Project Report: Email Spam Detection project

Created by: Kumar Gourabh | Fliprobo Internship 10| Batch 1823 – Datatrained |

Problem Statement: Build & Train a model to identify spam e-mails based on subject and message data.

Use case:

=========

You were recently hired in start-up company and you were asked to build a system to identify spam emails.

Perform all necessary actions not only limited to,

1. Data Preparation
2. Building word dictionary
3. Feature extraction
4. Training classifiers
5. Testing
6. Performance evaluation using multiple metrics

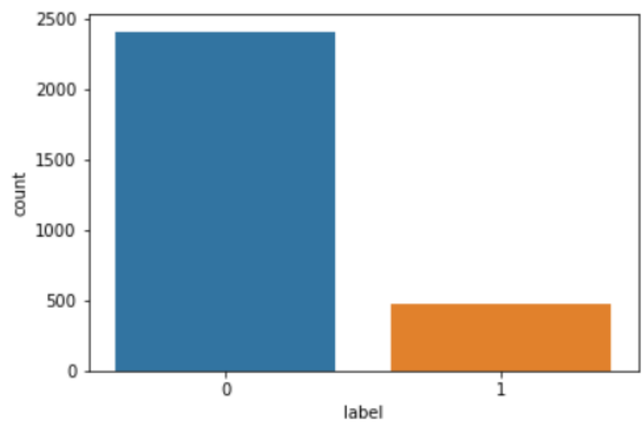
Solution:

Step 1: Importing important Libraries and loading the given dataset:

Imported pandas, numpy, matplotlib and seaborn. Also imported warnings to ignore the warnings. There are 2 features ‘**subject’** and ‘**message’** and a **label** having values 0 and 1.

Here 0 indicates not-spam (also called ‘ham’) and 1 indicates ‘spam’

Dataset contains 2893 rows and 3 columns.




```
spam ratio =  17.0 %
ham ratio  =  83.0 %
```

In [1]:
```python
#Import libs
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
sms = pd.read_csv('messages.csv')
sms.head()
```

Out[2]:

	subject	message	label
0	job posting - apple-iss research center	content - length : 3386 apple-iss research cen...	0
1	NaN	lang classification grimes , joseph e . and ba...	0
2	query : letter frequencies for text identifica...	i am posting this inquiry for sergei atamas (...	0
3	risk	a colleague and i are researching the differin...	0
4	request book information	earlier this morning i was on the phone with a...	0

In [3]:
```python
sms.shape
```

Out[3]:
```
(2893, 3)
```

In [4]:
```python
sms.label.value_counts()
```

Out[4]:
```
0    2412
1     481
Name: label, dtype: int64
```

Added 2 new columns - **length_message** (Length of email body) & **length_subject** (Length of subject)

	subject	message	label	length_message	length_subject
0	job posting - apple-iss research center	content - length : 3386 apple-iss research cen...	0	2856	39.0
1	NaN	lang classification grimes , joseph e . and ba...	0	1800	NaN
2	query : letter frequencies for text identifica...	i am posting this inquiry for sergei atamas (...	0	1435	50.0
3	risk	a colleague and i are researching the differin...	0	324	4.0
4	request book information	earlier this morning i was on the phone with a...	0	1046	24.0

After doing pre-processing steps and cleaning the text, I will see the difference in total length of message and subject.

I have replaced the empty subjects and their lengths as shown below:

In [9]:
```python
# Let's replace empty subjects with 'Empty' and length as 0
sms['subject'].fillna("Empty",inplace=True)
sms['length_subject'].fillna(0,inplace=True)
```

Step 2: Pre-Processing

The steps are converting the message and subject to lower strings. Then after identifying email address formats, I've converted it to the word 'emailaddress' and similarly substituted phone number, webaddress, dollars and numbers. Also, replaced punctuation and white space with blank space. Further, removed leading and trailing whitespaces.

```
#Converting strings to lowercase
sms['message'] = sms['message'].str.lower()
sms['subject'] = sms['subject'].str.lower()
```

```
cols=['message','subject']
for j in cols:
    # Replace email addresses with 'email'
    sms[j] = sms[j].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',
                                   'emailaddress')

    # Replace URLs with 'webaddress'
    sms[j] = sms[j].str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$',
                                    'webaddress')

    # Replace money symbols with 'moneysymb' (£ can by typed with ALT key + 156)
    sms[j] = sms[j].str.replace(r'£|\$', 'dollers')

    # Replace 10 digit phone numbers (formats include paranthesis, spaces, no spaces, dashes)
    sms[j] = sms[j].str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$',
                                    'phonenumber')


    # Replace numbers with 'numbr'
    sms[j] = sms[j].str.replace(r'\d+(\.\d+)?', 'numbr')

    # Remove punctuation
    sms[j] = sms[j].str.replace(r'[^\w\d\s]', ' ')

    # Replace whitespace between terms with a single space
    sms[j] = sms[j].str.replace(r'\s+', ' ')

    # Remove leading and trailing whitespace
    sms[j] = sms[j].str.replace(r'^\s+|\s+?$', '')
```

Next, I have removed the stopwords, which do not add any meaning or context.

```
# Remove stopwords
import string
import nltk
from nltk.corpus import  stopwords

stop_words = set(stopwords.words('english') + ['u', 'ü', 'ur', '4', '2', 'im', 'dont', 'doin', 'ure'])

sms['message'] = sms['message'].apply(lambda x: ' '.join(
    term for term in x.split() if term not in stop_words))

sms['subject'] = sms['subject'].apply(lambda x: ' '.join(
    term for term in x.split() if term not in stop_words))
```

Step 3: Tokenizing

```
from nltk.tokenize import RegexpTokenizer
tokenizer=RegexpTokenizer(r'\w+')
sms['message'] = sms['message'].apply(lambda x: tokenizer.tokenize(x.lower()))
sms['subject'] = sms['subject'].apply(lambda x: tokenizer.tokenize(x.lower()))
sms.head()
```

	subject	message	label	length_message	length_subject
0	[job, posting, apple, iss, research, center]	[content, length, numbr, apple, iss, research,...	0	2856	39.0
1	[empty]	[lang, classification, grimes, joseph, e, barb...	0	1800	0.0
2	[query, letter, frequencies, text, identificat...	[posting, inquiry, sergei, atamas, satamas, um...	0	1435	50.0
3	[risk]	[colleague, researching, differing, degrees, r...	0	324	4.0
4	[request, book, information]	[earlier, morning, phone, friend, mine, living...	0	1046	24.0

As shown above, the messages and subjects have been converted into word tokens.

Step 4: Getting Root Words

In the below function, I am first selecting the tokens having length greater than 2. Then, I am using Lemmatizing to get the root words and later using Snowball Stemming, to further reduce characters.

```
# writing function for the entire dataset
# Lemmatizing and then Stemming with Snowball to get root words and further reducing characters

from nltk.stem import SnowballStemmer, WordNetLemmatizer
stemmer = SnowballStemmer("english")
import gensim
def lemmatize_stemming(text):
    return stemmer.stem(WordNetLemmatizer().lemmatize(text,pos='v'))

#Tokenize and Lemmatize
def preprocess(text):
    result=[]
    for token in text:
        if len(token)>=3:
            result.append(lemmatize_stemming(token))

    return result
```

Step 5: Storing list of words for Subject and Email messages

```
processed_docs_subject = []

for doc in sms.subject:
    processed_docs_subject.append(preprocess(doc))
```

```
print(len(processed_docs_subject))
processed_docs_subject[:3]
```

```
2893

[['job', 'post', 'appl', 'iss', 'research', 'center'],
 ['empti'],
 ['queri', 'letter', 'frequenc', 'text', 'identif']]
```

```
processed_docs_message = []

for doc in sms.message:
    processed_docs_message.append(preprocess(doc))
```

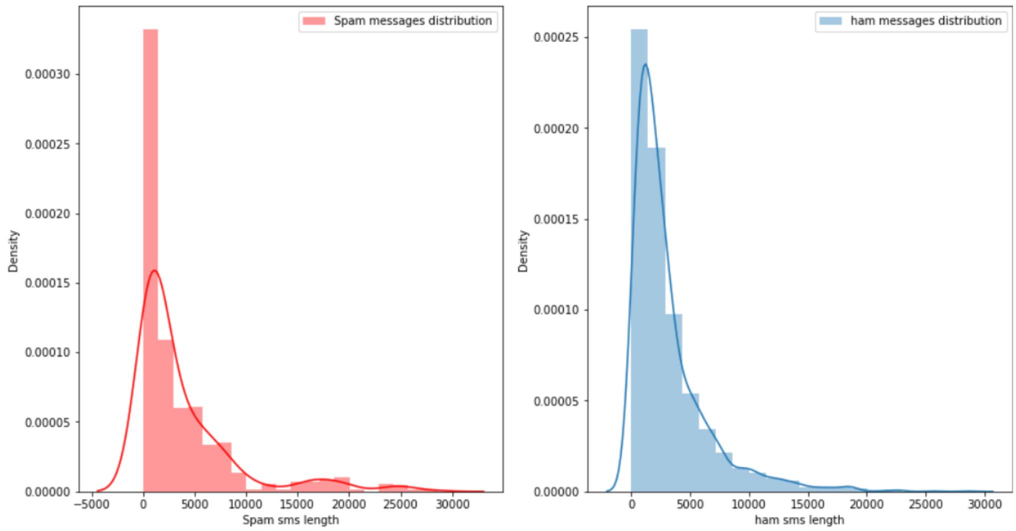
Step 6: Analysis

After cleaning the messages and subjects, the new lengths were calculated and below are the results showing total cleaned and total original lengths.

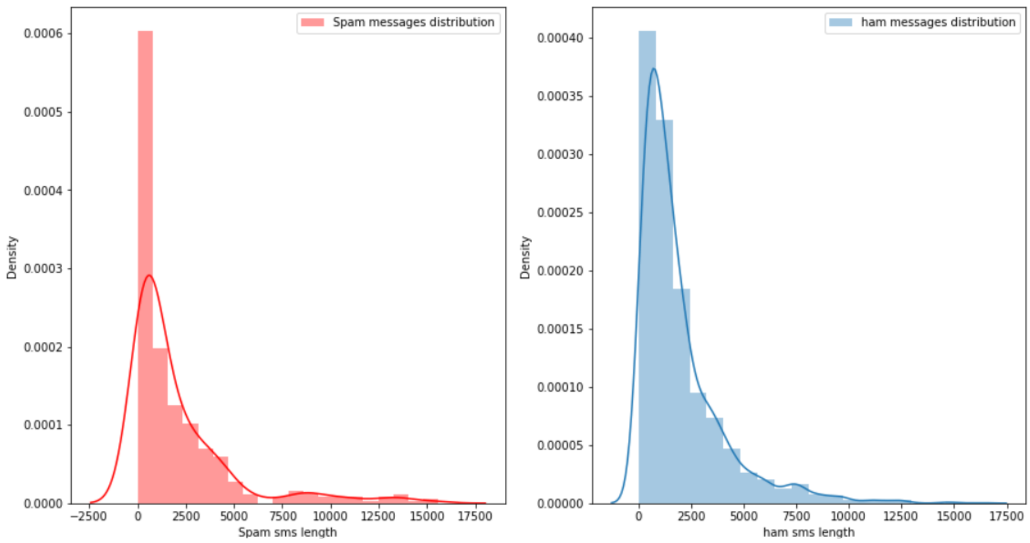
```
# Total length removal
print ('Original Length of message', sms.length_message.sum())
print ('Clean Length of message', sms.clean_length_message.sum())
print ('Original Length of subject', sms.length_subject.sum())
print ('Clean Length of subject', sms.clean_length_subject.sum())
```

```
Original Length of message 9344743
Clean Length of message 5680514
Original Length of subject 91663.0
Clean Length of subject 67148
```

Distribution Before Cleaning:



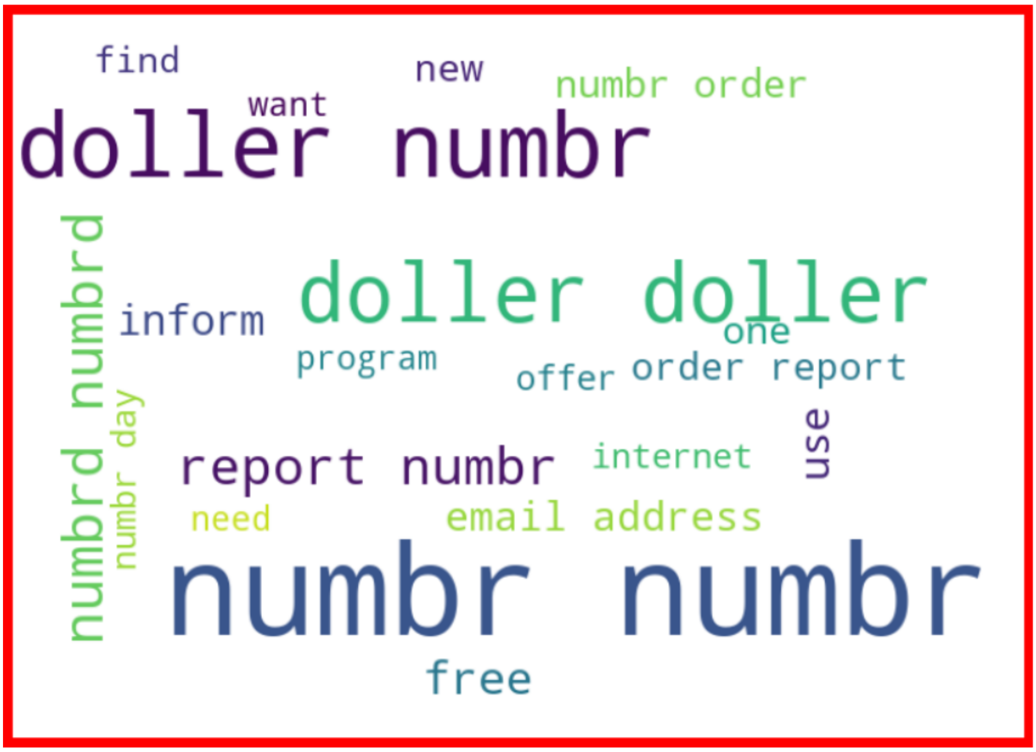
Distribution After Cleaning:



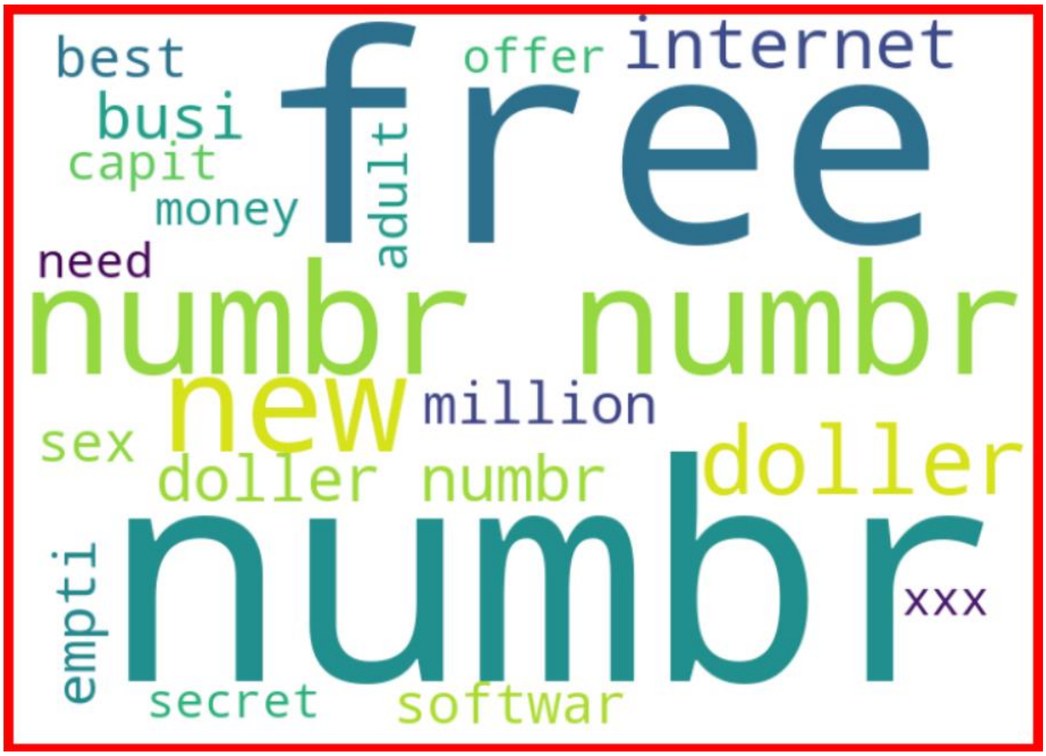
The above graphs show the distributions of email messages before and after cleaning. The range has reduced considerably. Also similar observations were made for subjects.

Getting a sense of all words using word-cloud:

Email Messages: The main words that are treated as spam, according to the below word-cloud are: doller, number, program, inform, report, free, order, and offer etc.



Email Subjects: The main words that are treated as spam, according to the below word-cloud are: free, best, number, new, empty, xxx, doller, internet, software, sex etc.



Step 7: Preparing Training and Test Data

```
# 1. Convert text into vectors using TF-IDF
# 2. Instantiate MultinomialNB classifier, SGDClassifier
# 3. Split feature and label
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

tf_vec = TfidfVectorizer()
features = tf_vec.fit_transform(sms['subject'] + sms['message'])

X = features
y = sms['label']
X.shape
```

```
(2893, 45239)
```

```
# Train and test splitting
X_train,x_test,Y_train,y_test = train_test_split(X,y,random_state=42)
```

Step 8: Model Testing and Prediction

I have used MultinomialNB and SGDClassifier to Model.

```
# Naive Bayes
naive = MultinomialNB()
naive.fit(X_train,Y_train)
y_pred= naive.predict(x_test)

print ('Accuracy = > ', accuracy_score(y_test,y_pred)*100)
```

```
Accuracy = >  82.18232044198895
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      1.00      0.90       585
           1       1.00      0.07      0.13       139

    accuracy                           0.82       724
   macro avg       0.91      0.54      0.52       724
weighted avg       0.85      0.82      0.75       724
```

```
#confusion matrix
conf_mat = confusion_matrix(y_test,y_pred)
conf_mat
```

```
array([[585,   0],
       [129,  10]], dtype=int64)
```

```
# SGDClassifier
sgd=SGDClassifier()
sgd.fit(X_train,Y_train)
y_pred_sgd= sgd.predict(x_test)

print ('Accuracy = > ', accuracy_score(y_test,y_pred_sgd)*100)
```

```
Accuracy = >  99.17127071823204
```

```
print(classification_report(y_test, y_pred_sgd))
#confusion matrix
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       585
           1       0.99      0.96      0.98       139

    accuracy                           0.99       724
   macro avg       0.99      0.98      0.99       724
weighted avg       0.99      0.99      0.99       724
```

```
#confusion matrix
conf_mat = confusion_matrix(y_test,y_pred_sgd)
conf_mat
```

```
array([[584,   1],
       [  5, 134]], dtype=int64)
```

As per above screenshots, we can see that SGD Classifier is giving better results.

Step 9: Model Selection and Results

Plotting the Confusion Matrix Heat Map

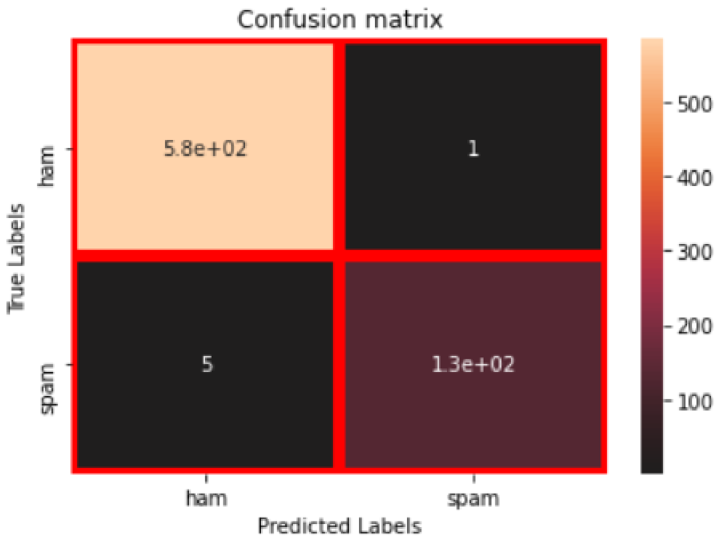
```
# plot confusion matrix heatmap
conf_mat = confusion_matrix(y_test,y_pred_sgd)

ax=plt.subplot()

sns.heatmap(conf_mat,annot=True,ax=ax,linewidths=5,linecolor='r',center=0)

ax.set_xlabel('Predicted Labels');ax.set_ylabel('True Labels')

ax.set_title('Confusion matrix')
ax.xaxis.set_ticklabels(['ham','spam'])
ax.yaxis.set_ticklabels(['ham','spam'])
plt.show()
```



Selected Model SGDClassifier with 99.17% accuracy and f1 score of 98%