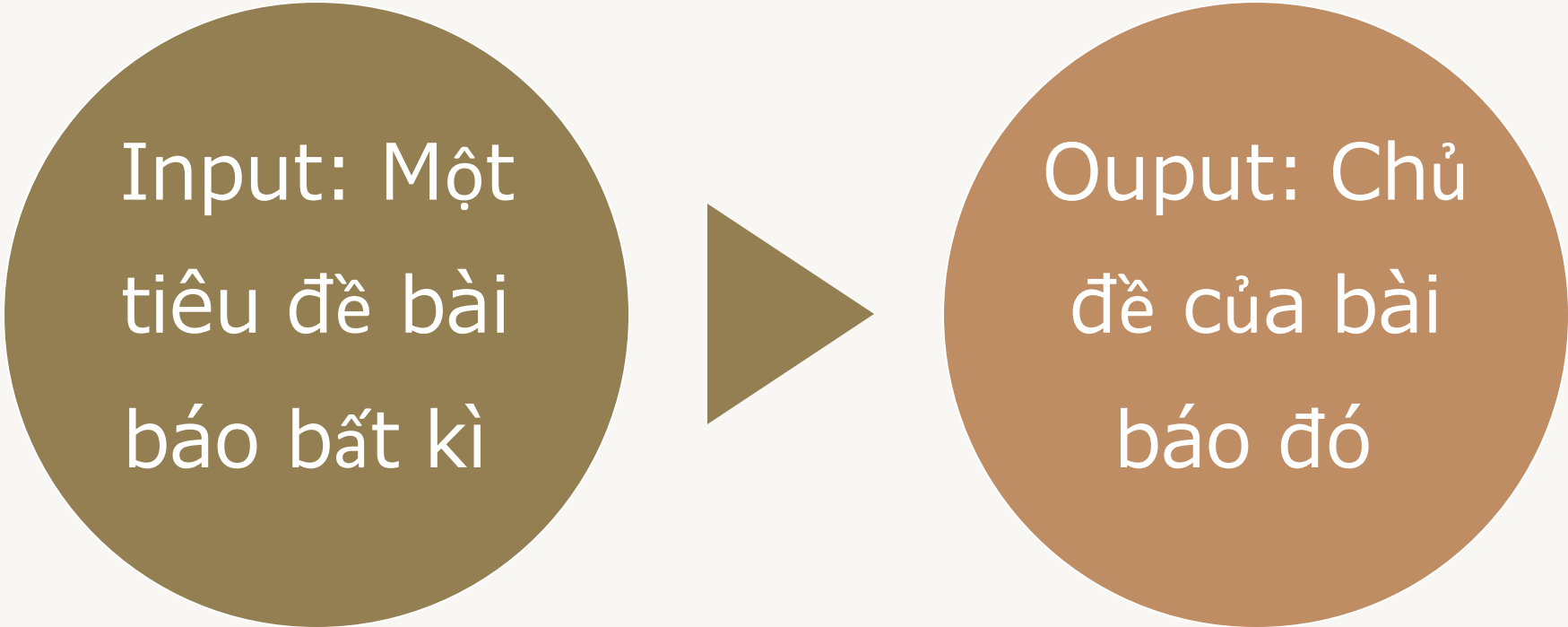


Mạng xã hội ngày càng phát triển, các fanpage group xuất hiện ngày càng nhiều, với mục đích trao đổi, thảo luận, chia sẻ kinh nghiệm,... Với số lượng bài viết lớn và nhiều đề tài cần quản lý thì việc phân loại bài viết là hết sức cần thiết. Việc phân loại bài viết có thể được chính chúng ta đọc bài và phân chia vào từng loại chủ đề khác nhau, nhưng việc này tốn khá nhiều thời gian và công sức cũng như tiền bạc để chi trả cho các admin page.

## I- Mô tả bài toán:

# I- Mô tả bài toán:

Mô tả mô hình:



Input: Một  
tiêu đề bài  
báo bất kì

Output: Chủ  
đề của bài  
báo đó

# II - Mô tả dữ liệu:

## 1) Phân lớp

Có 6 chủ đề bài viết

+ Chính trị

+ Công nghệ

+ Giáo dục

+ Pháp luật

+ Thể thao

+ Kinh doanh

# II - Mô tả dữ liệu:

## 2) Thu thập dữ liệu

---

Thu thập dữ liệu là các tiêu đề bài báo trên trang vietnamnet.vn

Sử dụng BeautifulSoup4 (bs4)

Chi tiết các bước thu thập dữ liệu:

- Khởi tạo hàm `get_content` (lấy content của trang web dưới dạng html)
- Sau đó, xem xét các tiêu đề nằm trong các mục nào, và tiến hành gọi hàm `find`, `findAll` và `text` để lấy tiêu đề
- Cứ duyệt như vậy qua nhiều trang của một chủ đề, ta sẽ được một lượng data (là các tiêu đề) khá lớn. Sau đó loại bỏ các bộ trùng bằng kiểu dữ liệu `set`.

## 2) Thu thập dữ liệu

Bước 1: Khởi tạo hàm để lấy content về

---

```
import bs4
import pandas
import requests
import gensim
from pyvi import ViTokenizer, ViPosTagger

def get_page_content(url):
    page = requests.get(url, headers={'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36'})
    return bs4.BeautifulSoup(page.text, "html.parser")
```

## 2) Thu thập dữ liệu:

Bước 2 và 3: Tiến hành thu thập dữ liệu:

---

*Ví dụ crawl data chính trị*

```
chinhtri_titles = []
for i in range(1, 501):
    url = "https://vietnamnet.vn/vn/thoi-su/chinh-tri/trang{}".format(i)    #Load từ trang 1 đến trang 500 mục chính trị
    soup = get_page_content(url)      #Lấy content từ một trang web
    h3_contents = soup.findAll('h3')   #Tìm mục h3

    new_titles = []
    for h3_content in h3_contents:
        if (h3_content.find('a')):     # Kiểm tra điều kiện để tránh bị lỗi trong quá trình crawl data
            text = h3_content.find('a').text # Trích xuất title
            new_titles.append(text)

    chinhtri_titles += new_titles
chinhtri_titles = list(set(chinhtri_titles)) # Loại bỏ các bộ trùng bằng kiểu dữ liệu set

print(len(chinhtri_titles))
```



## 2) Thu thập dữ liệu:

---

- Các chủ đề khác crawl tương tự, ta chỉ thay đổi đường dẫn đến chủ đề đó thôi.
- Sau khi thu thập xong data, ta sẽ lưu vào một file nào đó, để sử dụng sau này.
- Chi tiết thu thập dữ liệu ở [đây](#)

# II - Mô tả dữ liệu

## 3) Load và xử lý dữ liệu

---

- + Bước 1: Tiến hành load dữ liệu từ file đã lưu trước đó
- + Bước 2: Sử dụng gensim và ViTokenizer để tiến hành xử lý dữ liệu
  - Gensim: Xóa các kí tự đặc biệt, các kí tự không cần thiết, dấu câu
  - ViTokenizer: chuyển các chữ hoa về chữ thường, tách các từ trong câu theo từ điển tiếng Việt
- + Bước 3: Viết lệnh xóa bỏ các stop word để dataset được chất lượng hơn

### 3) Load và xử lý dữ liệu:

Bước 1 và 2:

---

```
from pyvi import ViTokenizer # thư viện NLP tiếng Việt
import numpy as np
import gensim #thư viện NLP
X = []
Y = []
train_data = []
for lines in thethao: #load data từ file the thao
    lines = gensim.utils.simple_preprocess(lines) # loại bỏ các kí tự đặc biệt, không cần thiết
    lines = ' '.join(lines)
    lines = ViTokenizer.tokenize(lines) # tách câu ra thành các từ tiếng việt
    lines = ' '.join(lines)

    X.append(lines)
    Y.append('thethao')
```

### 3) Load và xử lý dữ liệu:

Bước 3:

---

```
stop_word = open('/content/drive/My Drive/NLP_data/stopword.txt') #đọc file stop word
stop_word = stop_word.read() #đọc file stop word
stop_word = stop_word.split('\n')
stop_word = list(set(stop_word)) #lọc stop_word để tránh những stop word trùng nhau

#Bắt đầu xóa stop_word

for i in range(0, len(X)): # Load qua tất cả các headline có trong X
    s_new = ''
    for s in X[i].split(' '): # Tách headline ra thành các từ
        if s not in stop_word: #Xem xét từ đó có trong danh sách stop word hay không, nếu không thì giữ lại
            s_new = s_new + s + ' ' #Tạo headline mới với những stop word đã bị loại bỏ
    X[i] = s_new # Thay thế
```

# II - Mô tả dữ liệu:

## 4) Tổng kết dữ liệu

---

```
print(len(thethao))  
print(len(giaoduc))  
print(len(phapluat))  
print(len(congnghe))  
print(len(chinhtri))  
print(len(kinhdoanh))  
# kích thước các chủ đề
```

7499

7510

7516

7520

7145

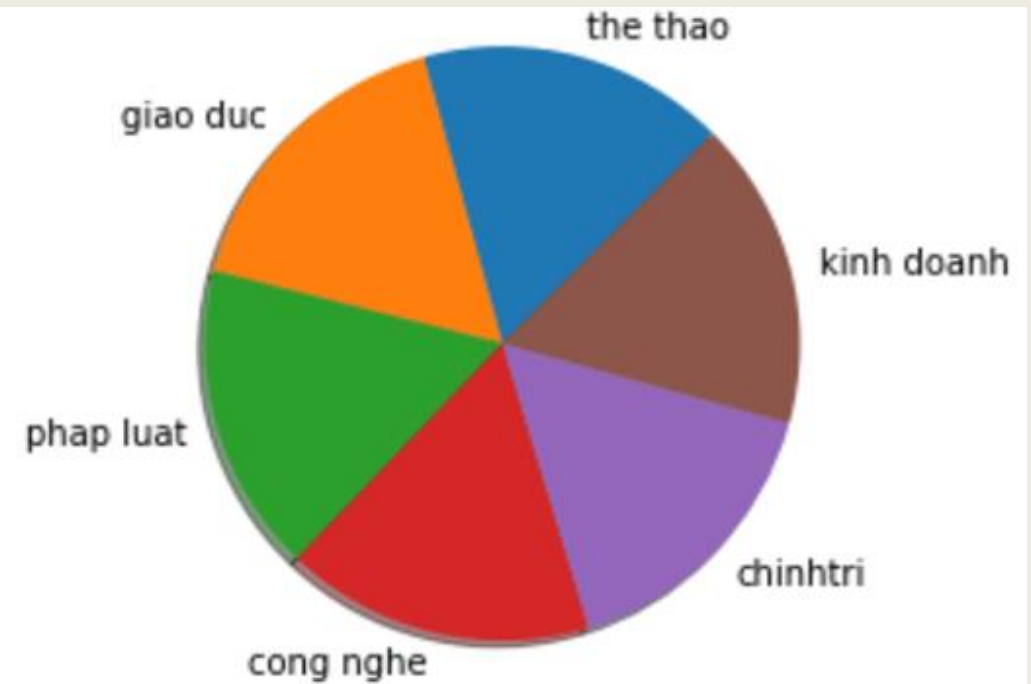
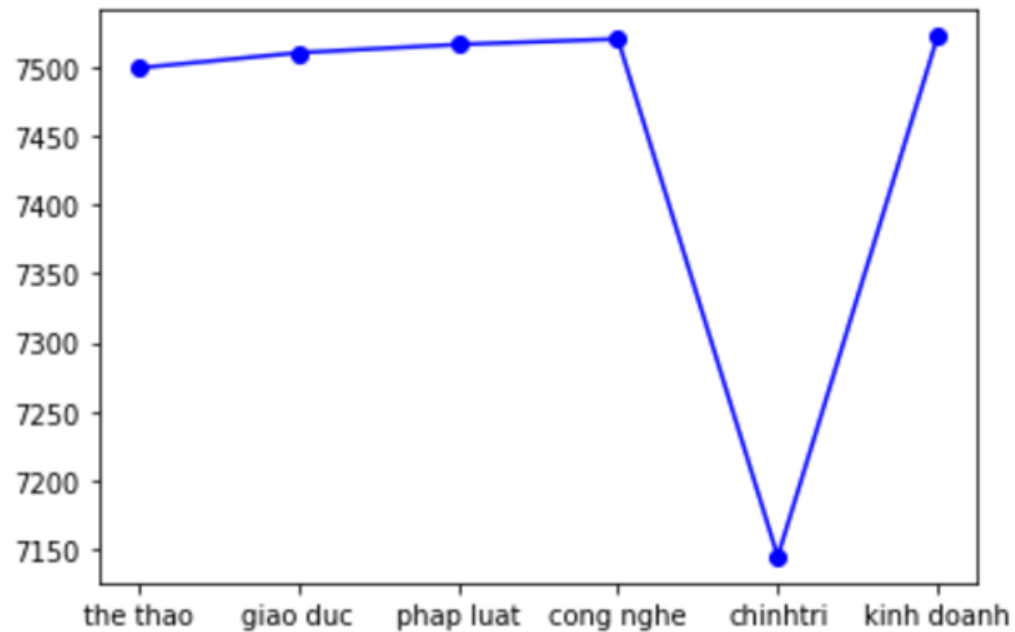
7522

## II - Mô tả dữ liệu:

### 4) Tổng kết và phân chia dữ liệu

---

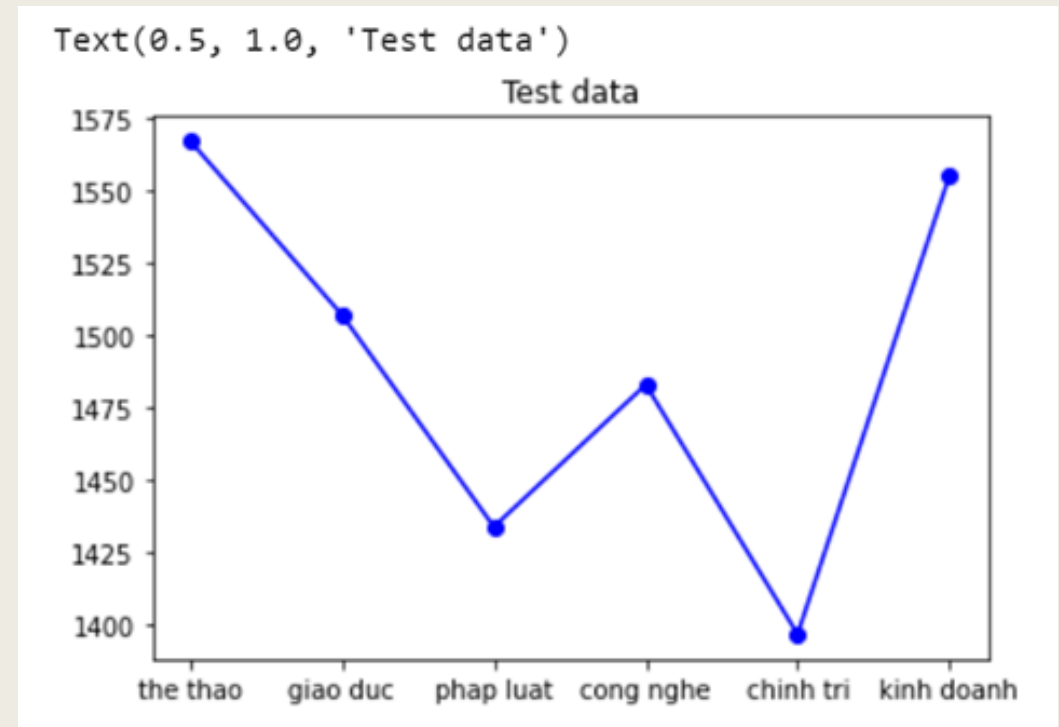
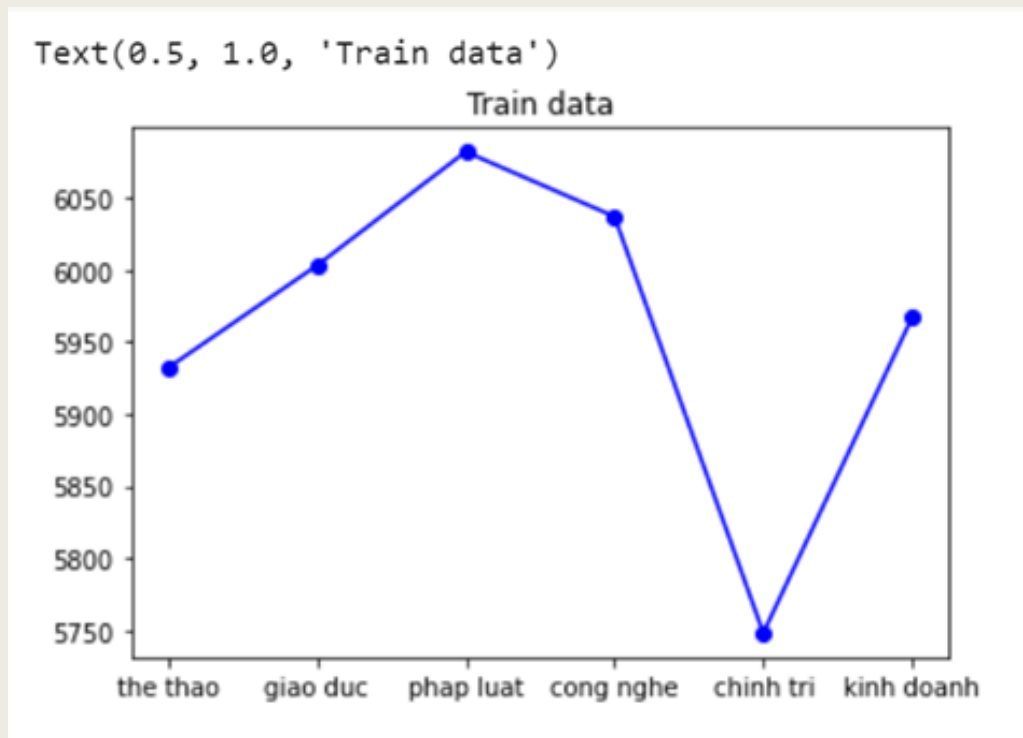
[<matplotlib.lines.Line2D at 0x7f53c812c438>]



## 4) Tổng kết và phân chia dữ liệu:

Sau khi phân chia train, test theo tỉ lệ 80%, 20% ta có biểu đồ

---



# III - Các bước tiến hành train data:

Có nhiều phương pháp phân tích dữ liệu text, phổ biến là dùng *TfidfVectorizer* và đơn giản là sử dụng *CountVecotrize*

## CountVecotrize

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer(analyzer='word', token_pattern=r'\w{1,}')
```

```
count_vect.fit(X_train) #tạo bộ từ vựng
X_train_countvec = count_vect.transform(X_train) #chuyển data dựa vào bộ từ vựng
X_test_countvec = count_vect.transform(X_test) #chuyển data dựa vào bộ từ vựng
```

## TfidfVectorize( Word Level )

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(analyzer='word', max_features=100000) #Lựa chọn số từ vào vocabulary
tfidf.fit(X_train) #tạo bộ từ vựng
X_train_tfidf_wordlevel = tfidf.transform(X_train) #chuyển data dựa vào bộ từ vựng
X_test_tfidf_wordlevel = tfidf.transform(X_test) #chuyển data dựa vào bộ từ vựng
```



# III - Các bước tiến hành train data:

Mã hóa label

---

Tiến hành mã hóa label

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
Y_train_encode = encoder.fit_transform(Y_train)
Y_test_encode = encoder.fit_transform(Y_test)
print(encoder.transform(['chinhtri', 'congnghe', 'giaoduc', 'kinhdoanh', 'phapluat', 'thethao']))
```

```
[0 1 2 3 4 5]
```

# III - Các bước tiến hành train data:

Model LinearSVC (sử dụng tfidf và count)

## LinearSVC

```
LSVC.fit(X_train_tfidf_wordlevel, Y_train_encode)
prediction=LSVC.predict(X_test_tfidf_wordlevel)
from sklearn.metrics import classification_report
print(classification_report(Y_test_encode, prediction))
```

	precision	recall	f1-score	support
0	0.90	0.90	0.90	1397
1	0.90	0.89	0.89	1483
2	0.91	0.92	0.92	1507
3	0.86	0.87	0.87	1555
4	0.94	0.93	0.93	1434
5	0.98	0.98	0.98	1567
accuracy			0.92	8943
macro avg	0.92	0.91	0.91	8943
weighted avg	0.92	0.92	0.92	8943

```
LSVC_countvec.fit(X_train_countvec, Y_train_encode)
prediction=LSVC_countvec.predict(X_test_countvec)
print(classification_report(Y_test_encode, prediction))
```

	precision	recall	f1-score	support
0	0.88	0.89	0.88	1397
1	0.88	0.88	0.88	1483
2	0.89	0.92	0.90	1507
3	0.86	0.84	0.85	1555
4	0.93	0.92	0.92	1434
5	0.98	0.97	0.98	1567
accuracy			0.90	8943
macro avg	0.90	0.90	0.90	8943
weighted avg	0.90	0.90	0.90	8943

# III - Các bước tiến hành train data:

Model LogisticRegression (sử dụng tfidf và count)

---

LogisticRegression

```
LR.fit(X_train_tfidf_wordlevel, Y_train_encode)
prediction=LR.predict(X_test_tfidf_wordlevel)
from sklearn.metrics import classification_report
print(classification_report(Y_test_encode, prediction))
```

	precision	recall	f1-score	support
0	0.89	0.89	0.89	1397
1	0.89	0.88	0.89	1483
2	0.92	0.91	0.91	1507
3	0.83	0.88	0.86	1555
4	0.94	0.92	0.93	1434
5	0.98	0.97	0.97	1567
accuracy			0.91	8943
macro avg	0.91	0.91	0.91	8943
weighted avg	0.91	0.91	0.91	8943

```
LR_countvec.fit(X_train_countvec, Y_train_encode)
prediction=LR_countvec.predict(X_test_countvec)
print(classification_report(Y_test_encode, prediction))
```

	precision	recall	f1-score	support
0	0.88	0.89	0.89	1397
1	0.89	0.87	0.88	1483
2	0.91	0.91	0.91	1507
3	0.84	0.86	0.85	1555
4	0.94	0.91	0.93	1434
5	0.98	0.97	0.98	1567
accuracy			0.90	8943
macro avg	0.91	0.90	0.90	8943
weighted avg	0.91	0.90	0.91	8943

# III - Các bước tiến hành train data:

Model RandomForestClassifier (sử dụng tfidf và count)

RandomForestClassifier

```
RFC.fit(X_train_tfidf_wordlevel, Y_train_encode)
prediction=RFC.predict(X_test_tfidf_wordlevel)
from sklearn.metrics import classification_report
print(classification_report(Y_test_encode, prediction))
```

	precision	recall	f1-score	support
0	0.86	0.87	0.87	1397
1	0.88	0.84	0.86	1483
2	0.90	0.89	0.89	1507
3	0.78	0.83	0.81	1555
4	0.88	0.88	0.88	1434
5	0.97	0.96	0.96	1567
accuracy			0.88	8943
macro avg	0.88	0.88	0.88	8943
weighted avg	0.88	0.88	0.88	8943

```
RFC_countvec.fit(X_train_countvec, Y_train_encode)
prediction=RFC_countvec.predict(X_test_countvec)
print(classification_report(Y_test_encode, prediction))
```

	precision	recall	f1-score	support
0	0.86	0.88	0.87	1397
1	0.88	0.84	0.86	1483
2	0.89	0.89	0.89	1507
3	0.79	0.83	0.81	1555
4	0.89	0.88	0.89	1434
5	0.97	0.96	0.96	1567
accuracy			0.88	8943
macro avg	0.88	0.88	0.88	8943
weighted avg	0.88	0.88	0.88	8943

# IV – Fine tuning:

1) Sử dụng TfidfVectorizer (ngram-word level) với TruncatedSVD:

---

TfidfVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_ngram = TfidfVectorizer(analyzer='word', max_features=100000, ngram_range=(2,3)) #lựa chọn số từ vào vocabulary
tfidf_ngram.fit(X_train) #tạo bộ từ vựng
X_train_tfidf_nwordlevel = tfidf_ngram.transform(X_train) #chuyển data dựa vào bộ từ vựng
X_test_tfidf_nwordlevel = tfidf_ngram.transform(X_test) #chuyển data dựa vào bộ từ vựng
```

Sử dụng TruncatedSVD nhằm giảm chiều dữ liệu của ma trận nhưng vẫn giữ nguyên các đặc trưng

```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=300, random_state=42) #Ta sẽ giảm số chiều xuống còn 300
svd.fit(X_train_tfidf_nwordlevel)
X_train_svd_nwordlevel = svd.transform(X_train_tfidf_nwordlevel)
X_test_svd_nwordlevel = svd.transform(X_test_tfidf_nwordlevel)
```

# IV – Fine tuning:

2) Sử dụng TfidfVectorizer (ngram-char level) với TruncatedSVD:

---

TfidfVectorize ngram-char level

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_ngram = TfidfVectorizer(analyzer='char', max_features=100000, ngram_range=(2,3)) #lựa chọn số từ vào vocabulary
tfidf_ngram.fit(X_train) #tạo bộ từ vựng
X_train_tfidf_ncharlevel = tfidf_ngram.transform(X_train) #chuyển data dựa vào bộ từ vựng
X_test_tfidf_ncharlevel = tfidf_ngram.transform(X_test) #chuyển data dựa vào bộ từ vựng
```

Sử dụng TruncatedSVD nhằm giảm chiều dữ liệu của ma trận nhưng vẫn giữ nguyên các đặc trưng

```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=300, random_state=42)
svd.fit(X_train_tfidf_ncharlevel)
X_train_svd_ncharlevel = svd.transform(X_train_tfidf_ncharlevel)
X_test_svd_ncharlevel = svd.transform(X_test_tfidf_ncharlevel)
```

# IV – Fine tuning:

3.a) Tiến hành train với mô hình LinearSVC (feature engineering ngram-word level và chưa giảm chiều dữ liệu)

```
LSVC_finetuning = LinearSVC()
LSVC_finetuning.fit(X_train_tfidf_nwordlevel, Y_train_encode)
prediction=LSVC_finetuning.predict(X_test_tfidf_nwordlevel)
from sklearn.metrics import classification_report
print(classification_report(Y_test_encode, prediction))
```

	precision	recall	f1-score	support
0	0.87	0.83	0.85	1397
1	0.68	0.84	0.75	1483
2	0.88	0.81	0.85	1507
3	0.78	0.74	0.76	1555
4	0.89	0.89	0.89	1434
5	0.95	0.89	0.92	1567
accuracy			0.83	8943
macro avg	0.84	0.83	0.84	8943
weighted avg	0.84	0.83	0.84	8943

# IV – Fine tuning:

3.b) Tiến hành train với mô hình LinearSVC (feature engineering ngram-word level và giảm chiều dữ liệu)

```
LSVC_finertuning = LinearSVC()
LSVC_finertuning.fit(X_train_svd_nwordlevel, Y_train_encode)
prediction=LSVC_finertuning.predict(X_test_svd_nwordlevel)
from sklearn.metrics import classification_report
print(classification_report(Y_test_encode, prediction))
```

	precision	recall	f1-score	support
0	0.85	0.63	0.73	1397
1	0.41	0.81	0.54	1483
2	0.85	0.60	0.70	1507
3	0.63	0.53	0.58	1555
4	0.83	0.75	0.79	1434
5	0.89	0.72	0.80	1567
accuracy			0.67	8943
macro avg	0.74	0.67	0.69	8943
weighted avg	0.74	0.67	0.69	8943



# IV – Fine tuning:

4) Tiến hành train với mô hình Deep Neural Network đơn giản:

---

Tiến hành chia train data và validation data theo tỉ lệ train 85% và validation 15%

Phân chia dữ liệu

```
X_train_new, X_val_new, Y_train_new, Y_val_new = train_test_split(X_train_svd_mwordlevel, Y_train_encode, test_size=0.15, random_state=42)
```

# IV – Fine tuning:

4) Tiến hành train với mô hình Deep Neural Network đơn giản:

---

Khởi tạo mô hình Deep Neural Network đơn giản:

```
#!/usr/bin/env python
import tensorflow
from tensorflow import keras
input_layer = Input(shape=(300,)) #Khởi tạo một layer input với shpe = (300,) vì ở giai đoạn giảm chiều dữ liệu, em đã giảm xuống còn 300
layer = Dense(512, activation='relu')(input_layer) #Hidden layer với số neuron trong layer, hàm kích hoạt là hàm relu
layer = Dense(512, activation='relu')(layer)
layer = Dense(256, activation='relu')(layer)
layer = Dense(128, activation='relu')(layer)
layer = Dense(64, activation='relu')(layer)
output_layer = Dense(6, activation='softmax')(layer) # Khởi tạo layer out_put, vì có 6 class nên số neuron hàm out_put là 6, hàm kích hoạt out_put sẽ là softmax

DNN = Model(input_layer, output_layer)
DNN.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

# IV – Fine tuning:

## 4) Tiến hành train với mô hình Deep Neural Network đơn giản:

Train mô hình vừa xây dựng với dataset sau khi đã giảm chiều dữ liệu:

```
DNN.fit(X_train_new, Y_train_new, validation_data=(X_val_new, Y_val_new), epochs=10, batch_size=512)

val_predictions = DNN.predict(X_val_new)
test_predictions = DNN.predict(X_test_svd_nwordlevel)
val_predictions = val_predictions.argmax(axis=-1)
test_predictions = test_predictions.argmax(axis=-1)

print("Validation accuracy: ", metrics.accuracy_score(val_predictions, Y_val_new)) #xác nhận chính xác
print("Test accuracy: ", metrics.accuracy_score(test_predictions, Y_test_encode)) #test chính xác

Epoch 1/10
60/60 [=====] - 3s 42ms/step - loss: 1.3201 - accuracy: 0.5043 - val_loss: 0.9824 - val_accuracy: 0.6223
Epoch 2/10
60/60 [=====] - 2s 41ms/step - loss: 0.9092 - accuracy: 0.6696 - val_loss: 0.8574 - val_accuracy: 0.6906
Epoch 3/10
60/60 [=====] - 2s 40ms/step - loss: 0.8271 - accuracy: 0.7012 - val_loss: 0.8134 - val_accuracy: 0.7136
Epoch 4/10
60/60 [=====] - 2s 40ms/step - loss: 0.7787 - accuracy: 0.7159 - val_loss: 0.8050 - val_accuracy: 0.7083
Epoch 5/10
60/60 [=====] - 2s 38ms/step - loss: 0.7409 - accuracy: 0.7301 - val_loss: 0.8053 - val_accuracy: 0.7067
Epoch 6/10
60/60 [=====] - 2s 38ms/step - loss: 0.7212 - accuracy: 0.7386 - val_loss: 0.7618 - val_accuracy: 0.7287
Epoch 7/10
60/60 [=====] - 2s 41ms/step - loss: 0.7137 - accuracy: 0.7371 - val_loss: 0.7683 - val_accuracy: 0.7294
Epoch 8/10
60/60 [=====] - 2s 39ms/step - loss: 0.7091 - accuracy: 0.7375 - val_loss: 0.8112 - val_accuracy: 0.7059
Epoch 9/10
60/60 [=====] - 2s 40ms/step - loss: 0.6976 - accuracy: 0.7414 - val_loss: 0.7698 - val_accuracy: 0.7216
Epoch 10/10
60/60 [=====] - 3s 43ms/step - loss: 0.6855 - accuracy: 0.7461 - val_loss: 0.7592 - val_accuracy: 0.7275
Validation accuracy: 0.7275437942601566
Test accuracy: 0.7321927764732192
```

# V – Nhận xét:

1) Chưa fine tuning:

---

	LinearSVC	LogisticRegression	RandomForest
CountVectorizer	90%	90%	88%
TfidfVectorizer	92%	91%	88%

- + Ta thấy TfidfVectorizer là phương pháp feature engineering đem lại hiệu quả cao hơn với accuracy các mô hình đều lớn hơn bằng so với sử dụng CountVectorizer. Đó là lí do tại sao ngày nay TfidfVectorizer được sử dụng phổ biến hơn CountVectorizer.
- + LinearSVC là mô hình đem lại hiệu quả dự đoán cao nhất trong 3 mô hình và cả trong 2 cách feature engineering.
- + Accuracy trong tất cả các trường hợp đều ổn, chủ yếu là nhờ dataset chất lượng, quá trình xử lý dữ liệu kĩ càng, và feature engineering phù hợp.

# V – Nhận xét:

## 2) Sau khi fine tuning:

---

- + Sau khi fine tuning, ta mong muốn đạt được một kết quả tốt hơn. Nhưng không, ta nhận lại điều hoàn toàn ngược lại. Sau khi tạo bộ từ vựng mới với `ngram_range=(2,3)` thì kết quả khi sử dụng mô hình `LinearSVC` không còn tốt như trước. Mà cụ thể là `accuracy` chỉ còn 88%.
- + Sau khi ta giảm chiều dữ liệu đi thì kết quả còn tệ hơn, sau khi dùng `LinearSVC` để huấn luyện và dự đoán, thì `accuracy` chỉ còn 67% => Cách này không ổn trong trường hợp này.
- + Dùng mô hình `DNN` cũng không khả quan cho lắm, `accuracy` cho tập `val` và `test` lần lượt là 72.8% và 73.2%
- + Có thể cách fine tuning của em chưa thực sự đạt hiệu quả, mà trái lại, nó đem đến nhiều bất lợi trong quá trình `train` và `predict`
- + Mô hình `DNN` chưa thực sự tốt với tập dữ liệu như thế này, cần phải khắc phục mô hình

**THANKS**

**FOR WATCHING**