ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

BÁO CÁO ĐỒ ÁN

MÁY HỌC (MACHINE LEARNING)

PHÂN LOẠI CHỦ ĐỀ BÀI BÁO DỰA VÀO TIÊU ĐỀ

GIẢNG VIÊN PHỤ TRÁCH:

GV. LÊ ĐÌNH DUY

GV. PHẠM NGUYỄN TRƯỜNG AN

SINH VIÊN THỰC HIỆN:

THÁI HOÀNG NHÂN – 18521182

TP HỒ CHÍ MINH - 8,2020

Mục lục:

I. Mô tả bài toán:

- 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.
- Hướng giải quyết vấn đề:
  Xây dựng mô hình Machine learning để dự đoán, phân loại các bài viết dựa vào tiêu đề thì sẽ giải quyết được nhiều vấn đề ở trên(tiết kiệm thời gian, công sức và tiền bạc).
- Mô tả mô hình;
  Input: Một tiêu đề bài báo bất kì
  Ouput: Chủ đề của bài báo đó

II. Mô tả dữ liệu

1) Chia lớp chủ đề:
   Có 6 chủ đề bài viết:
   + Chính trị
   + Công nghệ
   + Giáo dục
   + Pháp luật
   + Thể thao
   + Kinh doanh

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.

   Ví dụ:

   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) Appl
0.4103.106 Safari/537.36'})
   return bs4.BeautifulSoup(page.text,"html.parser")
```

Bước 2 và bước 3:

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))
```

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.

3) Load và xử lý dữ liệu:
   + Bước 1: Tiến hành load dữ liệu từ file đã lưu trước đó
   + Bước 2: Sử dụng genism 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

Ví dụ:

Bước 1 và bước 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')
```

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
```

4) Tổng kết và phân chia dữ liệu:

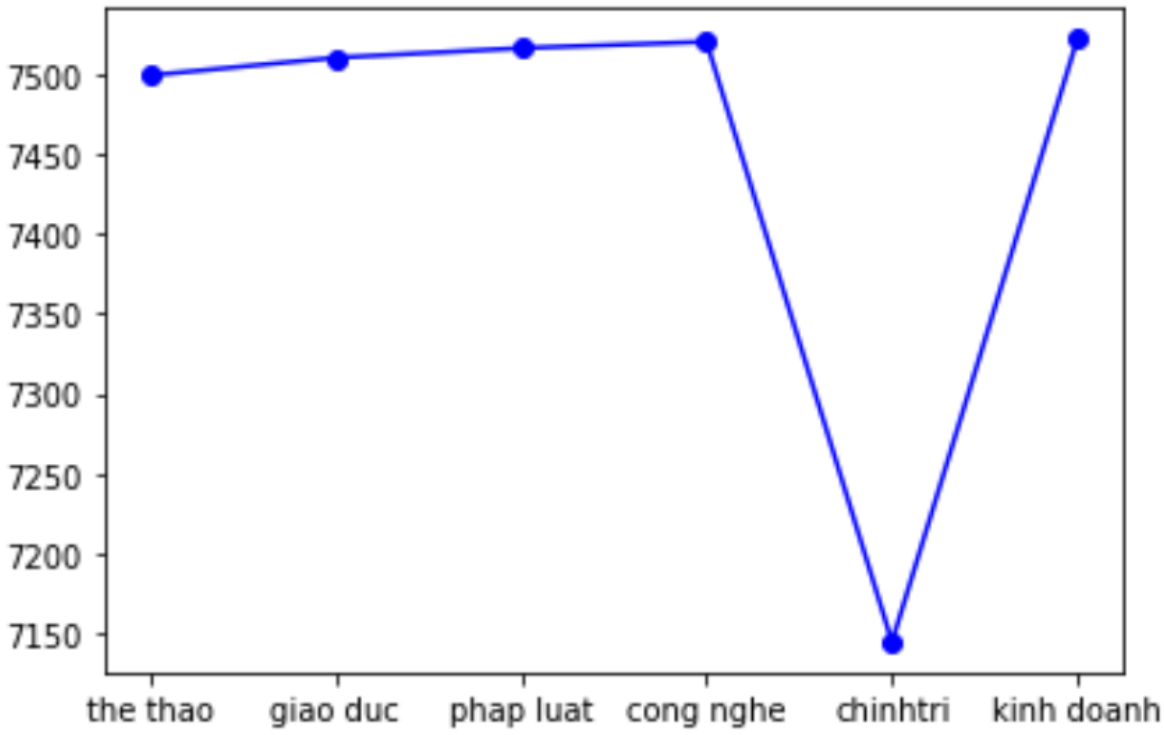
+ Mỗi chủ đề có hơn 7000 tiêu đề, tổng cộng dataset có hơn 44000 tiêu đề, phân bổ khá đồng đều ở mỗi chủ đề

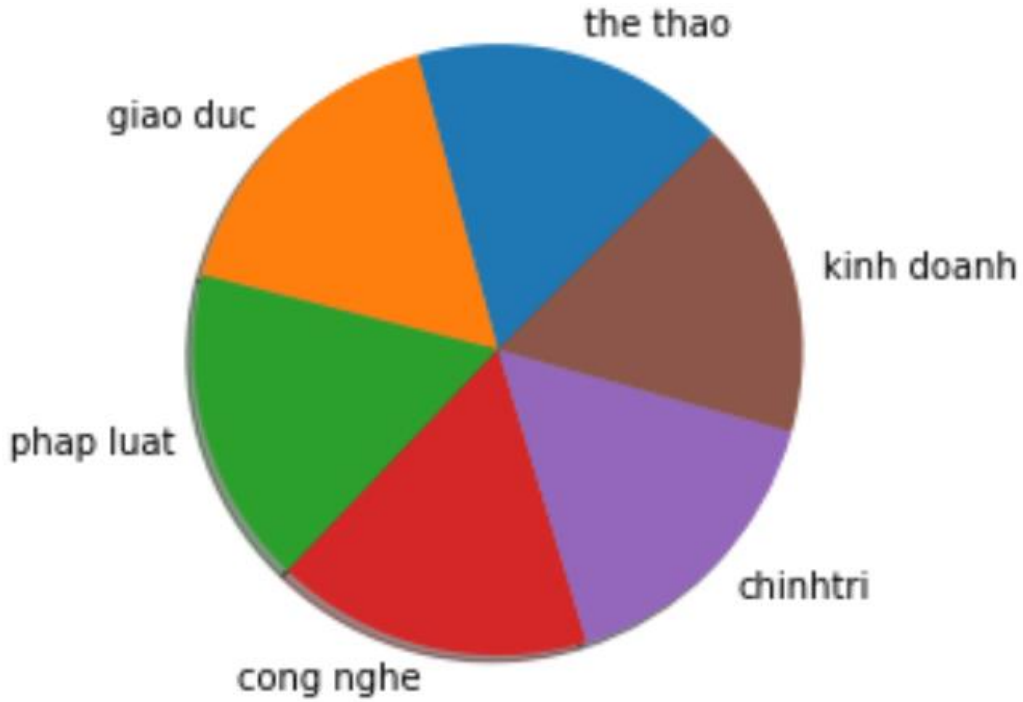
```
print(len(thethao))
print(len(giaoduc))
print(len(phapluat))
print(len(congnghe))
print(len(chinhtri))
print(len(kinhdoanh))
# kích thước các chủ đề
```

```
7499
7510
7516
7520
7145
7522
```

+ Biểu đồ tỉ lệ và số lượng tiêu đề có được được thể hiện bên dưới:

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





+ Phân chia dữ liệu theo tỉ lệ train: 80%, test: 20%:

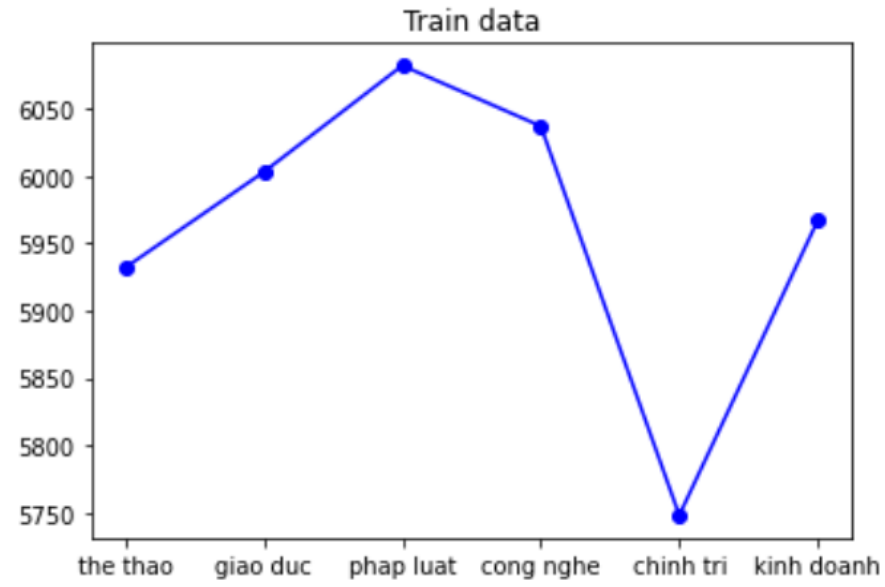
Dữ liệu được phân chia theo tỉ lệ 80% train, 20% test

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
print(len(X_train))
print(len(X_test))
```

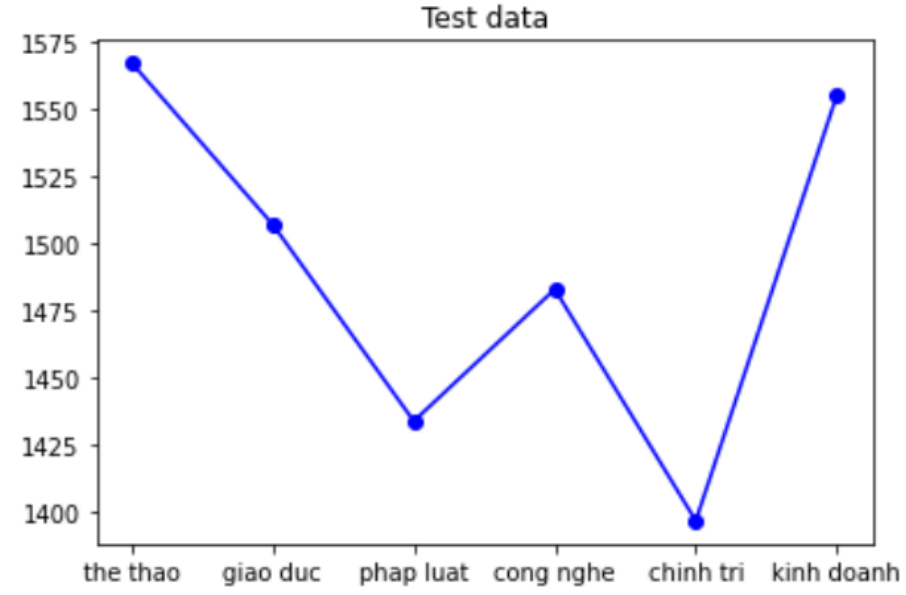
```
35769
8943
```

Sau khi phân chia ta có biểu đồ sau:

```
Text(0.5, 1.0, 'Train data')
```




```
Text(0.5, 1.0, 'Test data')
```



III. Các bước tiến hành train data

1) Feature Engineering

+ Có nhiều phương pháp phân tích dữ liệu text, phổ biến là dùng *TfidfVectorizer* và đơn giản là sử dụng *CountVecotrize*. Ta sẽ tiến hành tìm hiểu 2 phương pháp xử lí dữ liệu text này.

- CountVectorizer; Đây là phương pháp tính số lần xuất hiện của một từ trong văn bản. Khi thực hiện, ta sẽ thu được một kết quả trả về một vector biểu diễn các từ và số lần từ đó xuất hiện có trong văn bản.
- TfidfVectorizer: Đây là phương pháp tính trọng số của từ xuất hiện trong văn bản. Trọng số càng lớn thì độ quan trọng của từ đó càng cao.
  - TF: là tần số từ đó xuất hiện trong văn bản. Vì 1 văn bản cần xử lý có thể dài ngắn khác nhau, nên để thấy được độ quan trọng của từ đó, ta có thể tính bằng $\text{tf}(w, d) = \frac{f(w,d)}{\max(\{r\_w,d \mid r\_w \in d\})}$

    Trong đó:

    tf(t, d): tần suất xuất hiện của từ t trong văn bản d

    f(t, d): Số lần xuất hiện của từ t trong văn bản d

    max({r_w,d | r_w ∈ d}): Số lần xuất hiện của từ có số lần xuất hiện nhiều nhất trong văn bản d
  - IDF: giúp đánh giá độ quan trọng của một từ bằng cách tính log nghịch đảo của tỉ lệ số văn bản xuất hiện từ đang xét với tổng số văn bản. Được tính bởi công thức $\text{idf}(w, D) = \log \frac{|D|}{\{d \in D, w \in d\}}$

    Trong đó:

    idf(w, D): giá trị idf của từ w trong tập văn bản

    |D|: tổng số văn bản trong tập D

    {d ∈ D : t ∈ d}: số văn bản có từ w trong tập D

Ví dụ:

❖ CountVectorizer:

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
```

❖ TfidfVectorizer(word level):

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
```

+ Mã hóa label: Ta sẽ mã hóa các label về các con số

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]
```

2) Chọn model và train data:

+ Trong đồ án này, em sẽ tiến hành chọn 3 model để train ứng với mỗi cách feature engineering khác nhau, sau đó đối chiếu và so sánh, đó là LinearSVC, LogisticRegression và RandomForestClasifier.

+ Bỏ qua các bước import thư viện và chọn model, em sẽ cho thấy kết quả mà các mô hình đạt được.

❖ LinearSVC với TfidfVectorizer(word level) và CountVectorizer(theo thứ tự lần lượt):

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
```

- LogisticRegression với TfidfVectorizer(word level) và CountVectorizer(theo thứ tự lần lượt):

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
```

- ❖ RandomForestClasifier với TfidfVectorizer(word level) và CountVectorizer(theo thứ tự lần lượt):

RandomForestClasifier

```
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
```

+ Ta không quên lưu lại các mô hình để sử dụng sau này

Tiến hành lưu model LinearSVC

```
import joblib

path = '/content/drive/My Drive/FinalModel/LSVC.sav'  #khởi tạo đường dẫn để lưu model
joblib.dump(LSVC, path)
```

```
['/content/drive/My Drive/FinalModel/LSVC.sav']
```

Tiến hành lưu model LogisticRegression

```
import joblib

path = '/content/drive/My Drive/FinalModel/LR.sav'  #khởi tạo đường dẫn để lưu model
joblib.dump(LR, path)
```

```
['/content/drive/My Drive/FinalModel/LR.sav']
```

Tiến hành lưu model RandomForestClasifier

```
import joblib

path = '/content/drive/My Drive/FinalModel/RFC.sav'  #khởi tạo đường dẫn để lưu model
joblib.dump(RFC, path)
```

```
['/content/drive/My Drive/FinalModel/RFC.sav']
```

IV. Fine tuning

1) Sử dụng TfidfVectorizer (ngram-word level) với TruncatedSVD:
   + Với cách này, ta nối các từ trong 1 tiêu đề lại với nhau theo ngram_range
   + Sử dụng TruncatedSVD để giảm chiều dữ liệu nhưng vẫn dữ được đặc trưng của dữ liệu

TfidfVectorize

```
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)
```

2) Sử dụng TfidfVectorizer (ngram-char level) với TruncatedSVD;
   + Với cách này, ta nối các kí tự trong 1 tiêu đề lại với nhau theo ngram_range
   + Sử dụng TruncatedSVD để giảm chiều dữ liệu nhưng vẫn dữ được đặc trưng của dữ liệu

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)
```

3) Tiến hành train với model LinearSVC:
   Ở phần này, em chỉ tiến hành huấn luyện với phương pháp feature engineering ngram-word level
   - ❖ Trong trường hợp 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
```

❖ Trong trường hợp đã giảm chiều dữ liệu:

```
LSVC_finetuning = LinearSVC()
LSVC_finetuning.fit(X_train_svd_nwordlevel, Y_train_encode)
prediction=LSVC_finetuning.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
```

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_nwordlevel, Y_train_encode, test_size=0.15, random_state=42)
```

+ Khởi tạo mô hình Deep Neural Network đơn giản:

- DNN là kiểu kiến trúc mạng neural feed-forward network, tức là tín hiệu chỉ đi từ đầu vào đi đến đầu ra
- Mô hình có 7 lớp các layer trong đó 1 layer input, 1 layer output(Dense layer) và 5 hidden layer(Dense layer)
- Trong các hidden layer, active function đều là hàm relu, output layer có active function là hàm softmax
- Hoàn thiện mô hình với các parameter là: optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy']
- Trong đó, công thức hàm ReLU là: $f(x) = \max(0, x)$, tức là hàm này chọn ra các giá trị lớn hơn 0.

- Dense layer là các fully-connected-layer, tức là mỗi neural của layer phía trước sẽ kết nối với tất cả các neural của layer liền sau. Dense layer cho phép người triển khai lựa chọn số neural của layer, hàm kích hoạt, …

```
#%tensorflow_version 1.x
import tensorflow
from tensorflow import sparse
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)  #Hiden 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'])
```

+ 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:

Tổng hợp kết quả:

	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 sự 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 hù hợp.

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

VI. Dự đoán:

```
from sklearn.model_selection import train_test_split
X_train_app, X_test_app, Y_train_app, Y_test_app = train_test_split(X, Y, test_size=0.2, random_state=42)
```

```
print(encoder.transform(['chinhtri', 'congnghe', 'giaoduc', 'kinhdoanh', 'phapluat', 'thethao']))
```

```
import joblib

lines = input()   #input tiêu đề muốn dự đoán

#các bước xử lý tiêu đề đưa vào
lines = gensim.utils.simple_preprocess(lines)
lines = ' '.join(lines)
lines = ViTokenizer.tokenize(lines)
lines = ''.join(lines)
count_vect = TfidfVectorizer(analyzer='word', max_features=100000)
lines = [lines]
count_vect.fit(X_train_app)
lines = count_vect.transform(lines)
#kết thúc xử lý input

model = joblib.load('/content/drive/My Drive/FinalModel/LSVC.sav')
```

Dòng cuối cùng trong phần này là tiến hành load mô hình mà ta đã lưu lại trước đó

```
prediction = model.predict(lines)
if (prediction==0):
  print('Thuộc chủ đề chính trị')
if (prediction==1):
  print('Thuộc chủ đề công nghệ')
if (prediction==2):
  print('Thuộc chủ đề giáo dục')
if (prediction==3):
  print('Thuộc chủ đề kinh doanh')
if (prediction==4):
  print('Thuộc chủ đề pháp luật')
if (prediction==5):
  print('Thuộc chủ đề thể thao')
```

```
bộ trưởng bộ y tế đề nghị tự bảo vệ bản thân
Thuộc chủ đề chính trị
```

VII. Xây dựng mô hình dự đoán với Flask:

- Sử dụng flask để xây dựng web demo dự đoán của mô hình ta vừa huấn luyện.

- Chi tiết code được up trên github với đường dẫn: https://github.com/hoangnhan12-arc/CS114.K21.KHTN/tree/master/FINAL_MACHINELEARNING
- Trong đó, file full.html trong thư mục template chính là giao diện của web.
- File app.py chứa class dự đoán title, trong đó các bước xử lí tiêu đề input giống như trên (sử dụng TfidfVectorizer word level)
- Bên trong mục `@app.route('/check', methods=['GET', 'POST'])` chính là phần chính ta sử dụng
- Request.form[‘title_result’] chính là yêu cầu input mà ta nhập vào, ứng với biến title_result trong file full.html
- Và class_title chính là kết quả trả về, nằm trong class “result” và ứng với biến class_result, sẽ được viết vào sau dòng “Title topic: “
- Link video demo youtube: https://www.youtube.com/watch?v=tMEb_1Ici8Q

Reference: https://github.com/NguyenVanHieuBlog/programing-language-identify

https://xltiengviet.fandom.com/wiki/Danh_s%C3%A1ch_stop_word
https://github.com/NguyenVanHieuBlog/vietnamesestopwords/blob/master/stopwords.txt