FLIP ROBO

MALIGNANT COMMENTS CLASSIFIER PROJECT

Submitted by:
SARANYA M

ACKNOWLEDGMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many Google references.

Google Search for E-MAIL SPAM Classifier Project

https://www.kaggle.com/code/niteshk97/email-spam-classifier

I would like to express my special gratitude and thanks to our institute DataTrained & others seen unseen hands which have given us direct & indirect help in completion of this project. With help of their brilliant guidance and encouragement, I was able to complete my tasks properly and were up to the mark in all the tasks assigned. During the process, I got a chance to see the stronger side of my technical and non-technical aspects and also strengthen my concepts.

INTRODUCTION

Problem Statement

The project titled “Email Spam classification” is implemented using the CRISP-DM methodology. You will get to know Business understanding, Data Understanding (Data Description and Exploration), Data Preparation, Modelling, and Evaluation steps. Project is implemented using Python class object-based style. Email spam detection is done using machine learning algorithms Naive Bayes and SVM (Support vector machines). Further, it shows the complete program flow for Python-based email spam classifier implementation such as Data Retrieval Flow, Data Visualization Flow, Data Preparation Flow, Modeling, and Evaluation Flow. Also, including the section regarding Data ethics. In case you want to understand and demystify the python code using a top-down approach visit the following link email spam classification and detection python code.

Business Understanding

Most of us consider spam emails as one which is annoying and repetitively used for purpose of advertisement and brand promotion. We keep on blocking such email-ids but it is of no use as spam emails are still prevalent. Some major categories of spam emails that are causing great risk to security, such as fraudulent e-mails, identify theft, hacking, viruses, and malware. In order to deal with spam emails, we need to build a robust real-time email spam classifier that can efficiently and correctly flag the incoming mail spam, if it is a spam message or looks like a spam message. The latter will further help to build an Anti-Spam Filter.

Google and other email services are providing utility for flagging email spam but are still in the infancy stage and need regular feedback from the end-user. Also, popular email services such as

Gmail, Yandex, yahoo mail, etc provide basic services as free to the end-user and that of course comes with EULA. There is a great scope in building email spam classifiers, as the private companies run their own email servers and want them to be more secure because of the confidential data, in such cases email spam classifier solutions can be provided to such companies.

Multilabel vs Multiclass classification

For classifying a given message, first we preprocess it. For each word w in the processed messaged we find a product of P(w|spam). If w does not exist in the train dataset we take TF(w) as 0 and find P(w|spam) using above formula. We multiply this product with P(spam) The resultant product is the P(spam|message). Similarly, we find P(ham|message). Whichever probability among these two is greater, the corresponding tag (spam or ham) is assigned to the input message. Note than we are not dividing by P(w) as given in the formula. This is because both the numbers will be divided by that and it would not affect the comparison between the two.

Understanding Spam Emails

Let's suppose you got up in the morning and opened your Gmail and found a mail saying something like:
"Hey, you have won an iPhone 10 in the luck draw conducted by amazon yesterday. To receive the prize please log in to your account and claim your gift."
Seeing mail you first checked the sender and you found him to be genuine, and you happily rushed to the site and logged in and found there was no prize at all. Feeling sad you returned to your works.
A few hours later you received a message stating there has been a recent transaction from your bank account and you

are shocked how it happened. After telling the incident to the bank, they told you have been spammed and millions are facing the same difficulty, sounds terrible right!

Spam emails are unwanted emails share in bulk intending to gather data/ do phishing/ perform social engineering/ start an attack and a lot more – mostly for bad causes. Usually, these are in form of advertisement and marketing stuff.

Since these are sent in bulk, each one will have a similar underlying pattern and format, so what the mechanism of google does is finds these underlying patterns and separates them.

This method is generally called spam classification and uses a model which is trained on spam and not a spam set of data.

Statement

We have a message m = (w*1*, w*2*, ... , w*n*), where (w*1*, w*2*, ... , w*n*) is a set of unique words contained in the message. We need to find

$$P(spam|w1 \cap w2 \cap ... \cap wn) = \frac{P(w1 \cap w2 \cap ... \cap wn|spam). P(spam)}{P(w1 \cap w2 \cap ... \cap wn)}$$

If we assume that occurrence of a word are independent of all other words, we can simplify the above expression to

$$\frac{P(w1|spam). P(w2|spam)... P(wn|spam). P(spam)}{P(w1). P(w2).. P(wn)}$$

In order to classify we have to determine which is greater

$$P(spam|w1 \cap w2 \cap ... \cap wn) \ versus \ P(\sim spam|w1 \cap w2 \cap ... \cap wn)$$

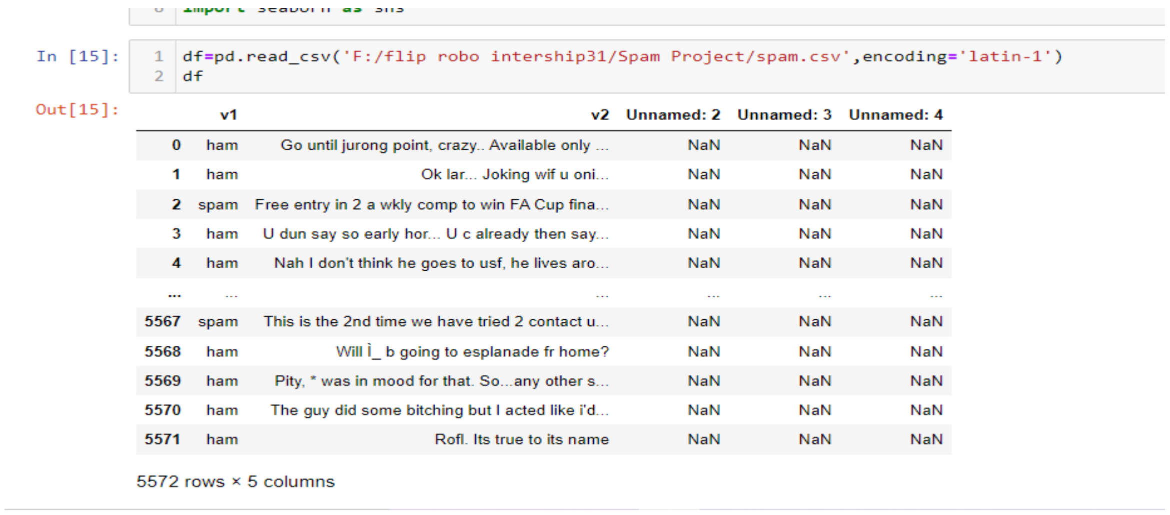
Data understanding

The email spam classifier focuses on either header, subject, and content of the email. In this project, we are focusing mainly on the subject and content of the email.

To download the email spam classification dataset files and complete code and visit the link email spam detection and classification project GitHub repository.

Data Description

The dataset contains two columns. The total corpus of 5728 documents. The descriptive feature consists of text. The target feature consists of two classes ham and spam, the column name is spam. The classes are labeled for each document in the data set and represent our target feature with a binary string-type alphabet of {ham; spam}. Classes are further mapped to integer 0 (ham) and 1 (spam).



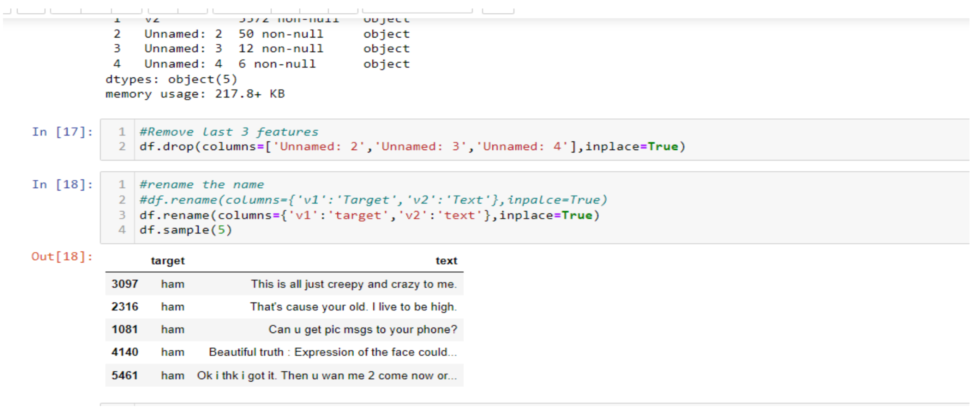
```
import seaborn as sns
```

```
In [15]:  df=pd.read_csv('F:/flip robo intership31/Spam Project/spam.csv',encoding='latin-1')
          df
```

Out[15]:

	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
...	...	...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...	NaN	NaN	NaN
5568	ham	Will Ì_ b going to esplanade fr home?	NaN	NaN	NaN
5569	ham	Pity, * was in mood for that. So...any other s...	NaN	NaN	NaN
5570	ham	The guy did some bitching but I acted like i'd...	NaN	NaN	NaN
5571	ham	Rofl. Its true to its name	NaN	NaN	NaN

5572 rows × 5 columns



```
 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 [17]:  #Remove last 3 features
          df.drop(columns=['Unnamed: 2','Unnamed: 3','Unnamed: 4'],inplace=True)

In [18]:  #rename the name
          #df.rename(columns={'v1':'Target','v2':'Text'},inpalce=True)
          df.rename(columns={'v1':'target','v2':'text'},inplace=True)
          df.sample(5)
```

Out[18]:

	target	text
3097	ham	This is all just creepy and crazy to me.
2316	ham	That's cause your old. I live to be high.
1081	ham	Can u get pic msgs to your phone?
4140	ham	Beautiful truth : Expression of the face could...
5461	ham	Ok i thk i got it. Then u wan me 2 come now or...

ANALYTICAL PROBLEM FRAMING

Model Building Phase

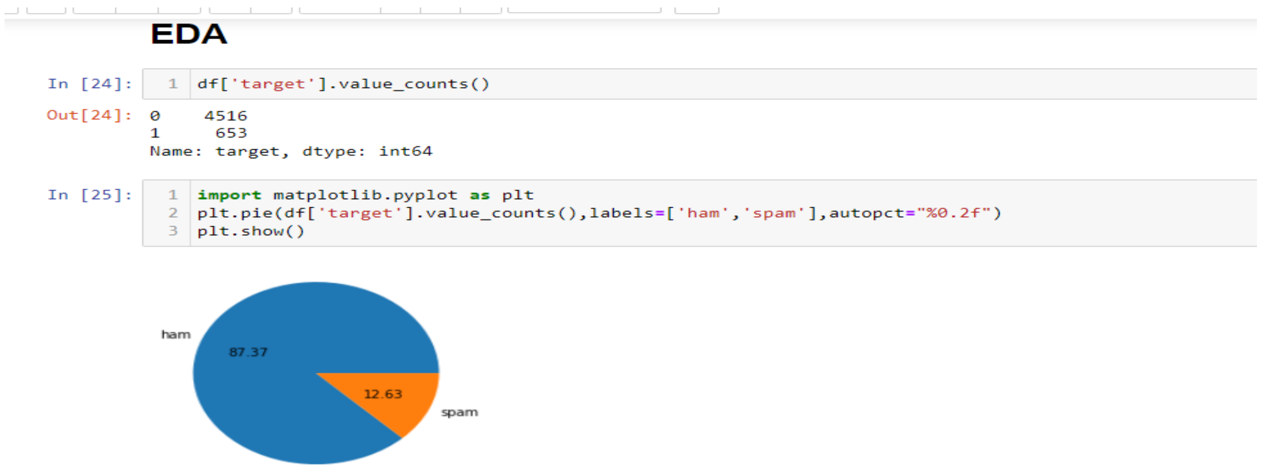
You need to build a machine learning model. Before model building do all data pre-processing steps involving NLP. Try different models with different hyper parameters and select the best model. Follow the complete life cycle of data science. Include all the steps like-

1. Data Cleaning
2. Exploratory Data Analysis
3. Data Pre-processing
4. Model Building
5. Model Evaluation
6. Selecting the best model

Spam Detection Exploratory Data Analysis

We will now do some of the Exploratory – Data Analysis to check how data is distributed along 2 categories. This will give us a feel if we need to do some type of preprocessing over data or is it on the same scale.

To perform this operation we will just be grouping the data based on category and call value_counts() method on it like:

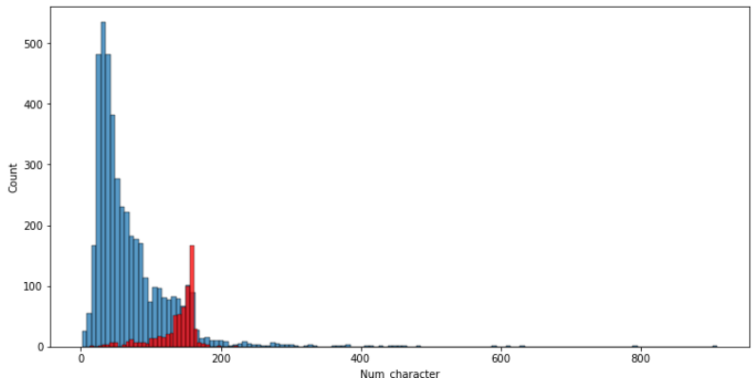


EDA

```
In [24]:  df['target'].value_counts()
Out[24]:  0     4516
          1      653
          Name: target, dtype: int64

In [25]:  import matplotlib.pyplot as plt
          plt.pie(df['target'].value_counts(),labels=['ham','spam'],autopct="%0.2f")
          plt.show()
```

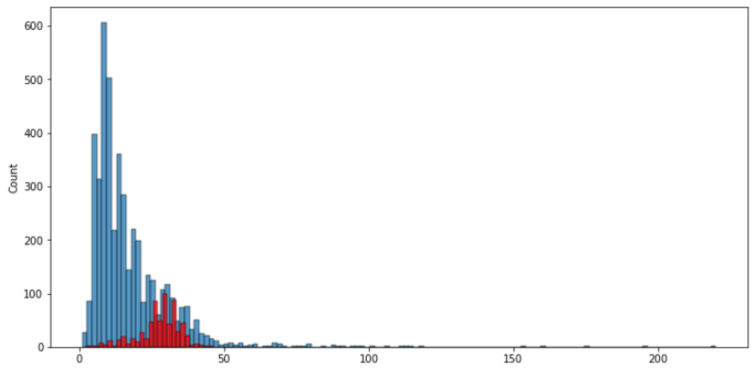
In [29]:
```python
plt.figure(figsize=(12,6))
sns.histplot(df[df['target']==0]['Num_character'])
sns.histplot(df[df['target']==1]['Num_character'],color='red')
```

Out[29]:
```
<AxesSubplot:xlabel='Num_character', ylabel='Count'>
```



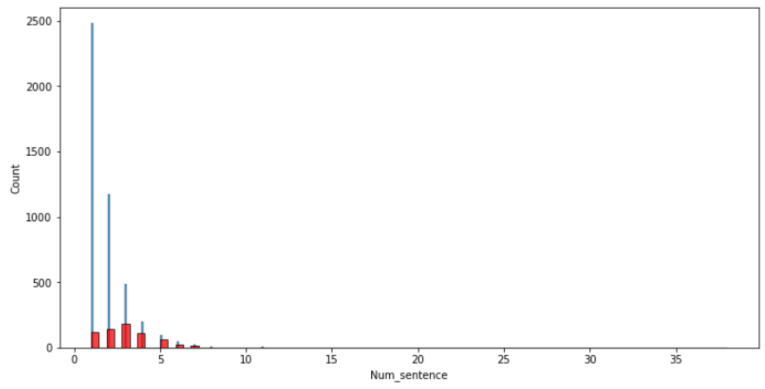
In [30]:
```python
plt.figure(figsize=(12,6))
sns.histplot(df[df['target']==0]['Num_words'])
sns.histplot(df[df['target']==1]['Num_words'],color='red')
```

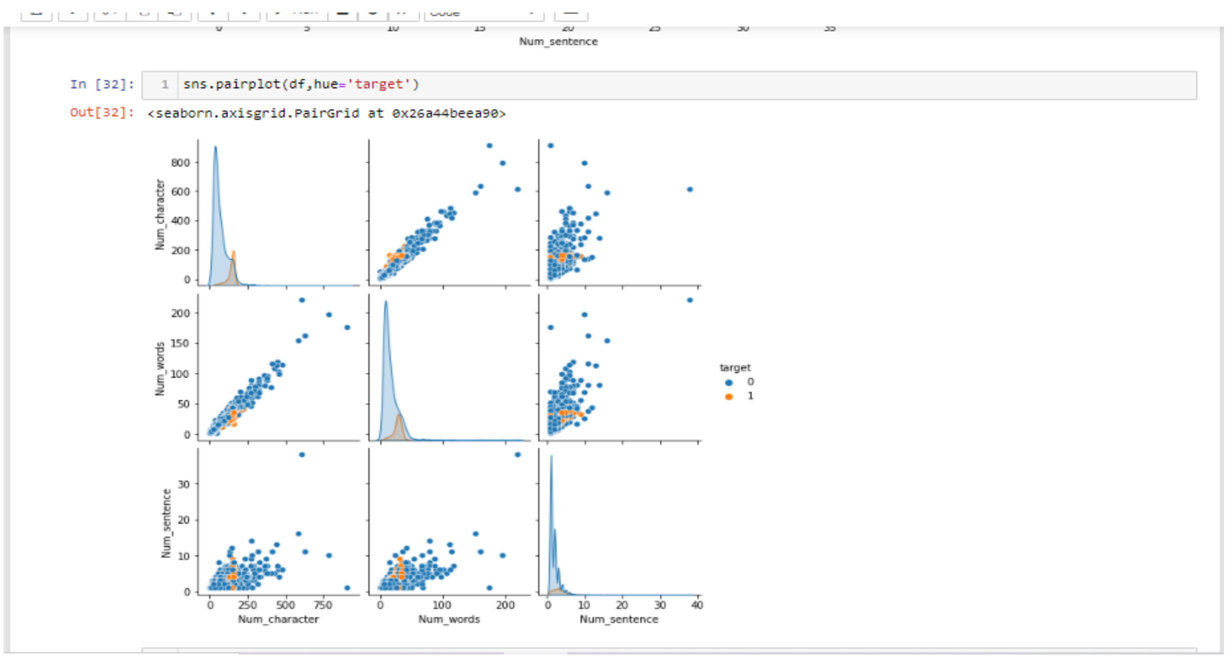
Out[30]:
```
<AxesSubplot:xlabel='Num_words', ylabel='Count'>
```



```python
sns.histplot(df[df['target']==0]['Num_sentence'])
sns.histplot(df[df['target']==1]['Num_sentence'],color='red')
```

Out[31]:
```
<AxesSubplot:xlabel='Num_sentence', ylabel='Count'>
```





```
In [32]:  sns.pairplot(df,hue='target')
Out[32]: <seaborn.axisgrid.PairGrid at 0x26a44beea90>
```

Preprocessing of Spam Detection Data

One Hot Encoding Categories

As can be seen, we have only text as categorical data, and the model doesn't understand them. So instead of text, we can just assign integer labels to our class ham and spam as 0 and 1 respectively, and store it in new column spam. This is called- Hot-Encoding

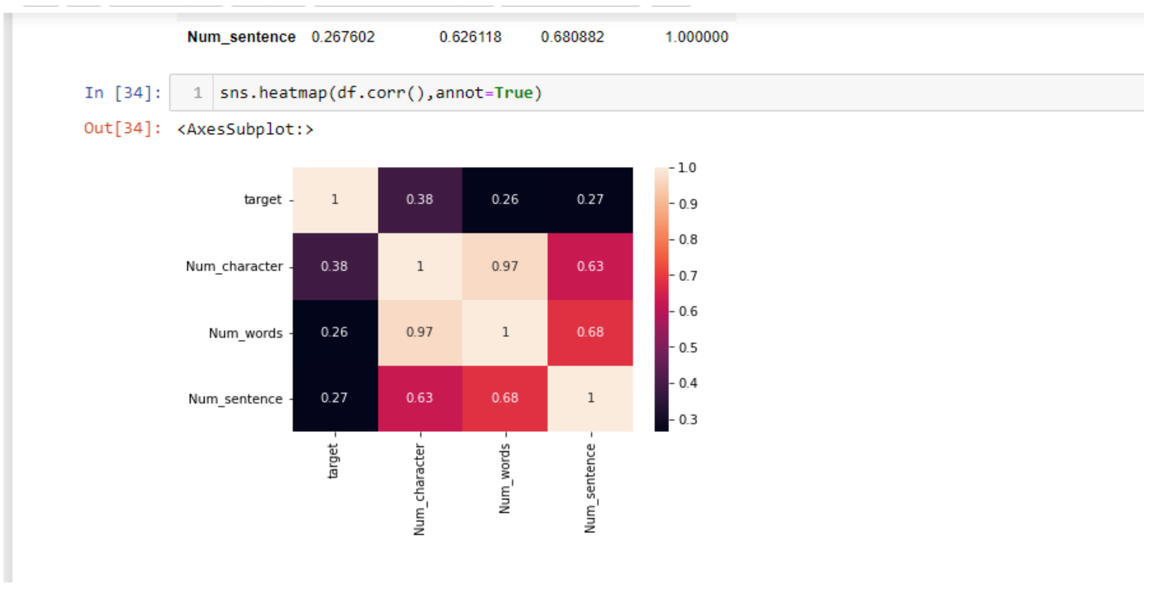
To achieve this we will just be filtering the column category and perform operations:

1 – If a category is a ham/ not spam

0 – if the category is spam

The best part is that we can use the one-liner lambda function to achieve the following result and apply it to the dataframe for all values.

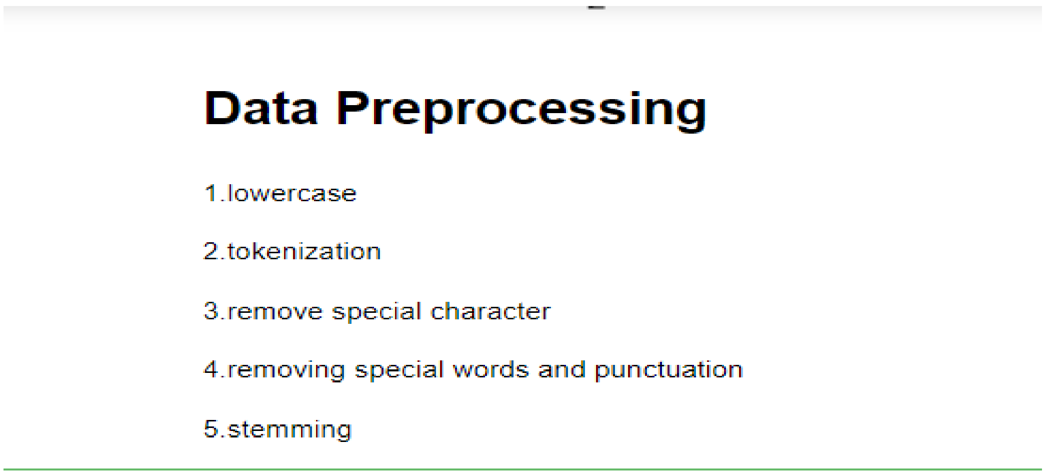
Lambda Fn Syntax = [lambda x : value expression else value]



```
Num_sentence  0.267602  0.626118  0.680882  1.000000

In [34]:  sns.heatmap(df.corr(),annot=True)
Out[34]: <AxesSubplot:>
```

	target	Num_character	Num_words	Num_sentence
target	1	0.38	0.26	0.27
Num_character	0.38	1	0.97	0.63
Num_words	0.26	0.97	1	0.68
Num_sentence	0.27	0.63	0.68	1

As we see multicollinearity here, we cannot use all three columns instead we shall use only one and that should be num_characters has it has highest correlation with message_type.
Since we are working to classify Spam vs Non-Spam emails, it is crucial for us to avoid False-Positive classification, i.e., classifying a Non-Spam email as a Spam email. For this, we will check distribution of our classification.



Data Preprocessing

1.lowercase

2.tokenization

3.remove special character

4.removing special words and punctuation

5.stemming

Preprocessing:

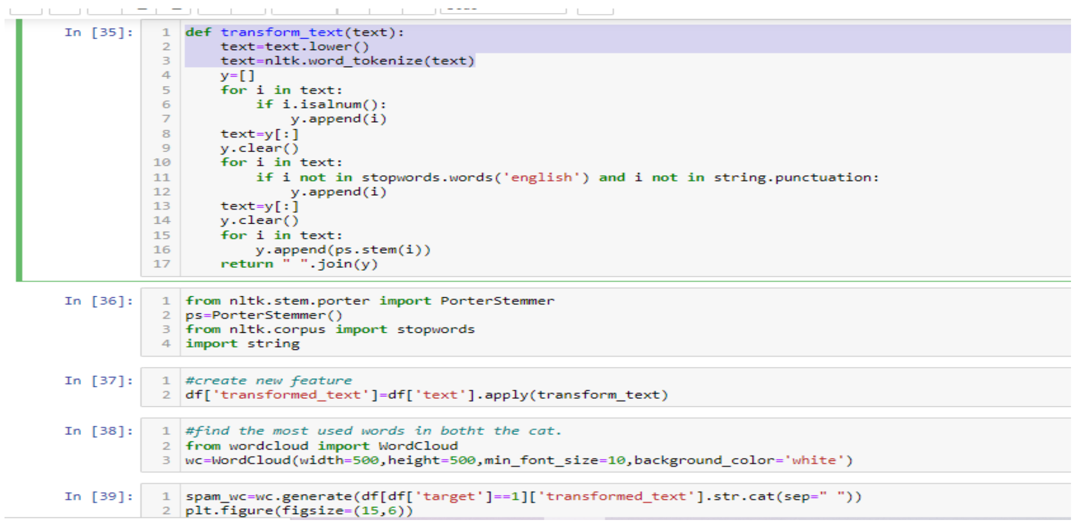
Before starting with training we must preprocess the messages. First of all, we shall make all the character lowercase. This is because ‘free’ and ‘FREE’ mean the same and we do not want to treat them as two different words.

Then we tokenize each message in the dataset. Tokenization is the task of splitting up a message into pieces and throwing away the punctuation characters.

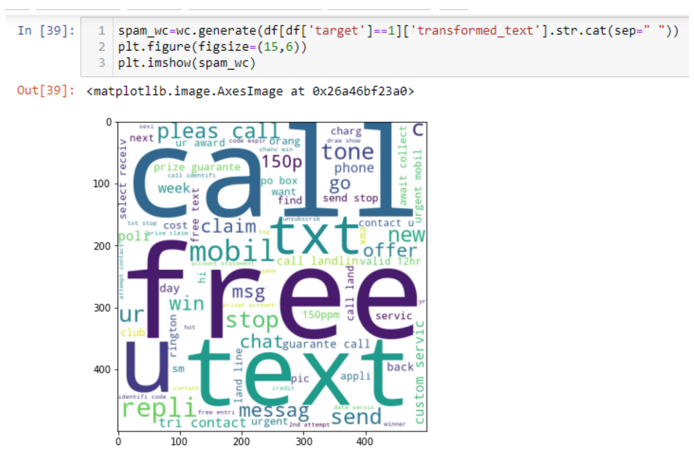
The words like ‘go’, ‘goes’, ‘going’ indicate the same activity. We can replace all these words by a single word ‘go’. This is called stemming. We are going to use Porter Stemmer, which is a famous stemming algorithm.

We then move on to remove the stop words. Stop words are those words which occur extremely frequently in any text. For example words like ‘the’, ‘a’, ‘an’, ‘is’, ‘to’ etc. These words do not give us any information about the content of the text. Thus it should not matter if we remove these words for the text.

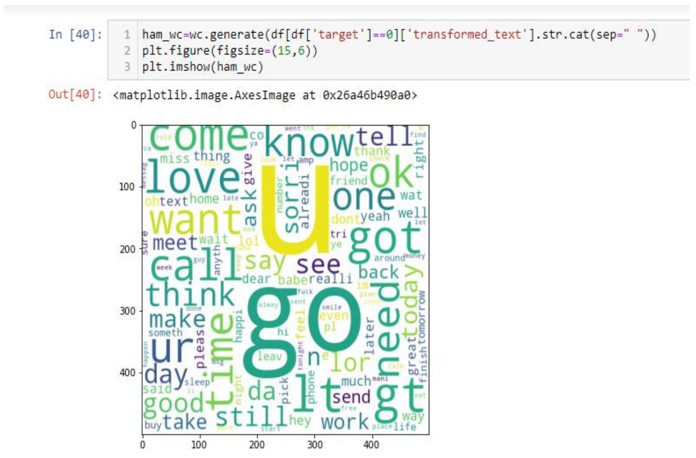
Optional: You can also use n-grams to improve the accuracy. As of now, we only dealt with 1 word. But when two words are together the meaning totally changes. For example, ‘good’ and ‘not good’ are opposite in meaning. Suppose a text contains ‘not good’, it is better to consider ‘not good’ as one token rather than ‘not’ and ‘good’. Therefore, sometimes accuracy is improved when we split the text into tokens of two (or more) words than only word.



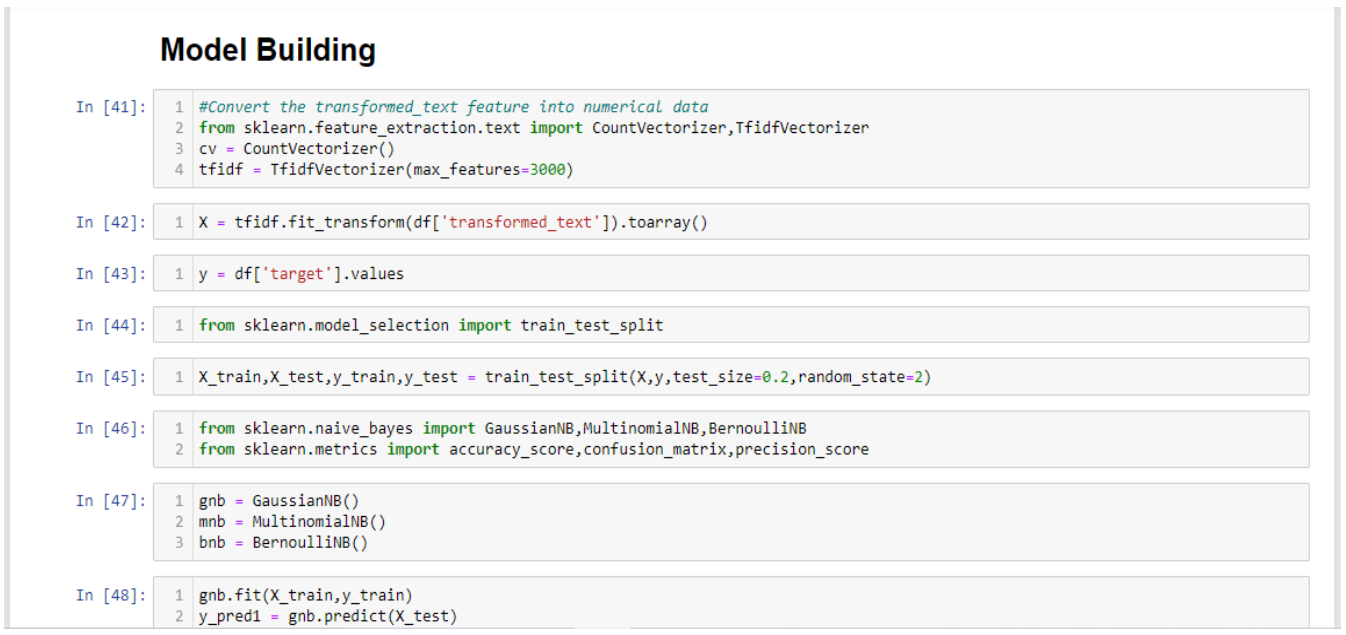
```
In [35]:  def transform_text(text):
               text=text.lower()
               text=nltk.word_tokenize(text)
               y=[]
               for i in text:
                   if i.isalnum():
                       y.append(i)
               text=y[:]
               y.clear()
               for i in text:
                   if i not in stopwords.words('english') and i not in string.punctuation:
                       y.append(i)
               text=y[:]
               y.clear()
               for i in text:
                   y.append(ps.stem(i))
               return " ".join(y)

In [36]:  from nltk.stem.porter import PorterStemmer
           ps=PorterStemmer()
           from nltk.corpus import stopwords
           import string

In [37]:  #create new feature
           df['transformed_text']=df['text'].apply(transform_text)

In [38]:  #find the most used words in botht the cat.
           from wordcloud import WordCloud
           wc=WordCloud(width=500,height=500,min_font_size=10,background_color='white')

In [39]:  spam_wc=wc.generate(df[df['target']==1]['transformed_text'].str.cat(sep=" "))
           plt.figure(figsize=(15,6))
```



```
In [39]:  spam_wc=wc.generate(df[df['target']==1]['transformed_text'].str.cat(sep=" "))
           plt.figure(figsize=(15,6))
           plt.imshow(spam_wc)

Out[39]: <matplotlib.image.AxesImage at 0x26a46bf23a0>
```



```
In [40]:  ham_wc=wc.generate(df[df['target']==0]['transformed_text'].str.cat(sep=" "))
           plt.figure(figsize=(15,6))
           plt.imshow(ham_wc)

Out[40]: <matplotlib.image.AxesImage at 0x26a46b490a0>
```



Model Building

```
In [41]:  #Convert the transformed_text feature into numerical data
           from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
           cv = CountVectorizer()
           tfidf = TfidfVectorizer(max_features=3000)

In [42]:  X = tfidf.fit_transform(df['transformed_text']).toarray()

In [43]:  y = df['target'].values

In [44]:  from sklearn.model_selection import train_test_split

In [45]:  X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)

In [46]:  from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
           from sklearn.metrics import accuracy_score,confusion_matrix,precision_score

In [47]:  gnb = GaussianNB()
           mnb = MultinomialNB()
           bnb = BernoulliNB()

In [48]:  gnb.fit(X_train,y_train)
           y_pred1 = gnb.predict(X_test)
```

Building a Model using Naive Bayes

As it is known that on Textual Data Naive Bayes Algorithm works the best hence we will use it but along the way also compare it with different algorithms

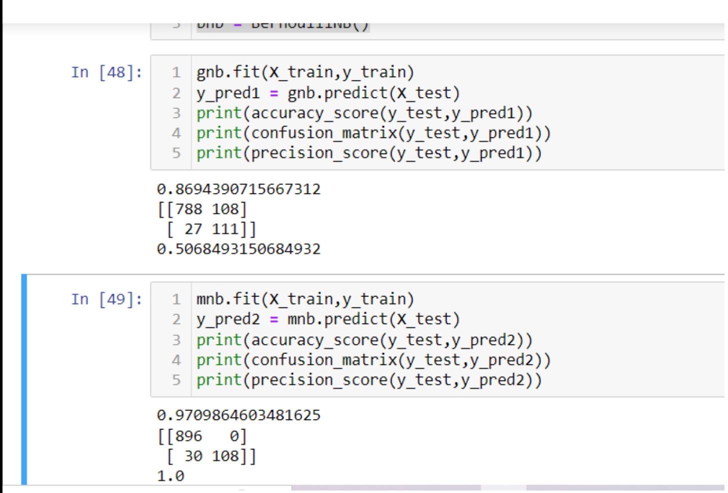
Input is categorical Output is Numerical.

But as we know in the Naive Bayes algorithm the input columns should be numerical so we have to convert (VECTORIZE) the column.

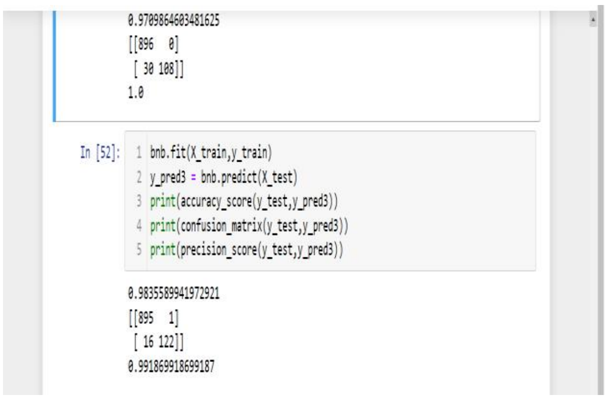
How to vectorize:

- Bag of Words
- TFIDF
- Word2vec

After trying out different techniques, I came to the conclusion that TFIDF vectorization gives the best accuracy and precision score so we will be using it.



```
In [48]:  gnb.fit(X_train,y_train)
          y_pred1 = gnb.predict(X_test)
          print(accuracy_score(y_test,y_pred1))
          print(confusion_matrix(y_test,y_pred1))
          print(precision_score(y_test,y_pred1))

          0.8694390715667312
          [[788 108]
           [ 27 111]]
          0.5068493150684932

In [49]:  mnb.fit(X_train,y_train)
          y_pred2 = mnb.predict(X_test)
          print(accuracy_score(y_test,y_pred2))
          print(confusion_matrix(y_test,y_pred2))
          print(precision_score(y_test,y_pred2))

          0.9709864603481625
          [[896   0]
           [ 30 108]]
          1.0
```



```
          0.9709864603481625
          [[896   0]
           [ 30 108]]
          1.0

In [52]:  bnb.fit(X_train,y_train)
          y_pred3 = bnb.predict(X_test)
          print(accuracy_score(y_test,y_pred3))
          print(confusion_matrix(y_test,y_pred3))
          print(precision_score(y_test,y_pred3))

          0.9835589941972921
          [[895   1]
           [ 16 122]]
          0.991869918699187
```

Gaussian Naive Bayes is useful when working with continuous values which probabilities can be modeled using a Gaussian distribution

The multinomial naive Bayes performs better than the Gaussian variant and the result is not really surprising. In fact, each sample can be thought as a feature vector derived from a dictionary of 64 symbols. The value can be the count of each occurrence, so a multinomial distribution can better fit the data, while a Gaussian is slightly more limited by its mean and variance.

Hence we finalise the Model with MNB(Multinomial Naive Bayes) and TFIDF Vectorization.

Conclusion

The classifier accurately identified the email messages as spam or not spam with 99.2 % accuracy on the test data !

It is very unusual to having 100% success from a model. Obviously, it is due to small training and test dataset. I have tested my own emails using the model. It turned out that it is not as effective as my existing paid spam filter. It makes sense. There are many ways we can improve the model. If the model trains with sufficient data, it will deliver more accurate results.

With the increase usage of emails, this study focuses on using automated ways to detect spam emails written. The study uses various machine learning and deep learning algorithms to detect them. In the study, a translated emails dataset including spam and ham emails is generated from Kaggle, which is preprocessed for various approaches. Accuracy, precision, recall, F-measure, ROC-AUC, and model loss are used as comparative measures to examine performance. The study concludes that deep learning models are more successful in classifying spam emails. Comparatively, LSTM algorithm has a high accuracy rate of around 98% with low model loss rate of 5%. Even though LSTM takes a little longer to train than CNN, SVM, or Naive Bayes, its efficiency and accuracy rate are far better than those of the other approaches. The creation of an actual dataset of emails can be considered as a viable future task. In addition, more recent artificial intelligent approaches may also be considered to detect spams.

A supportive tool using a browser plugin or API can be built for companies running their own email servers so that they can keep a check on emails and can identify and flag spam emails. Such a supportive tool can be used in conjunction with existing email service providers as well.

Data Ethics

There are many ethical and legal issues that can really take a toll on designing such models. Bank and Investment organizations that run their own email servers have confidential emails and identity information related to customers. Using such confidential information for the predictive analysis required to safeguard the client data with the care of a professional fiduciary. Need to protect the customer data from both intentional and inadvertent disclosure, also protecting it from misuse.

Also, while implementation it is possible to generate false positive, which means the emails which are not spam fall into the spam category. An important piece of information a company can miss if the user's legit email is marked as spam. A client or user who is a loyal customer, his email can be marked spam, which is an ethical issue.