Chapter 6 - Ex3: Spam vs Ham

- Cho dữ liệu spam.csv chứa thông tin là nội dung các email. Bộ dữ liệu này có thể được sử dụng để dự đoán một email gửi đến là ham hay spam.(link: https://www.kaggle.com/uciml/sms-spam-collection-dataset (https://www.kaggle.com/uciml/sms-spam-collection-dataset))

Yêu cầu:

- Đọc dữ liệu, tìm hiểu sơ bộ về dữ liệu
- Chọn phương pháp để chuẩn hóa dữ liệu và thực hiện việc chuẩn hóa.

Với CountVectorizer

In [1]:
```
# Load libraries
import numpy as np
import pandas as pd
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
```

In [2]:
```
# import some data to play with
data = pd.read_csv("spam.csv", encoding='latin-1')
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
```

In [3]:
```
data.head()
```

Out[3]:

	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

In [4]:
```
source = data['v2']
type(source)
```

Out[4]:
```
pandas.core.series.Series
```

In [5]:
```
source.head()
```

Out[5]:
```
0    Go until jurong point, crazy.. Available only ...
1                        Ok lar... Joking wif u oni...
2    Free entry in 2 a wkly comp to win FA Cup fina...
3    U dun say so early hor... U c already then say...
4    Nah I don't think he goes to usf, he lives aro...
Name: v2, dtype: object
```

In [6]:
```
target = data['v1']
type(target)
```

Out[6]:
```
pandas.core.series.Series
```

In [7]:
```
target.head()
```

Out[7]:
```
0     ham
1     ham
2    spam
3     ham
4     ham
Name: v1, dtype: object
```

In [8]:
```
# 0: ham, 1:spam
target = pd.get_dummies(target, drop_first=True)
target.head()
```

Out[8]:

	spam
0	0
1	0
2	1
3	0
4	0

In [9]:
```
# Import CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# Instantiate CountVectorizer
cv = CountVectorizer(stop_words='english')
cv
```

Out[9]:
```
CountVectorizer(stop_words='english')
```

In [10]:
```
# Fit the vectorizer
cv.fit(source)
```

Out[10]:
```
CountVectorizer(stop_words='english')
```

In [11]:
```
# Print feature names
# print(cv.get_feature_names())
```

In [12]:
```
#cv.vocabulary_
list(cv.vocabulary_.keys())[0:5]
```

Out[12]:
```
['jurong', 'point', 'crazy', 'available', 'bugis']
```

In [13]:
```
# Apply the vectorizer
cv_transformed = cv.transform(source)
# Print the full array
cv_array = cv_transformed.toarray()
```

In [14]:
```
cv_array.shape
```

Out[14]:
```
(5572, 8404)
```

In [15]:
```
 from scipy import sparse
```

In [16]:
```
a0 = sparse.csr_matrix(cv_array[0])
print(a0)
```

```
  (0, 1051)     1
  (0, 1271)     1
  (0, 1701)     1
  (0, 1703)     1
  (0, 1994)     1
  (0, 2271)     1
  (0, 3494)     1
  (0, 3534)     1
  (0, 4224)     1
  (0, 4349)     1
  (0, 5741)     1
  (0, 8026)     1
  (0, 8227)     1
```

Với Tf-Idf

In [17]:
```
# Import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
# Instantiate TfidfVectorizer
tv = TfidfVectorizer(max_features=500, stop_words='english')
tv
```

Out[17]:
```
TfidfVectorizer(max_features=500, stop_words='english')
```

In [18]:
```
# Fit the vectroizer
tv.fit_transform(source)
```

Out[18]:
```
<5572x500 sparse matrix of type '<class 'numpy.float64'>'
        with 23808 stored elements in Compressed Sparse Row format>
```

In [19]:
```
# print(tv.get_feature_names())
```

In [20]:
```
#tv.vocabulary_
list(tv.vocabulary_.keys())[0:5]
```

Out[20]:
```
['available', 'great', 'world', 'got', 'wat']
```

In [21]:
```
# Transform the data
tv_transformed = tv.transform(source)
tv_array = tv_transformed.toarray()
```

In [22]:
```
print(tv_array)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

In [23]:
```
# Create a DataFrame with these features
tv_df = pd.DataFrame(tv_transformed.toarray(),
                     columns=tv.get_feature_names()).add_prefix('TFIDF_')
tv_df.head()
```

Out[23]:

	TFIDF_000	TFIDF_10	TFIDF_100	TFIDF_1000	TFIDF_10p	TFIDF_12hrs	TFIDF_150	TFIDF_150p
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 500 columns

In [24]:
```
examine_row = tv_df.iloc[0]
print(examine_row. sort_values(ascending=False).head())
```

```
TFIDF_available    0.549238
TFIDF_world        0.496702
TFIDF_wat          0.410286
TFIDF_great        0.405632
TFIDF_got          0.344604
Name: 0, dtype: float64
```

Với TF-IDF và N-grams

In [25]:
```
tv_bi_gram_vec = TfidfVectorizer(ngram_range = (1, 2),
                                 stop_words='english')
# Fit and apply bigram vectorizer
tv_bi_gram = tv_bi_gram_vec.fit_transform(source)
```

In [26]:
```
# Print the bigram features
# print(tv_bi_gram_vec.get_feature_names())
```

In [27]:
```
#tv_bi_gram_vec.vocabulary_
list(tv_bi_gram_vec.vocabulary_.keys())[0:5]
```

Out[27]:
```
['jurong', 'point', 'crazy', 'available', 'bugis']
```

In [28]:
```
# Create a DataFrame with the Counts features
tv_df = pd.DataFrame(tv_bi_gram.toarray(),
       columns=tv_bi_gram_vec.get_feature_names()).add_prefix('Counts_')
tv_sums = tv_df.sum()
```

In [29]:
```
print(tv_sums.head())
```

```
Counts_00           1.540984
Counts_00 easter    0.168684
Counts_00 sub       0.838168
Counts_00 subs      0.346519
Counts_000          4.507586
dtype: float64
```