SpamHamClassification

December 17, 2023

0.1 Spam Ham Classification - Naive Bayes

0.1.1 Import

[45]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import re,string,collections
from nltk.stem import WordNetLemmatizer
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix,classification_report
```

0.1.2 Gaining Intuition

[2]:
```
df = pd.read_csv('spam.csv')
df.head()
```

[2]:
```
  Category                                            Message
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…
```

[3]:
```
df.groupby('Category').describe()
```

[3]:
```
          Message                                                                \
            count unique                                                    top
Category
ham          4825   4516                               Sorry, I'll call later
spam          747    641  Please call our customer service representativ…
```

```
         freq
Category
ham        30
spam        4
```

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Category  5572 non-null   object
 1   Message   5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

[5]:
```
df.columns
```

[5]:
```
Index(['Category', 'Message'], dtype='object')
```

[6]:
```
df.isnull().sum()
```

[6]:
```
Category    0
Message     0
dtype: int64
```

[7]:
```
df['Category'].value_counts()
```

[7]:
```
ham     4825
spam     747
Name: Category, dtype: int64
```

[8]:
```
df.shape
```

[8]:
```
(5572, 2)
```

0.1.3 Data Cleaning

Data Processing

- Removing Punctuations
- Links
- Stop words
- Performing Lemmatization

[9]:
```
df['Message'] = [re.sub(r'[https|www]\S+', "", msg) for msg in df['Message']]
df['Message'].head()
```

[9]:
```
0    Go un jurong  crazy.. Available only in bugis …
1                           Ok lar… Joking  u oni…
2    Free en in 2 a  comp   FA Cup final  21 May 20…
3                        U dun   early  U c already
4               Nah I don't   goes  u  lives around
Name: Message, dtype: object
```

[10]:
```
# Since I found individual words like '..' , '...' I'm removing everything that␣
 ↪starts with '.' & repeats more than once...
df['Message'] = [re.sub(r'\.{1,}', "", msg) for msg in df['Message']]
df['Message'].head()
```

[10]:
```
0    Go un jurong  crazy Available only in bugis n …
1                                Ok lar Joking  u oni
2    Free en in 2 a  comp   FA Cup final  21 May 20…
3                        U dun   early  U c already
4               Nah I don't   goes  u  lives around
Name: Message, dtype: object
```

Processing words

[11]:
```
stop_words = stopwords.words('english')
lemma = WordNetLemmatizer()
def processWords(message: str):
    words = message.lower()
    words = word_tokenize(words)
    # Removing punctuations
    words = [word for word in words if word not in string.punctuation]
    # Single letters and digits need'nt be considered
    words = [word for word in words if len(word)>1]
    # Removing all of the stop words
    words = [word for word in words if word not in stop_words]
    # Lemmatizing the words...
    words = [lemma.lemmatize(word) for word in words]

    return words
```

Separating spam and ham messages for easy understanding and visualization

[12]:
```
# Separating
spamMessages,hamMessages = df[df['Category'] == 'spam']['Message'],␣
 ↪df[df['Category'] == 'ham']['Message']
# Processing
spamMessages,hamMessages = [processWords(message) for message in␣
 ↪spamMessages],[processWords(message) for message in hamMessages ]
```

0.1.4 Visualization

Counting words in each messages

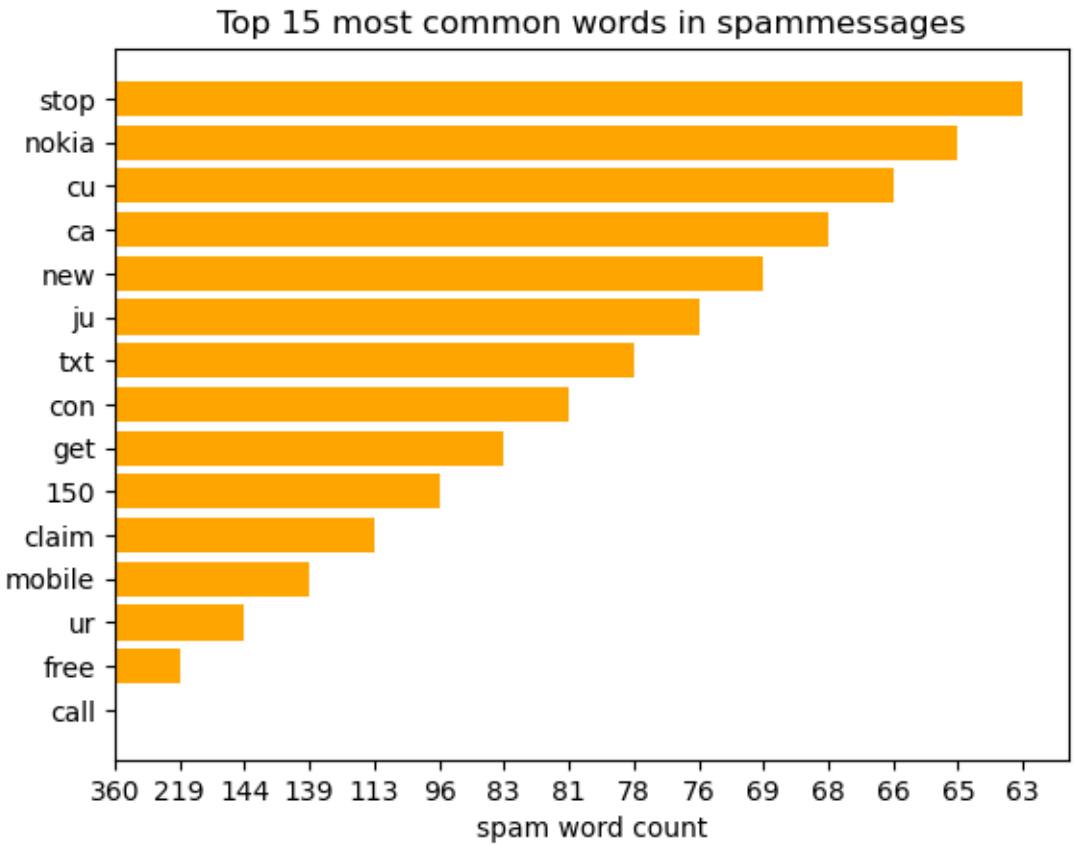
[13]:
```
# Calculating frequence for each words in order to represent it in the bar chart
def count_Words(Message: list):
    counter = collections.OrderedDict()
    for words in Message:
        for word in words:
            counter[word] = counter.get(word,0)+1
    return counter
```

[14]:
```
spam_words_dict = count_Words(spamMessages)
ham_words_dict = count_Words(hamMessages)
```

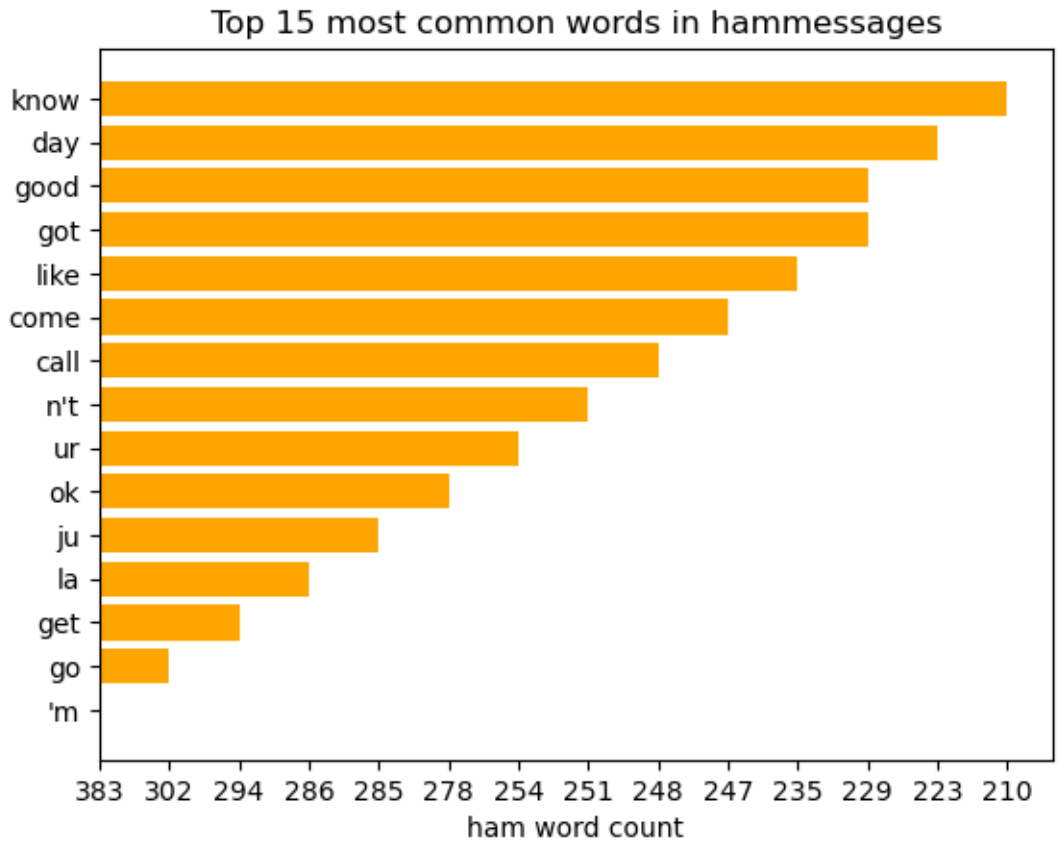
[51]:
```
# Plotting data in bar chart for better understanding

def bar_chart_words(count_dict: dict, top=10, messages_type="",␣
 ↪color="#1f77b4"):
    # Sorting the keys with high values in descending order to vizualize
    words = np.array(sorted(count_dict.items(), key = lambda x: -x[1]))[:top]
    top_words = words[:,0] # [:,0] - Represents column 1
    top_words_count = words[:,1] #[:,1] Represents column 2

    # Graph representation
    plt.title(f'Top {top} most common words in {messages_type}messages')
    plt.xlabel(f'{messages_type} word count')
    plt.barh(top_words,top_words_count, color = color)
```

Visualization with `Bar Chart`

[16]:
```
# spam
bar_chart_words(spam_words_dict, top = 15, messages_type='spam',color =␣
 ↪'orange')
```



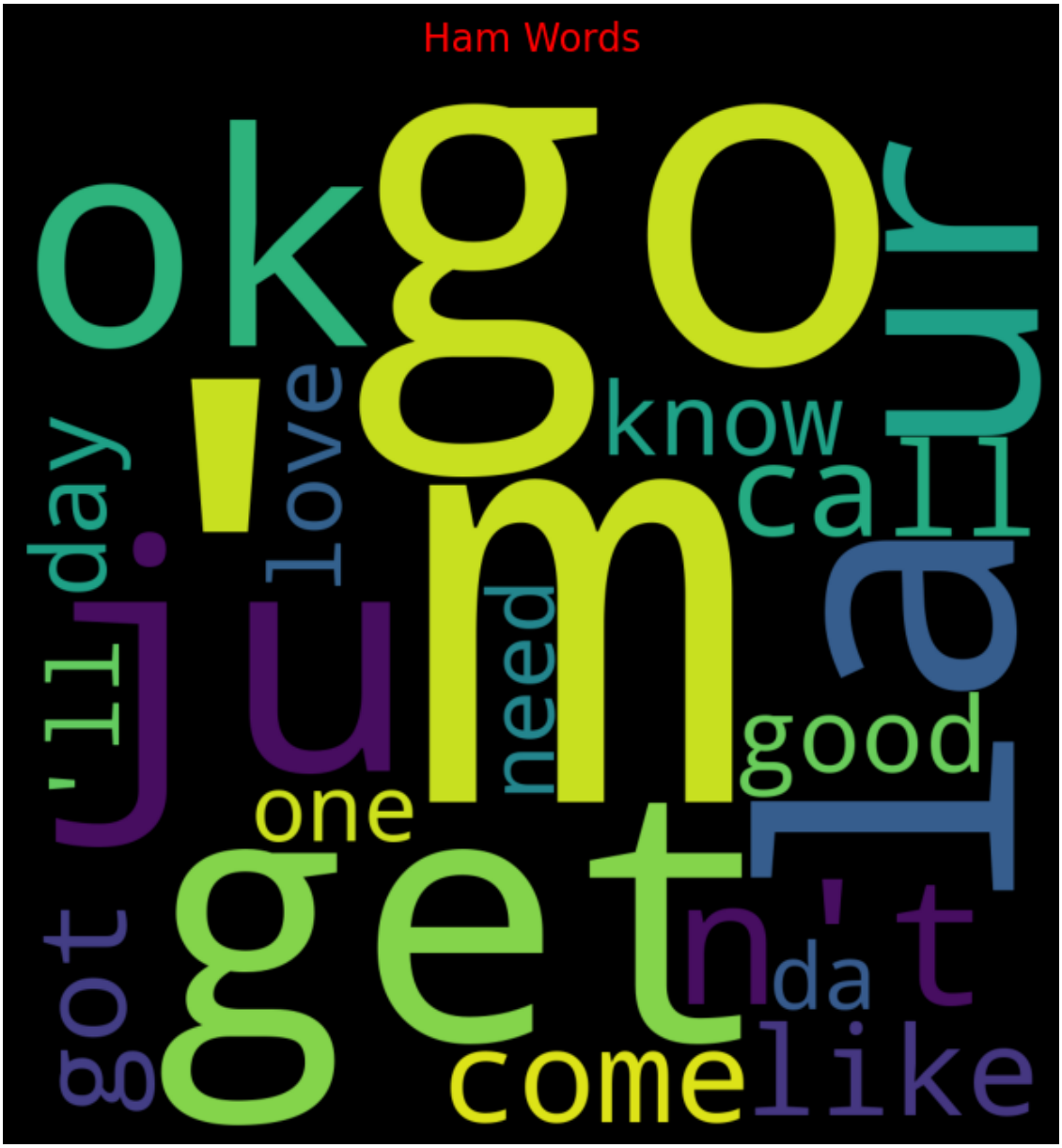
[17]:
```
# ham
bar_chart_words(ham_words_dict, top = 15, messages_type='ham',color = 'orange')
```



Visualizing with `word cloud` gives even more clear view

[18]:
```
# Visualizing with word cloud gives even more clear view
# Spam
wc = WordCloud(width=1024,height = 1024, max_words=20).
 ↪generate_from_frequencies(spam_words_dict)
plt.figure(figsize=(8,6),facecolor='k')
plt.imshow(wc)
plt.axis('off')
plt.tight_layout(pad=0)
plt.text(0.5, 1.05, 'Spam Words', color='Red', fontsize=16, ha='center',␣
 ↪transform=plt.gca().transAxes)
plt.show()
```

[19]:
```
# Ham
wc = WordCloud(width=1024,height = 1024, max_words=20).
  ↪generate_from_frequencies(ham_words_dict)
plt.figure(figsize=(8,6),facecolor='k')
plt.imshow(wc)
plt.axis('off')
plt.tight_layout(pad=0)
plt.text(0.5, 1.05, 'Ham Words', color='Red', fontsize=16, ha='center',␣
  ↪transform=plt.gca().transAxes)
plt.show()
```



[20]:
```
# Combining spam and ham messages after vizualization
allMessages = spamMessages + hamMessages
all_words = count_Words(allMessages)
# Now the messages are ready to be vectorized...
allMessages = [' '.join(msglist) for msglist in allMessages]
```

0.1.5 Training the Model

Splitting the model

[21]:
```
allMessages[:2]
```

[21]:
```
["free en comp fa cup final 21 may 2005 text fa 87121 receive en que ra
08452810075over18 's",
 "freem hey darling back 'd like fun tb ok xxx £150 rcv"]
```

Defining spam column that represents the one hot encoding of the `Category` column

[22]:
```
df['spam'] = df.Category.apply(lambda x: 1 if x== 'spam' else 0)
df.head()
```

[22]:

	Category	Message	spam
0	ham	Go un jurong crazy Available only in bugis n …	0
1	ham	Ok lar Joking u oni	0
2	spam	Free en in 2 a comp FA Cup final 21 May 20…	1
3	ham	U dun early U c already	0
4	ham	Nah I don't goes u lives around	0

Vectorize

[30]:
```
vector = CountVectorizer()
# Since the algorithm can only take numbers as arguments...
# Since X_train is an array of array of strings, we should change that to array␣
 ↪of strings such that it is vectorized
X = vector.fit_transform(allMessages)
X.toarray()[:3]
```

[30]:
```
array([[0, 0, 0, …, 0, 0, 0],
       [0, 0, 0, …, 0, 0, 0],
       [0, 0, 0, …, 0, 0, 0]], dtype=int64)
```

[31]:
```
# splitting train & test
X_train,X_test,Y_train,Y_test = train_test_split(X,df.spam,test_size=0.25)
```

[35]:
```
X_train.shape,Y_train.shape
```

[35]:
```
((4179, 5132), (4179,))
```

Fitting model

[36]:
```
model = MultinomialNB()
model.fit(X_train,Y_train)
```

[36]:
```
MultinomialNB()
```

0.1.6 Prediction and Evaluation

Measures

[40]:
```
Y_pred = model.predict(X_test)
Y_pred
```

[40]:
```
array([0, 0, 0, …, 0, 0, 0], dtype=int64)
```

[49]:
```
print(classification_report(y_true = Y_test, y_pred = Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.95      0.90      1188
           1       0.15      0.05      0.07       205

    accuracy                           0.82      1393
   macro avg       0.50      0.50      0.49      1393
weighted avg       0.75      0.82      0.78      1393
```

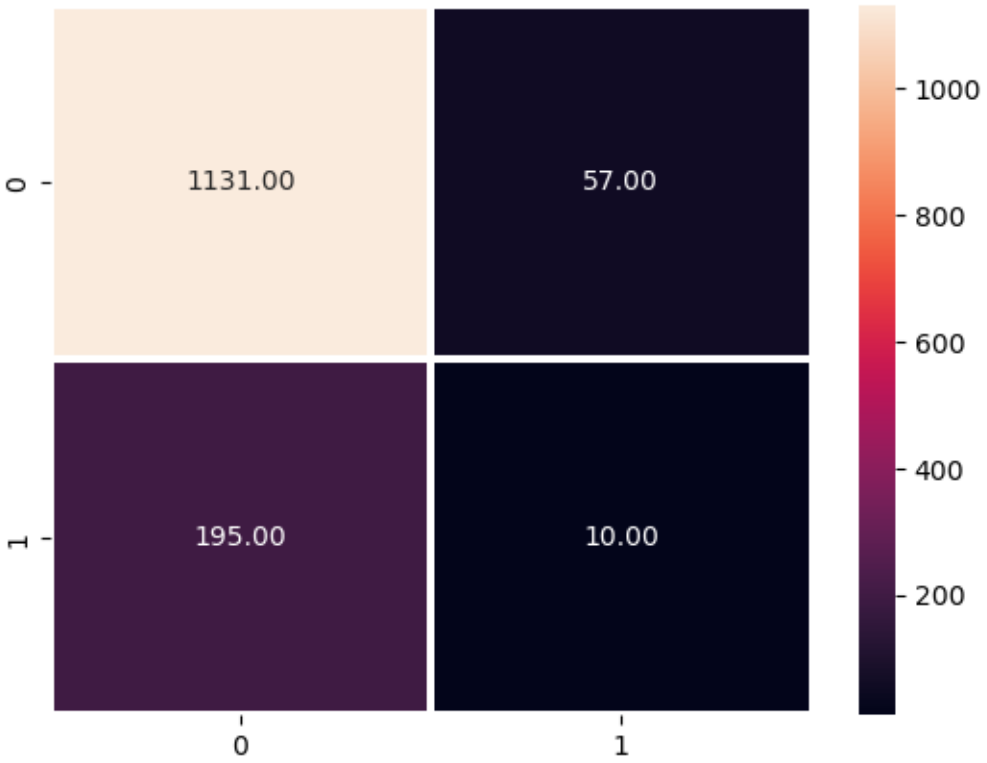
[41]:
```
matrix = confusion_matrix(y_pred = Y_pred,y_true = Y_test)
matrix
```

[41]:
```
array([[1131,   57],
       [ 195,   10]], dtype=int64)
```

Vizualization

[43]:
```
sns.heatmap(matrix,annot=True,fmt=".2f",linewidths=1.5)
```

[43]: <Axes: >



Ofcourse the model seems to be biased on ham messages for this particular data set since, we have only limited spam messages. But excluding `NLP` seems to be providing better results for this particular dataset……refer: https://youtu.be/nHIUYwN-5rM