TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC
School of Applied Mathematics and Informatics

Báo cáo cuối kỳ

HỆ HỖ TRỢ QUYẾT ĐỊNH

Chủ đề:
SMS Spam - Xử lý tin nhắn Spam

Giảng viên hướng dẫn: TS.Lê Hải Hà

Sinh viên thực hiện:	Phạm Thu Trang
MSSV:	20195931
Mã lớp:	133591
Mã học phần:	MI4216
Học kỳ:	20212

Ngày 11 tháng 8 năm 2022

Mục lục

Danh sách hình vẽ

Mở đầu

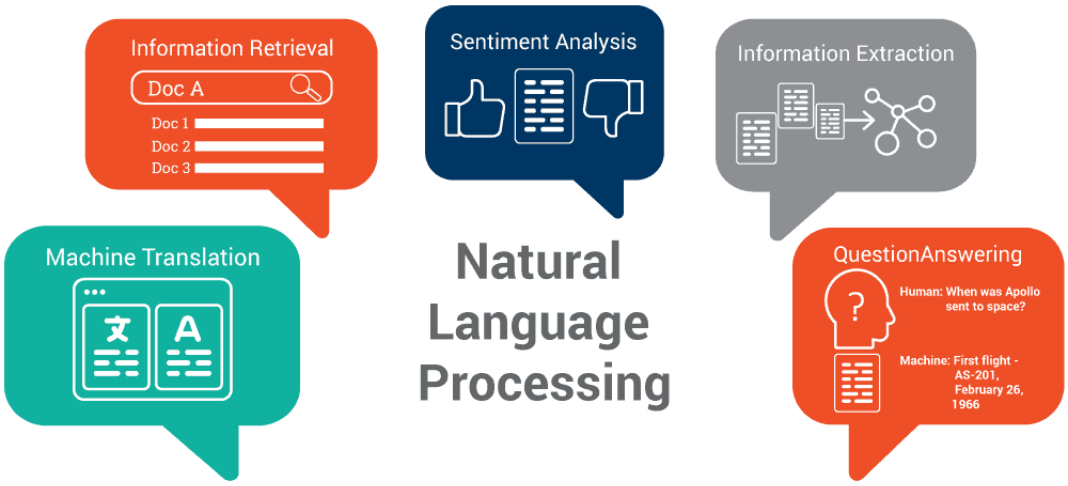
Những năm gần đây, khi mà khả năng tính toán của các máy tính được nâng lên một tầm cao mới và lượng dữ liệu khổng lồ được thu thập bởi các hãng công nghệ lớn, Machine Learning đã tiến thêm một bước dài và một lĩnh vực mới được ra đời gọi là Deep Learning (Học Sâu - thực sự tôi không muốn dịch từ này ra tiếng Việt). Deep Learning đã giúp máy tính thực thi những việc tưởng chừng như không thể vào 10 năm trước: phân loại cả ngàn vật thể khác nhau trong các bức ảnh, tự tạo chú thích cho ảnh, bắt chước giọng nói và chữ viết của con người, giao tiếp với con người, hay thậm chí cả sáng tác văn hay âm nhạc.

Trong bài báo cáo này, em sẽ trình bày cách làm việc với dữ liệu trong Machine Learning cụ thể là trong việc phân loại thư rác thông qua việc xử lý dữ liệu và xây dựng các mô hình lọc thư rác. Em xin cảm ơn sự giúp đỡ của thầy Lê Hải Hà, trong thời gian qua, thầy đã truyền đạt những kiến thức vô cùng quý báu về môn học này cũng như các thuật toán hữu ích. Tuy nhiên,do thời gian và trình độ của em còn hạn chế nên bài báo cáo sẽ khó tránh khỏi những sai sót. Em mong thầy bỏ qua và góp thêm để em hoàn thiện đầy đủ hơn. Em xin cảm ơn thầy và các bạn đã đọc bài báo cáo này!

1 Tổng quan

Xử lý ngôn ngữ tự nhiên (Natural Language Processing hay NLP) là một lĩnh vực con của ngôn ngữ học , khoa học máy tính và trí tuệ nhân tạo liên quan đến các tương tác giữa máy tính và ngôn ngữ con người, đặc biệt là cách lập trình máy tính để xử lý và phân tích lượng lớn dữ liệu ngôn ngữ tự nhiên . Mục tiêu là một máy tính có khả năng "hiểu" nội dung của các tài liệu, bao gồm các sắc thái ngữ cảnh của ngôn ngữ bên trong chúng. Sau đó, công nghệ này có thể trích xuất chính xác thông tin và hiểu biết sâu sắc có trong các tài liệu cũng như tự phân loại và sắp xếp các tài liệu.

Những thách thức trong xử lý ngôn ngữ tự nhiên thường liên quan đến nhận dạng giọng nói , hiểu ngôn ngữ tự nhiên và tạo ngôn ngữ tự nhiên .



Hình 1: Natural Language Processing

Xử lý ngôn ngữ tự nhiên bắt nguồn từ những năm 1950. Vào năm 1950, Alan Turing đã xuất bản một bài báo có tiêu đề "Computing Machinery and Intelligence", đề xuất cái mà ngày nay được gọi là bài kiểm tra Turing như một tiêu chí của trí thông minh, một nhiệm vụ liên quan đến việc giải thích tự động và tạo ra ngôn ngữ tự nhiên, nhưng vào thời điểm đó, nó không được coi là một vấn đề tách biệt với trí tuệ nhân tạo.

1.1 Bộ dữ liệu

Dưới đây ta sẽ tìm hiểu quá trình trong việc xử lý ngôn ngữ tự nhiên thông qua việc thực hành trực tiếp trên một bộ dữ liệu cụ thể như sau:

- Bộ dữ liệu có tên "SMS Spam Collection Data Set" là một tập hợp các tin nhắn có nhãn SMS công khai đã được thu thập để nghiên cứu về spam trên điện thoại di động.
- Nguồn dữ liệu: https://archive.ics.uci.edu/ml/datasets/sms+spam+collection
- Bộ dữ liệu có kích thước 492KB, gồm 2 cột thuộc tính với 5572 bản ghi.

- Cột v1 chứa nhãn: ham hoặc spam
- Cột v2 chứa văn bản thô

1.2 Mục tiêu

- Sử dụng ngôn ngữ python thực hiện load dữ liệu lên Colab và trực quan hóa dữ liệu.
- Với dữ liệu là các văn bản thô thì ta cần thực hiện xử lý dữ liệu ví dụ như Cleaning text, Tokenization, Removing Stopwords, Lemmatization, ...
- Xây dựng một số mô hình phân lớp như NaiveBayes, RandomForest, KNeighbours, SVC và đưa ra đánh giá các mô hình đó.

2 Thực hành

2.1 Khai báo thư viện

```
import warnings
import seaborn as sns
import numpy as np
import pandas as pd
import re

import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.metrics import precision_score, recall_score, plot_confusion_matrix,
    classification_report, accuracy_score, f1_score
from sklearn import metrics

import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

2.2 Load dữ liệu

```
data = pd.read_csv("spam.csv",encoding='latin-1')
data.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

Hình 2: Load dữ liệu

Đây là kết quả sau khi chương trình đọc được file dữ liệu đầu vào dưới dạng csv và đưa ra 5 dòng đầu tiên của bộ dữ liệu. Để biết thông tin chi tiết các cột ta sử dụng câu lệnh sau:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   v1          5572 non-null   object
 1   v2          5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

Hình 3: Thông tin của các thuộc tính

Khi đó ta thực hiện xóa các cột không cần thiết và để lại hai cột thuộc tính chính với tên gọi lần lượt là "Target" và "Text"

```
# Dropping
to_drop = ["Unnamed: 2","Unnamed: 3","Unnamed: 4"]
data = data.drop(data[to_drop], axis=1)
# Renaming
data.rename(columns = {"v1":"Target", "v2":"Text"}, inplace = True)
data.head()
```

	Target	Text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Hình 4: Kết quả sau khi xóa cột không cần thiết và đổi tên

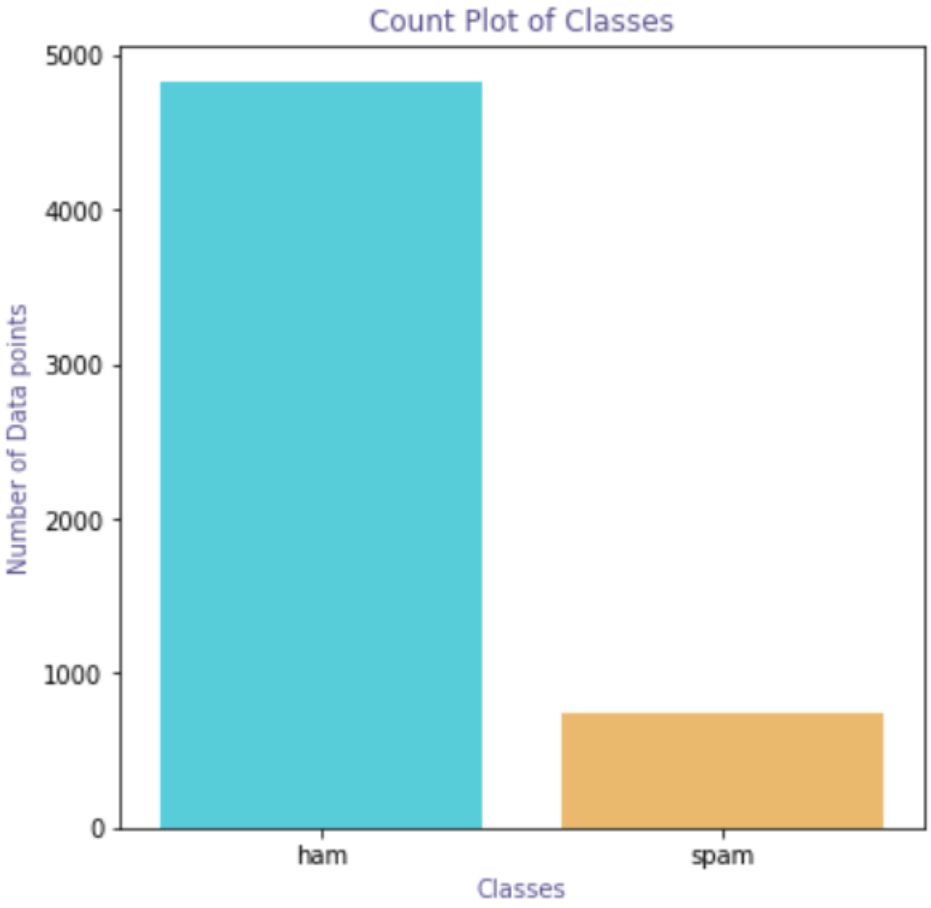
Kết quả thu được:

- Bộ dữ liệu bao gồm 5.572 tin nhắn bằng tiếng Anh.
- Dữ liệu được chỉ định là ham hoặc spam.
- DataFrame có hai cột:
  1. Target: biểu thị lớp tin nhắn là ham hoặc spam
  2. Text: chuỗi văn bản

2.3 Trực quan hóa dữ liệu

Đầu tiên, ta vẽ biểu đồ thể hiện số lượng tin nhắn thuộc lớp ham và spam.

```
cols= ["#41DFEF", "#FFBD59"]
plt.figure(figsize=(6,6))
fg = sns.countplot(x= data["Target"], palette= cols)
fg.set_title("Count Plot of Classes", color="#58508d")
fg.set_xlabel("Classes", color="#58508d")
fg.set_ylabel("Number of Data points", color="#58508d")
```



Hình 5: Biểu đồ thể hiện số lượng ham hoặc spam của cột Target

Nhận xét: Từ biểu đồ ở trên, sự mất cân bằng dữ liệu là khá rõ ràng.

Với mục đích khám phá dữ liệu, chúng ta sẽ tạo các thuộc tính mới

- No_of_Characters: Số ký tự trong tin nhắn văn bản
- No_of_Words: Số từ trong tin nhắn văn bản
- No_of_sentence: Số câu trong tin nhắn văn bản

```
import nltk
nltk.download('punkt')
```

```
#Adding a column of numbers of charachters,words and sentences in each msg
data["No_of_Characters"] = data["Text"].apply(len)
data["No_of_Words"] = data.apply(lambda row: nltk.word_tokenize(row["Text"]), axis=1).
    apply(len)
data["No_of_sentence"]=data.apply(lambda row: nltk.sent_tokenize(row["Text"]), axis=1)
    .apply(len)

data.describe().T
```

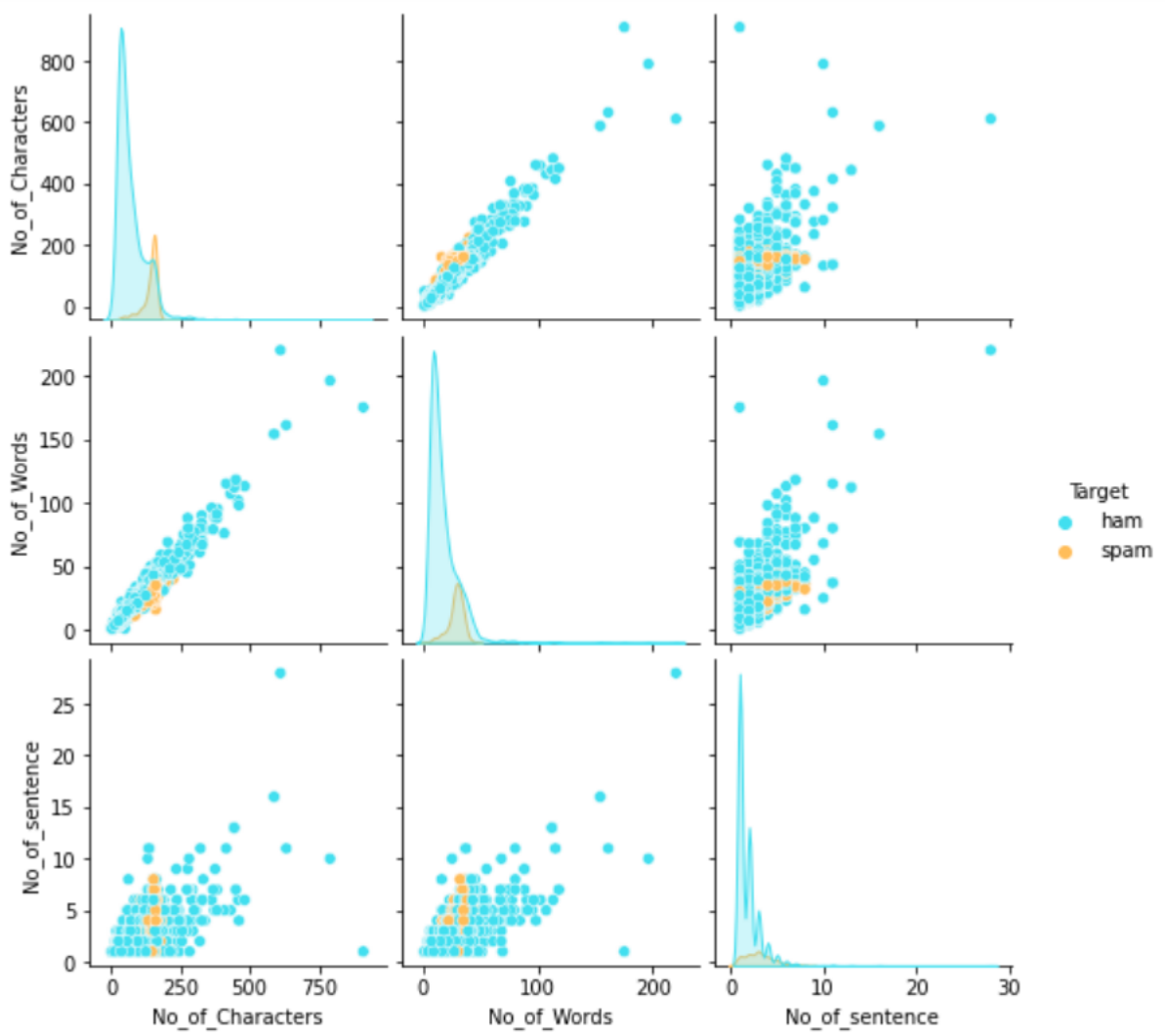
	count	mean	std	min	25%	50%	75%	max
No_of_Characters	5572.0	80.118808	59.690841	2.0	36.0	61.0	121.0	910.0
No_of_Words	5572.0	18.695621	13.742587	1.0	9.0	15.0	27.0	220.0
No_of_sentence	5572.0	1.970747	1.417778	1.0	1.0	1.0	2.0	28.0

Hình 6: Bảng thống kê các thuộc tính mới

Từ kết quả trên, để có cái nhìn tổng quan nhất ta sẽ trực quan hóa nó bằng biểu đồ thông qua các câu lệnh như sau

```
plt.figure(figsize=(12,8))
fg = sns.pairplot(data=data, hue="Target",palette=cols)
plt.show(fg)
```

Ta thu được kết quả



Hình 7: Biểu đồ thống kê các thuộc tính mới

Nhận xét: Từ biểu đồ, chúng ta có thể thấy một vài điểm ngoại lệ trong tất cả các lớp ham. Vì vậy chúng ta có thể đặt một giới hạn cho một trong những điểm này. Về cơ bản chúng chỉ ra độ dài của SMS.

2.4 Xử lý dữ liệu

2.4.1 Cleaning text - Làm sạch văn bản

Quá trình làm sạch dữ liệu NLP là rất quan trọng. Máy tính không hiểu văn bản. Đối với máy tính, nó chỉ là một cụm ký hiệu. Để tiếp tục xử lý dữ liệu, ta cần làm cho dữ liệu sạch hơn.

- Trong bước đầu tiên, ta chỉ trích xuất các ký tự chữ cái và loại bỏ dấu câu, số.
- Trong bước tiếp theo, ta sẽ chuyển đổi tất cả các ký tự thành chữ thường.

Văn bản này sau đó sẽ được sử dụng trong phân tích kỹ thuật tiếp theo.

```
#Lets have a look at a sample of texts before cleaning
print("\033[1m\u001b[45;1m The First 5 Texts:\033[0m",*data["Text"][:5],sep="\n")
```

The First 5 Texts:
Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
Ok lar... Joking wif u oni...
Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's
U dun say so early hor... U c already then say...
Nah I don't think he goes to usf, he lives around here though

```
# Defining a function to clean up the text
def Clean(Text):
    sms = re.sub('[^a-zA-Z]', ' ', Text) #Replacing all non-alphabetic characters with
     a space
    sms = sms.lower() #converting to lowecase
    sms = sms.split()
    sms = ' '.join(sms)
    return sms

data["Clean_Text"] = data["Text"].apply(Clean)
#Lets have a look at a sample of texts after cleaning
print("\033[1m\u001b[45;1m The First 5 Texts after cleaning:\033[0m",*data["Clean_Text
    "][:5], sep = "\n")
```

The First 5 Texts after cleaning:
go until jurong point crazy available only in bugis n great world la e buffet cine there got amore wat
ok lar joking wif u oni
free entry in a wkly comp to win fa cup final tkts st may text fa to to receive entry question std txt rate t c s apply over s
u dun say so early hor u c already then say
nah i don t think he goes to usf he lives around here though

2.4.2 Tokenization

Trong bước này, văn bản được chia thành các đơn vị nhỏ hơn. Chúng ta có thể sử dụng mã thông báo câu hoặc mã thông báo từ dựa trên câu lệnh

```
data["Tokenize_Text"]=data.apply(lambda row: nltk.word_tokenize(row["Clean_Text"]),
    axis=1)

print("\033[1m\u001b[45;1m The First 5 Texts after Tokenizing:\033[0m",*data["
    Tokenize_Text"][:5], sep = "\n")
```

The First 5 Texts after Tokenizing:
['go', 'until', 'jurong', 'point', 'crazy', 'available', 'only', 'in', 'bugis', 'n', 'great', 'world', 'la', 'e', 'buffet', 'cine', 'there', 'got', 'amore', 'wat']
['ok', 'lar', 'joking', 'wif', 'u', 'oni']
['free', 'entry', 'in', 'a', 'wkly', 'comp', 'to', 'win', 'fa', 'cup', 'final', 'tkts', 'st', 'may', 'text', 'fa', 'to', 'to', 'receive', 'entry', 'question', 'std', 'txt', 'rate', 't', 'c', 's', 'apply', 'over', 's']
['u', 'dun', 'say', 'so', 'early', 'hor', 'u', 'c', 'already', 'then', 'say']
['nah', 'i', 'don', 't', 'think', 'he', 'goes', 'to', 'usf', 'he', 'lives', 'around', 'here', 'though']

2.4.3 Removing Stopwords

Từ dừng là những từ xuất hiện thường xuyên (chẳng hạn như ít, là, an, v.v.). Những từ này giữ ý nghĩa trong cấu trúc câu, nhưng không đóng góp nhiều vào quá trình xử lý ngôn ngữ trong NLP. Với mục đích loại bỏ phần dư thừa trong quá trình xử lý, ta sẽ loại bỏ những phần đó. Thư viện NLTK có một tập hợp các từ dừng mặc định mà chúng ta sẽ xóa.

```
import nltk
nltk.download('stopwords')
```

```
#Removing the stopwords function
def remove_stopwords(text):
    stop_words = set(stopwords.words("english"))
    filtered_text = [word for word in text if word not in stop_words]
    return filtered_text

data["Nostopword_Text"] = data["Tokenize_Text"].apply(remove_stopwords)

print("\033[1m\u001b[45;1m The First 5 Texts after removing the stopwords:\033[0m",*
    data["Nostopword_Text"][:5], sep = "\n")
```

The First 5 Texts after removing the stopwords:
['go', 'jurong', 'point', 'crazy', 'available', 'bugis', 'n', 'great', 'world', 'la', 'e', 'buffet', 'cine', 'got', 'amore', 'wat']
['ok', 'lar', 'joking', 'wif', 'u', 'oni']
['free', 'entry', 'wkly', 'comp', 'win', 'fa', 'cup', 'final', 'tkts', 'st', 'may', 'text', 'fa', 'receive', 'entry', 'question', 'std', 'txt', 'rate', 'c', 'apply']
['u', 'dun', 'say', 'early', 'hor', 'u', 'c', 'already', 'say']
['nah', 'think', 'goes', 'usf', 'lives', 'around', 'though']

2.4.4 Lemmatization

Lemmatization cũng chuyển một từ về dạng gốc của nó. Tuy nhiên, sự khác biệt là sự bổ sung đảm bảo rằng từ gốc thuộc về ngôn ngữ mà người ta đang xử lý, trong trường hợp của chúng ta, đó là tiếng Anh. Nếu chúng ta sử dụng lemmatization, đầu ra sẽ là tiếng Anh.

Original Word	After Stemming	After Lemmatization
goose	goos	goose
geese	gees	goose

Hình 8: Sự khác biệt giữa Stemming và Lemmatization

```
import nltk
nltk.download('wordnet')
```

```
import nltk
nltk.download('omw-1.4')
```

```
lemmatizer = WordNetLemmatizer()
# lemmatize string
def lemmatize_word(text):
    lemmas = [lemmatizer.lemmatize(word, pos ='v') for word in text]
    return lemmas

data["Lemmatized_Text"] = data["Nostopword_Text"].apply(lemmatize_word)
print("\033[1m\u001b[45;1m The First 5 Texts after lemitization:\033[0m",*data["
    Lemmatized_Text"][:5], sep = "\n")
```

The First 5 Texts after lemitization:
['go', 'jurong', 'point', 'crazy', 'available', 'bugis', 'n', 'great', 'world', 'la', 'e', 'buffet', 'cine', 'get', 'amore', 'wat']
['ok', 'lar', 'joke', 'wif', 'u', 'oni']
['free', 'entry', 'wkly', 'comp', 'win', 'fa', 'cup', 'final', 'tkts', 'st', 'may', 'text', 'fa', 'receive', 'entry', 'question', 'std', 'txt', 'rate', 'c', 'apply']
['u', 'dun', 'say', 'early', 'hor', 'u', 'c', 'already', 'say']
['nah', 'think', 'go', 'usf', 'live', 'around', 'though']

2.5 Vectorization

TF-IDF trong NLP là viết tắt của Term Frequency - Inverse document frequency. Trong NLP, dữ liệu được làm sạch cần được chuyển đổi thành một định dạng số trong đó mỗi từ được biểu diễn bằng một ma trận. Điều này còn được gọi là nhúng từ hoặc vector hóa từ.

$$\text{Tần suất thuật ngữ (TF)} = \frac{\text{Tần suất của một thuật ngữ trong tài liệu}}{\text{Tổng số thuật ngữ trong tài liệu}}$$

$$\text{Tần suất tài liệu nghịch đảo (IDF)} = \log\left(\frac{\text{Tổng số tài liệu}}{\text{Số tài liệu có thuật ngữ t}}\right)$$

Ta sẽ sử dụng TfidfVectorizer() để vectơ hóa dữ liệu được xử lý trước.

Các bước trong Vectorizing:

- Tạo một kho văn bản dạng lemmatized
- Chuyển đổi kho dữ liệu ở dạng vectơ
- Mã hóa nhãn cho các lớp trong Target

```
#Creating a corpus of text feature to encode further into vectorized form
corpus= []
for i in data["Lemmatized_Text"]:
    msg = ' '.join([row for row in i])
    corpus.append(msg)

corpus[:5]
print("\033[1m\u001b[45;1m The First 5 lines in corpus :\033[0m",*corpus[:5], sep = "\
    n")
```

The First 5 lines in corpus :
go jurong point crazy available bugis n great world la e buffet cine get amore wat
ok lar joke wif u oni
free entry wkly comp win fa cup final tkts st may text fa receive entry question std txt rate c apply
u dun say early hor u c already say
nah think go usf live around though

```
#Changing text data in to numbers.
tfidf = TfidfVectorizer()
X = tfidf.fit_transform(corpus).toarray()
#Let's have a look at our feature
X.dtype
```

dtype('float64')

```
#Label encode the Target and use it as y
label_encoder = LabelEncoder()
data["Target"] = label_encoder.fit_transform(data["Target"])
```

2.6 Các mô hình

2.6.1 Xây dựng mô hình

Các bước liên quan đến Xây dựng Mô hình

- Thiết lập các tính năng và nhắm mục tiêu dưới dạng X và y
- Tách bộ kiểm tra và đào tạo
- Xây dựng một hệ thống mô hình cho bốn bộ phân loại khác nhau.
  1. Naïve Bayes
  2. RandomForestClassifier
  3. KNeighborsClassifier
  4. Support Vector Machines
- Phù hợp với tất cả các mô hình trên dữ liệu đào tạo
- Nhận xác nhận chéo trên tập hợp đào tạo cho tất cả các mô hình để đảm bảo độ chính xác

```
#Setting values for labels and feature as y and X(we already did X in vectorizing...)
y = data["Target"]
# Splitting the testing and training sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
    =42)
```

```
#Testing on the following classifiers
classifiers = [MultinomialNB(),
               RandomForestClassifier(),
               KNeighborsClassifier(),
               SVC()]
for cls in classifiers:
    cls.fit(X_train, y_train)

# Dictionary of pipelines and model types for ease of reference
pipe_dict = {0: "NaiveBayes", 1: "RandomForest", 2: "KNeighbours",3: "SVC"}
```

```
# Cossvalidation
for i, model in enumerate(classifiers):
    cv_score = cross_val_score(model, X_train,y_train,scoring="accuracy", cv=10)
    print("%s: %f " % (pipe_dict[i], cv_score.mean()))
```

NaiveBayes: 0.967552
RandomForest: 0.975214
KNeighbours: 0.911450
SVC: 0.974086

2.6.2 Đánh giá các mô hình

Sau khi xây dựng được các mô hình ta sẽ tính toán các chỉ số đánh giá của mỗi mô hình đó bao gồm precision, recall, f1 score, accuracy on trainset, accuracy on testset

```
# Model Evaluation
# creating lists of varios scores
precision =[]
recall =[]
f1_score = []
trainset_accuracy = []
testset_accuracy = []

for i in classifiers:
    pred_train = i.predict(X_train)
    pred_test = i.predict(X_test)
    prec = metrics.precision_score(y_test, pred_test)
    recal = metrics.recall_score(y_test, pred_test)
    f1_s = metrics.f1_score(y_test, pred_test)
    train_accuracy = model.score(X_train,y_train)
    test_accuracy = model.score(X_test,y_test)

    #Appending scores
    precision.append(prec)
    recall.append(recal)
    f1_score.append(f1_s)
    trainset_accuracy.append(train_accuracy)
    testset_accuracy.append(test_accuracy)
```

```
# initialise data of lists.
data = {'Precision':precision,
'Recall':recall,
'F1score':f1_score,
'Accuracy on Testset':testset_accuracy,
'Accuracy on Trainset':trainset_accuracy}
# Creates pandas DataFrame.
Results = pd.DataFrame(data, index =["NaiveBayes", "RandomForest", "KNeighbours","SVC"
    ])
```

Dưới đây là bảng kết quả các chỉ số mà ta tính toán được của bốn thuật toán đã sử dụng

```
cmap2 = ListedColormap(["#41DFEF", "#FFBD59"])
Results.style.background_gradient(cmap=cmap2)
```

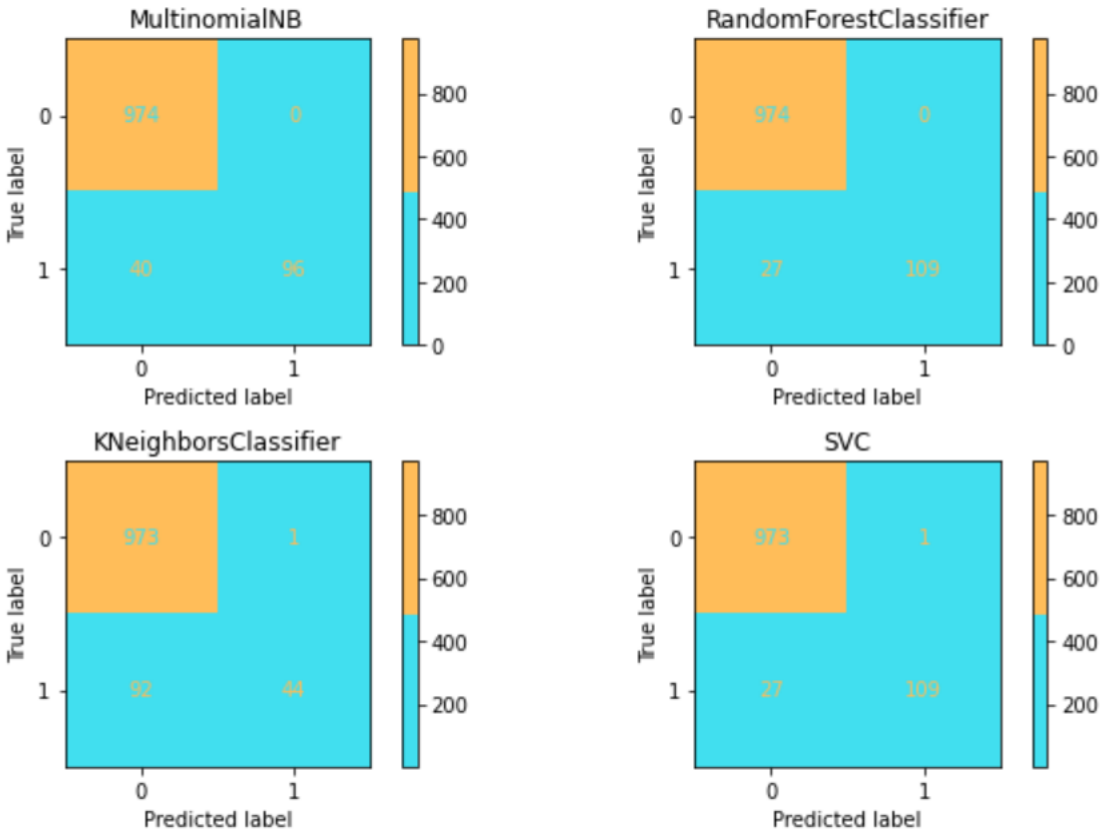
	Precision	Recall	F1score	Accuracy on Testset	Accuracy on Trainset
NaiveBayes	1.000000	0.705882	0.827586	0.974775	0.997521
RandomForest	1.000000	0.801471	0.889796	0.974775	0.997521
KNeighbours	0.977778	0.323529	0.486188	0.974775	0.997521
SVC	0.990909	0.801471	0.886179	0.974775	0.997521

Hình 9: Bảng thống kê các chỉ số của các thuật toán

Từ đó ta cũng dễ dàng tính được ma trận nhầm lẫn (Confusion matrix) của các thuật toán trên được biểu diễn bằng hình dưới đây

```
cmap = ListedColormap(["#41DFEF", "#FFBD59"])
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(9,6))

for cls, ax in zip(classifiers, axes.flatten()):
    plot_confusion_matrix(cls,
                          X_test,
                          y_test,
                          ax=ax,
                          cmap= cmap,
                          )
    ax.title.set_text(type(cls).__name__)
plt.tight_layout()
plt.show()
```



Hình 10: Ma trận nhầm lẫn của các thuật toán

Nhận xét: Qua việc xây dựng các mô hình phân lớp ta có thể thấy các mô hình này cho ta độ chính xác rất cao và khá tương đồng nhau.

Kết luận

Dựa trên bộ dữ liệu là tập hợp các tin nhắn có nhãn SMS công khai thì báo cáo đã trình bày được một số bước cơ bản khi làm việc với dữ liệu nhằm phân loại thư rác cũng như xây dựng các mô hình lọc thư rác. Với các kết quả thu được trong báo cáo này thì độ chính xác của các mô hình không khác biệt quá nhiều. Tuy nhiên, trong thực tế việc lọc thư rác sử dụng thuật toán Naive Bayes đơn giản và dễ triển khai hơn, phù hợp cho cả bộ lọc cá nhân và máy chủ, do vậy nó được sử dụng rộng rãi ngày nay mặc dù độ chính xác không quá cao và thiếu khả năng xử lý hình ảnh.