FLIP ROBO

Detecting Spam SMS Using Natural Language Processing

Submitted by:

Aditya Paruleker

INTRODUCTION

- Business Problem Framing
  Today, spam has become a big internet issue and in recent times, it has been proven through statistics that spam has accounted more than 60% of all SMS movement. Furthermore, phishing spam messages has been causing problem for the security of end users, since they attempt to persuade them to surrender personal information such as passwords and bank account numbers, using parody messages which replicates the appearance of original message from trustworthy organizations. Data science can thus play a key role in this domain by protecting the users from such fraud SMS using various natural language processing techniques. Our goal is to build a prototype of spam filtering system that will classify spam SMS and ham (non spam) messages and thus help the client to distinguish between the two.

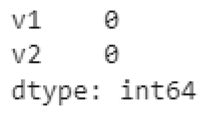
- Motivation for the Problem Undertaken
  Motivation behind this project is to help the users by alerting them of the possibility of the spam SMS being received. The information obtained from the model can be used to report such contact and thus enhance the security of the user.

Analytical Problem Framing

- Data Sources and their formats
  A collection of 5573 rows SMS spam messages was manually extracted from the Grumbletext Web site. This is a UK forum in which cell phone users make public claims about SMS spam messages, most of them without reporting the very spam message received.

- Preprocessing Done

a) Null Value Analysis



```
v1    0
v2    0
dtype: int64
```

No null values are detected in the dataset. Since all the data is of object type we skip the statistical analysis and move to data cleaning phase.

b) Encoding Target Variable
The target column is encoded into 0s and 1s.

c) Data Cleaning
- Strip if any html tags using beautiful soup html parser
- Remove all the .com and https links.
- Split the text and convert it to lower case.
- Remove all the numbers from the string.
- Using regex function, strip anything other than words and whitespaces like symbols.
- Eliminate the stopwords and punctuation from the string and then lemmatize the text.

- Remove any single character if remaining in the text.
- Delete rows with empty string.

d) **Separating Target and Input features**
Before proceeding with further preprocessing, the input features labelled x are separated from target features labelled y.

e) **Vectorize**
In NLP, word vectorization is a method i.e. used to map words or phrases from vocabulary to a corresponding vector of real numbers which used to find word predictions, word similarities/semantics. And this process of converting words into numbers is called Vectorization.

f) **TF – IDF (Term frequency-inverse document frequency)**
It evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document and the inverse document frequency of the word across a set of documents.

• Hardware and Software Requirements and Tools Used

Following libraries were used in this project,

Pandas
Created by Wes McKinney in 2008, this python library is widely used for data manipulation, analyzing and cleaning of data. Apart from this, it also helps in finding correlation between columns and in deleting rows.

Numpy
Created by Travis Oliphant in 2005, this python library provides an array object called ndarray i.e. up to 50x faster than traditional

python lists. It was has functions can be used in linear algebra, matrices etc

Seaborn
Seaborn is a high level interface based data visualization library that uses matplotlib library underneath the working.

Matplotlib
Unlike saeborn, matplotlib is a low level data visualization python library. Majority of its function lies in the submodule named pyplot

Scikit-Learn
It provides tools for classification, regression, clustering and dimensionality reduction through its interface in python.

Pickle
Used for serializing (pickling) and de-serializing (unpickling) python object structure so that the data can be easily transferred from system to system and then stored in a file.

NLTK
NLTK is a standard python library that provides a set of diverse algorithms for NLP.

BeautifulSoup
Beautiful Soup is a Python library for pulling data out of HTML and XML files.

Model/s Development and Evaluation

- Testing of Identified Approaches (Algorithms)

  In this study several machine learning models were tested and their results were compared before selecting the model that gave best result among all.

  Logistic Regression
  Using logistic approach for modelling, this supervised machine learning model aims to solve classification problems by predicting categorical outcomes, unlike linear regression that predicts a continuous outcome.

  KNeighbor Classifier
  The K-NN algorithm used in this classification stores all the available data and classifies a new data point based on the similarity. It is non parametric since it does not make any underlying assumption.

  Support Vector Classifier
  The SVC approach finds a hyperplane that creates a boundary between two classes of data to classify them.

  Multinomial Naive Bayes Classifier
  The algorithm first guesses the tag of a text using the Bayes theorem and then calculates each tag's likelihood for a given sample and lastly outputs the tag with the greatest chance.

  Ensemble Methods
  Ensemble technique uses several base estimators of machine learning model to predict the output. This results are then combined to improve the robustness of model over single estimator.

Randomforest Classifier
Using multiple decision trees as base, the model randomly performs the dataset sampling over the rows and features such that it form sample datasets for every model.

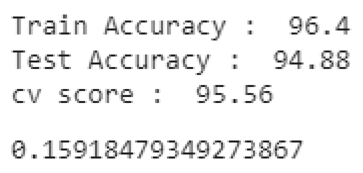
GradientBoost Classifier
The model after calculating the residual i.e. the difference between actual value and predicted target value; trains the weak model for mapping the features to that residual.

- Run and Evaluate models

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import r2_score,accuracy_score,classificatio
n_report,auc,roc_curve
from sklearn.model_selection import train_test_split,cross_val_sc
ore
from sklearn.metrics import log_loss


lg = LogisticRegression()

x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,
random_state=42)
lg.fit(X,y)
pred_train=lg.predict(x_train)
pred_test=lg.predict(x_test)
print("Train Accuracy : ",round(accuracy_score(y_train,pred_train
)*100,2))
print("Test Accuracy : ",round(accuracy_score(y_test,pred_test)*1
00,2))
print("cv score : ", round(cv_score.mean()*100,2))
logloss = log_loss(y_test, lg.predict_proba(x_test))
logloss
print("Classification Report:\n",classification_report(y_test,pre
d_test))
```

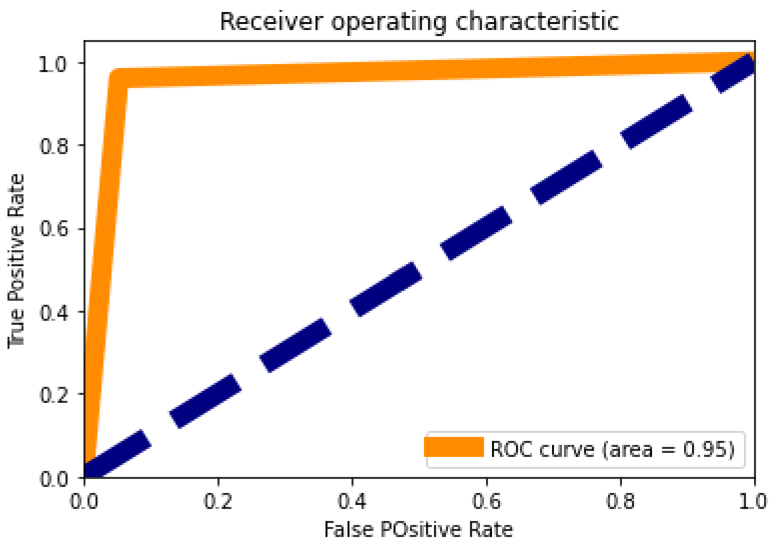
Output



```
Train Accuracy :  96.4
Test Accuracy :  94.88
cv score :  95.56

0.15918479349273867
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.99      0.97       942
           1       0.96      0.70      0.81       171

    accuracy                           0.95      1113
   macro avg       0.95      0.85      0.89      1113
weighted avg       0.95      0.95      0.95      1113
```

```
Confusion Matrix
 [[937   5]
 [ 52 119]]
```



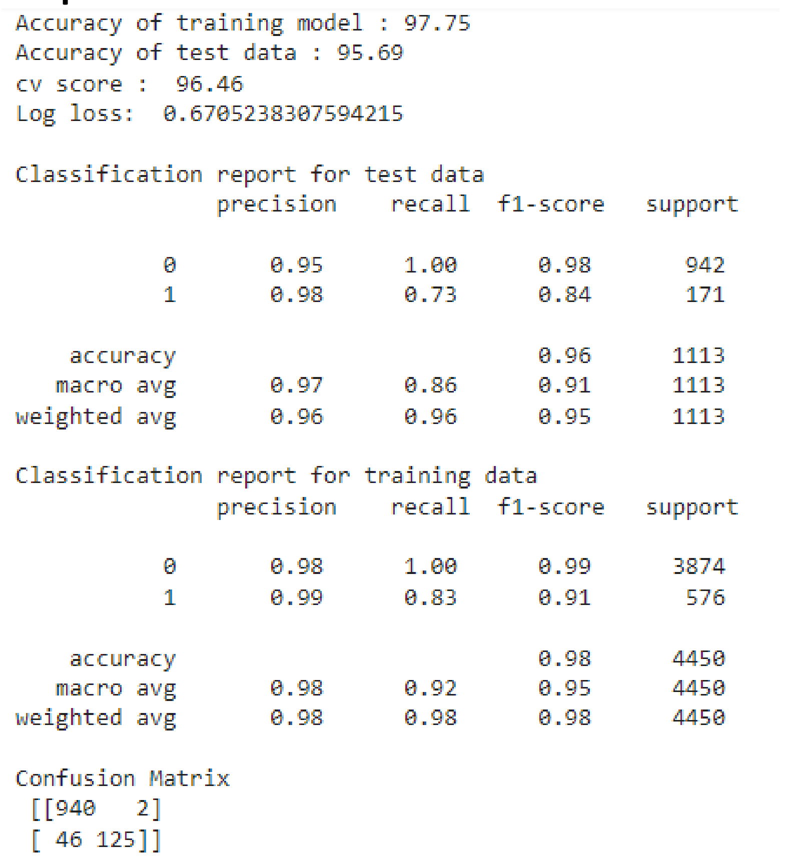
Model Selection User Defined Function

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve,auc

def model_selection(algorithm_instance,x_train,y_train,x_test,y_t
est):
algorithm_instance.fit(x_train,y_train)
model_pred_train=algorithm_instance.predict(x_train)
 model_pred_test=algorithm_instance.predict(x_test)
 print("Accuracy of training model :",round(accuracy_score(y_train
 ,model_pred_train)*100,2))
 print("Accuracy of test data :",round(accuracy_score(y_test,model
 _pred_test)*100,2))

 cv_score=cross_val_score(algorithm_instance,X,y,cv=5)
 print("cv score : ", round(cv_score.mean()*100,2))
print("\nClassification report for test data\n",classification_repo
 rt(y_test,model_pred_test))
print("Classification report for training data\n",classification_re
     port(y_train,model_pred_train))
    print("Confusion Matrix\n",confusion_matrix(y_test,model_pred_test))
    print("\n")

    fpr, tpr, thresholds = roc_curve(model_pred_test,y_test)
    roc_auc= auc(fpr,tpr)
    plt.figure()
    plt.plot(fpr,tpr,color='darkorange',lw=10,label='ROC curve (area =
%0.2f)' % roc_auc)
    plt.plot([0,1],[0,1],color='navy',lw=10,linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0,1.05])
    plt.xlabel("False POsitive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Receiver operating characteristic")
    plt.legend(loc='lower right')
    plt.show()
```

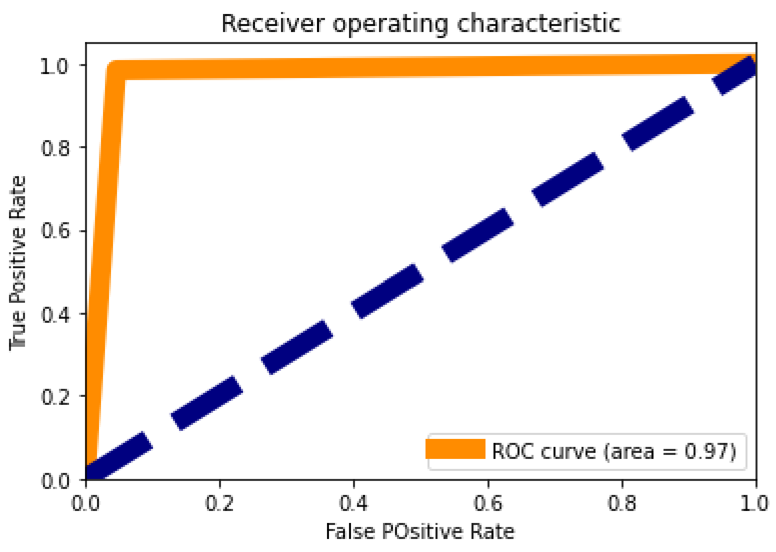
K Neighbour Classifier

```
from sklearn.neighbors import KNeighborsClassifier
k=KNeighborsClassifier()
model_selection(k,x_train,y_train,x_test,y_test)
```

Output



```
Accuracy of training model : 97.75
Accuracy of test data : 95.69
cv score :  96.46
Log loss:  0.6705238307594215

Classification report for test data
              precision    recall  f1-score   support

           0       0.95      1.00      0.98       942
           1       0.98      0.73      0.84       171

    accuracy                           0.96      1113
   macro avg       0.97      0.86      0.91      1113
weighted avg       0.96      0.96      0.95      1113

Classification report for training data
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      3874
           1       0.99      0.83      0.91       576

    accuracy                           0.98      4450
   macro avg       0.98      0.92      0.95      4450
weighted avg       0.98      0.98      0.98      4450

Confusion Matrix
 [[940   2]
 [ 46 125]]
```

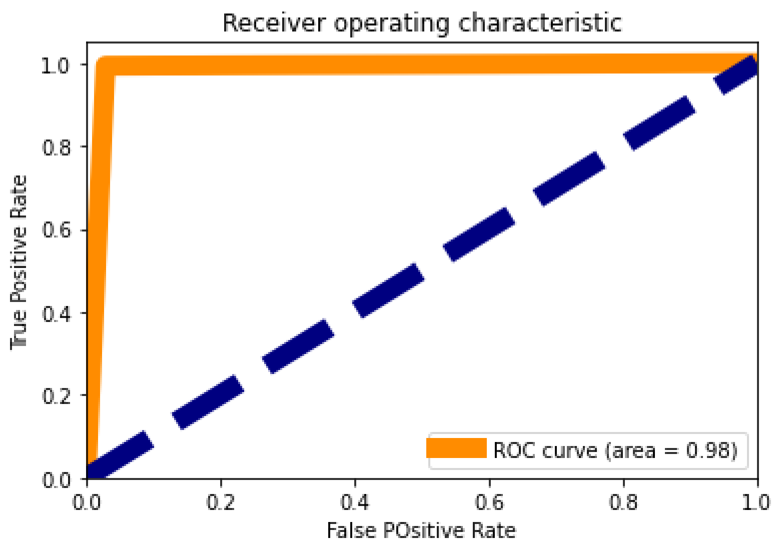


Support Vector Classifier

```
from sklearn import svm
s=svm.SVC()
model_selection(s,x_train,y_train,x_test,y_test)
```

Output

```
Accuracy of training model : 99.64
Accuracy of test data : 97.39
cv score :  97.03
Log loss:  0.0800859840132961

Classification report for test data
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       942
           1       0.99      0.84      0.91       171

    accuracy                           0.97      1113
   macro avg       0.98      0.92      0.95      1113
weighted avg       0.97      0.97      0.97      1113

Classification report for training data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3874
           1       1.00      0.97      0.99       576

    accuracy                           1.00      4450
   macro avg       1.00      0.99      0.99      4450
weighted avg       1.00      1.00      1.00      4450

Confusion Matrix
 [[941   1]
 [ 28 143]]
```



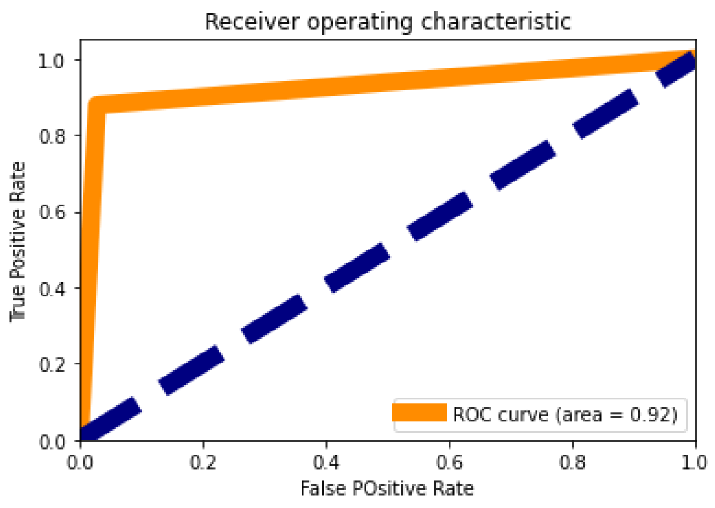
Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
model_selection(dtc,x_train,y_train,x_test,y_test)
```

Output

```
Accuracy of training model : 100.0
Accuracy of test data : 95.78
cv score :  95.63
Log loss:  1.4585107731902993

Classification report for test data
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       942
           1       0.88      0.84      0.86       171

    accuracy                           0.96      1113
   macro avg       0.92      0.91      0.92      1113
weighted avg       0.96      0.96      0.96      1113

Classification report for training data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3874
           1       1.00      1.00      1.00       576

    accuracy                           1.00      4450
   macro avg       1.00      1.00      1.00      4450
weighted avg       1.00      1.00      1.00      4450
```

```
Confusion Matrix
 [[922  20]
 [ 27 144]]
```



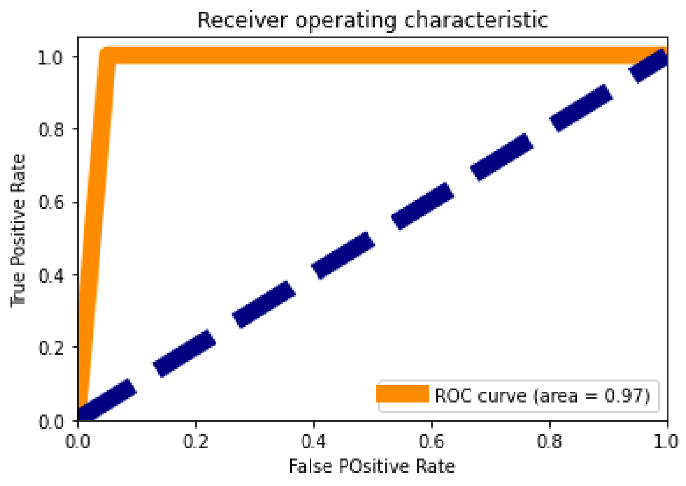
Multinomial Naïve Bayes Classifier

```
from sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB()
model_selection(mnb,x_train,y_train,x_test,y_test)
```

Output

```
Accuracy of training model : 97.19
Accuracy of test data : 95.42
cv score :  96.37
Log loss:  0.13285475903615468

Classification report for test data
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       942
           1       1.00      0.70      0.82       171

    accuracy                           0.95      1113
   macro avg       0.97      0.85      0.90      1113
weighted avg       0.96      0.95      0.95      1113
```

```
Classification report for training data
               precision    recall  f1-score   support

           0       0.97      1.00      0.98      3874
           1       1.00      0.78      0.88       576

    accuracy                           0.97      4450
   macro avg       0.98      0.89      0.93      4450
weighted avg       0.97      0.97      0.97      4450

Confusion Matrix
 [[942   0]
 [ 51 120]]
```



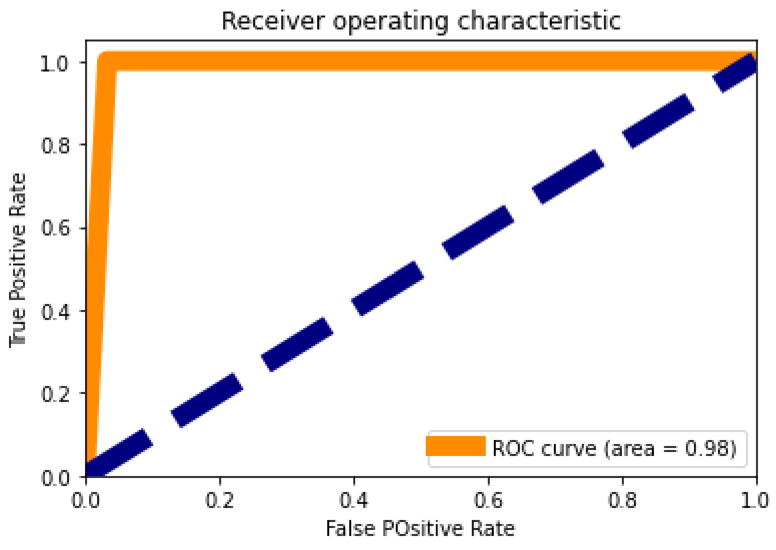
Randomforest Classifier

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

parameter={'criterion':['gini', 'entropy', 'log_loss'],
     'max_features' : [None,"sqrt","log2"],
      'class_weight':['balanced',' balanced_subsample'],
      'n_estimators':range(0,100,50)}
rf=RandomForestClassifier()
clf=GridSearchCV(rf,parameter)
clf.fit(x_train,y_train)

print(clf.best_params_)
```

```
{'class_weight': 'balanced', 'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators': 50}
```

```
from sklearn.ensemble import RandomForestClassifier

rf=RandomForestClassifier(n_estimators=50, class_weight= 'balanced
',criterion='entropy', max_features= None)
        model_selection(rf,x_train,y_train,x_test,y_test)
```

```
Accuracy of training model : 99.98
Accuracy of test data : 97.12
cv score :  97.16
Log loss:  0.20784666155597775

Classification report for test data
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       942
           1       1.00      0.81      0.90       171

    accuracy                           0.97      1113
   macro avg       0.98      0.91      0.94      1113
weighted avg       0.97      0.97      0.97      1113

Classification report for training data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3874
           1       1.00      1.00      1.00       576

    accuracy                           1.00      4450
   macro avg       1.00      1.00      1.00      4450
weighted avg       1.00      1.00      1.00      4450

Confusion Matrix
 [[942   0]
 [ 32 139]]
```



GradientBoost Regressor

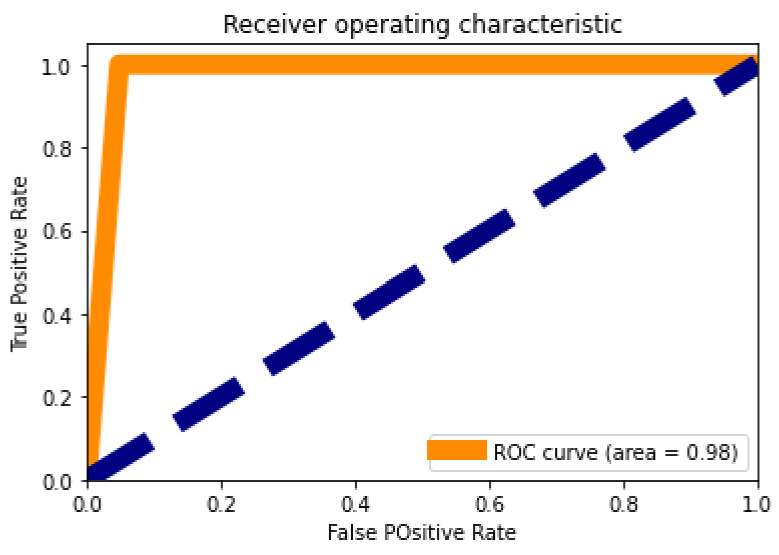
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier

parameter={'loss':['log_loss', 'deviance', 'exponential'],
        'criterion':['friedman_mse', 'squared_error', 'mse'],
        'max_features':['auto', 'sqrt', 'log2'],
        'n_estimators':range(0,100,50)}

    rf3=GradientBoostingClassifier()
    clf=GridSearchCV(rf3,parameter)
    clf.fit(x_train,y_train)

    print(clf.best_params_)
```

```
{'criterion': 'friedman_mse', 'loss': 'deviance', 'max_features': 'auto', 'n_estimators': 50}
```

```
rf3=GradientBoostingClassifier(criterion='friedman_mse', loss='deviance
', max_features= 'auto', n_estimators= 50)
model_selection(rf3,x_train,y_train,x_test,y_test)
```

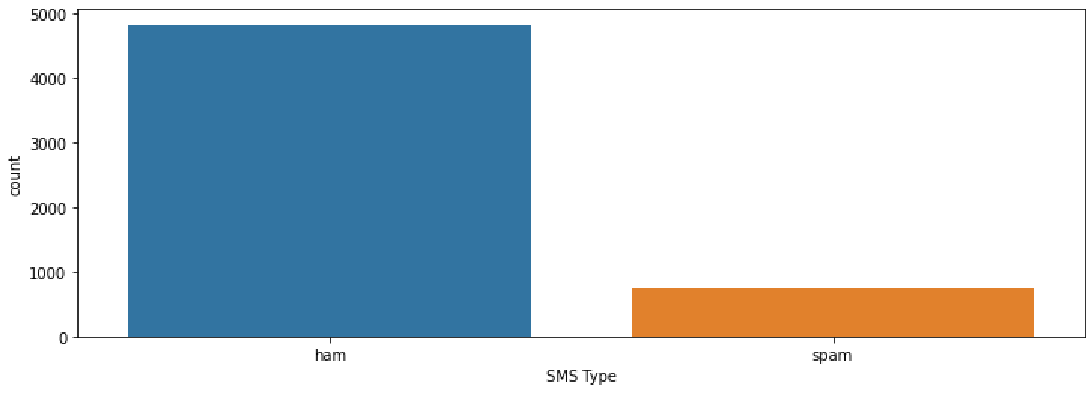
Output

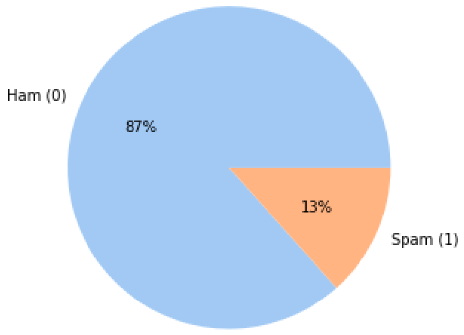
```
Accuracy of training model : 96.38
Accuracy of test data : 95.69
cv score :  94.88
Log loss:  0.1517099031830402

Classification report for test data
              precision    recall  f1-score   support

           0       0.95      1.00      0.98       942
           1       1.00      0.72      0.84       171

    accuracy                           0.96      1113
   macro avg       0.98      0.86      0.91      1113
weighted avg       0.96      0.96      0.95      1113

Classification report for training data
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      3874
           1       0.99      0.73      0.84       576

    accuracy                           0.96      4450
   macro avg       0.97      0.86      0.91      4450
weighted avg       0.96      0.96      0.96      4450

Confusion Matrix
 [[942   0]
 [ 48 123]]
```



- Dataset Visualizations

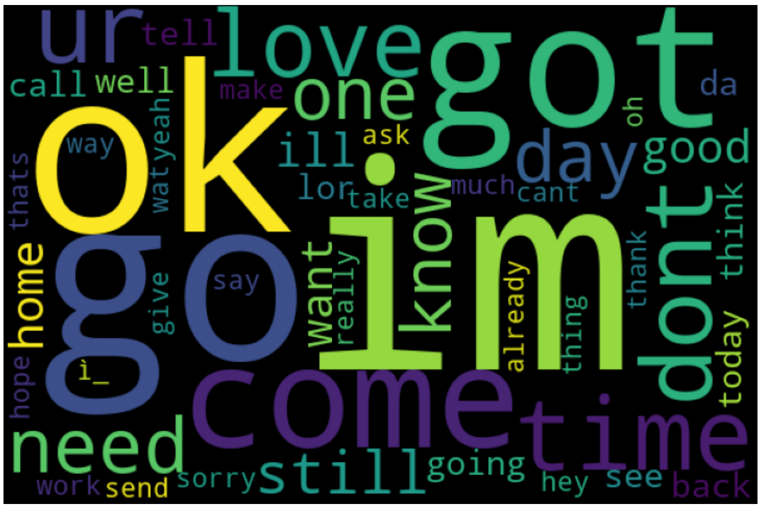
```
SMS Type
ham     4825
spam     747
Name: v1, dtype: int64
```



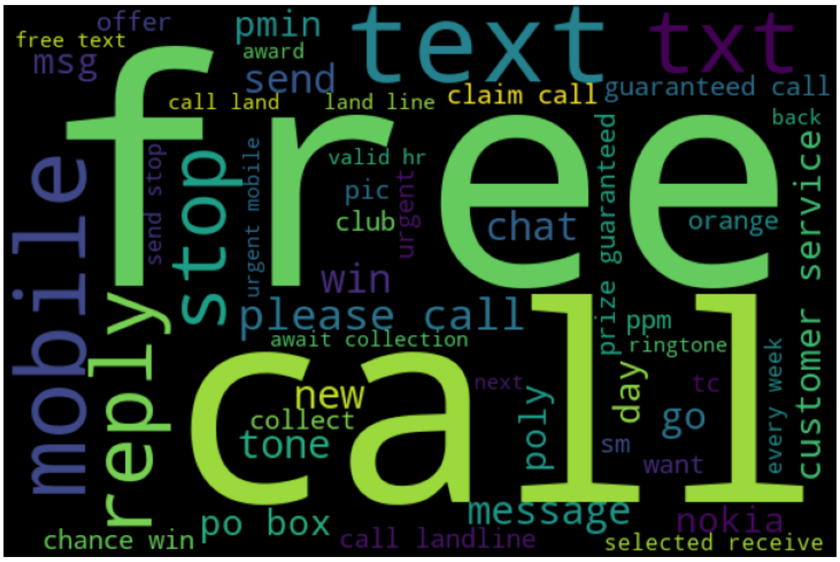


- Post Data cleaning Visualizations

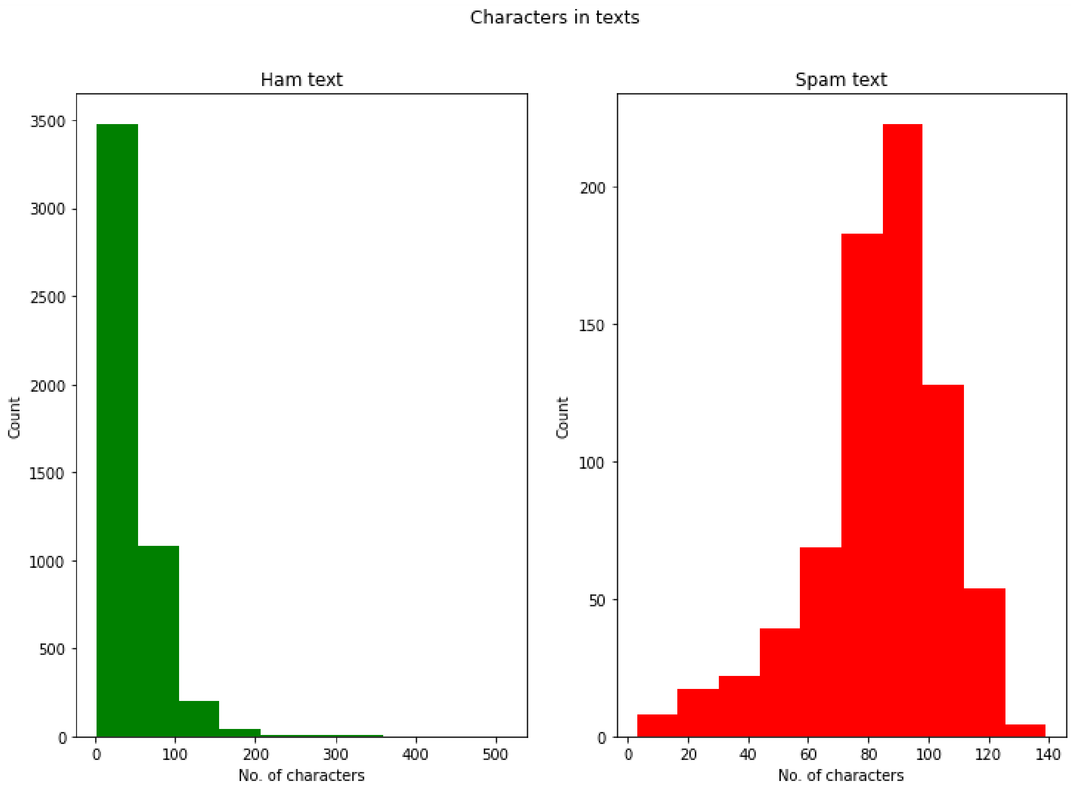
a) Word Cloud (Ham SMS)



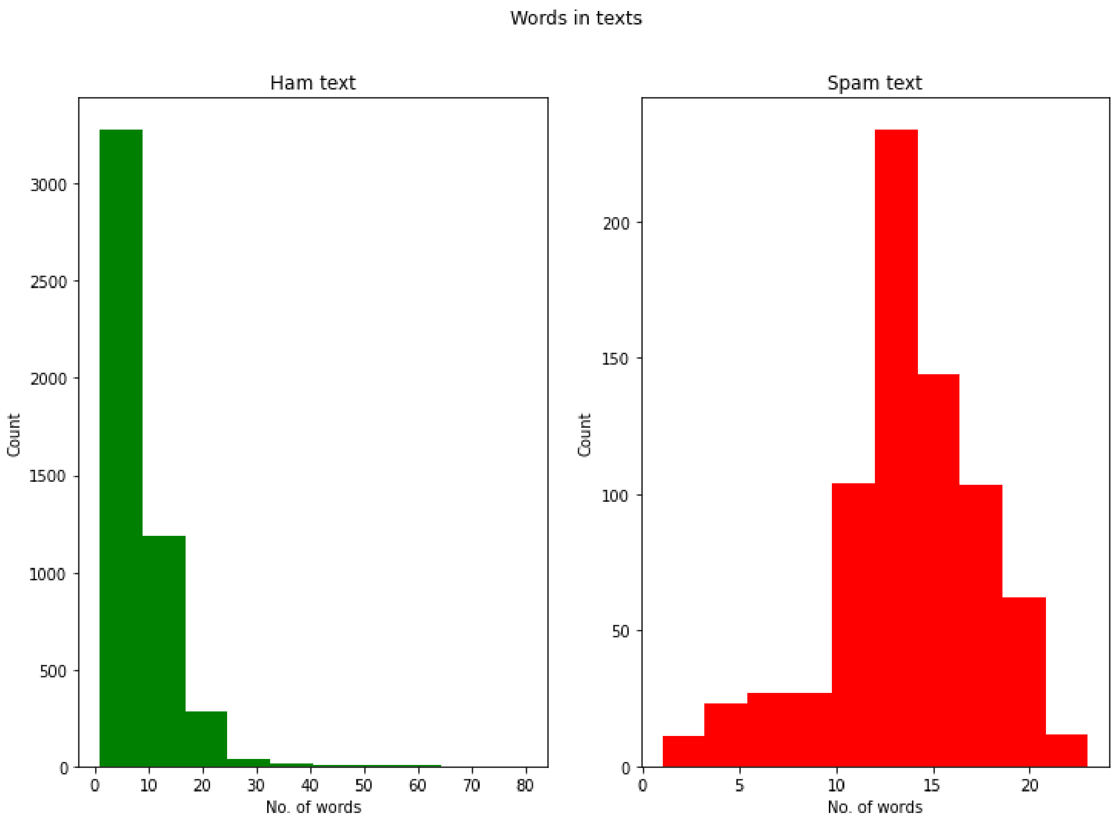
b) Word Cloud (SMS)



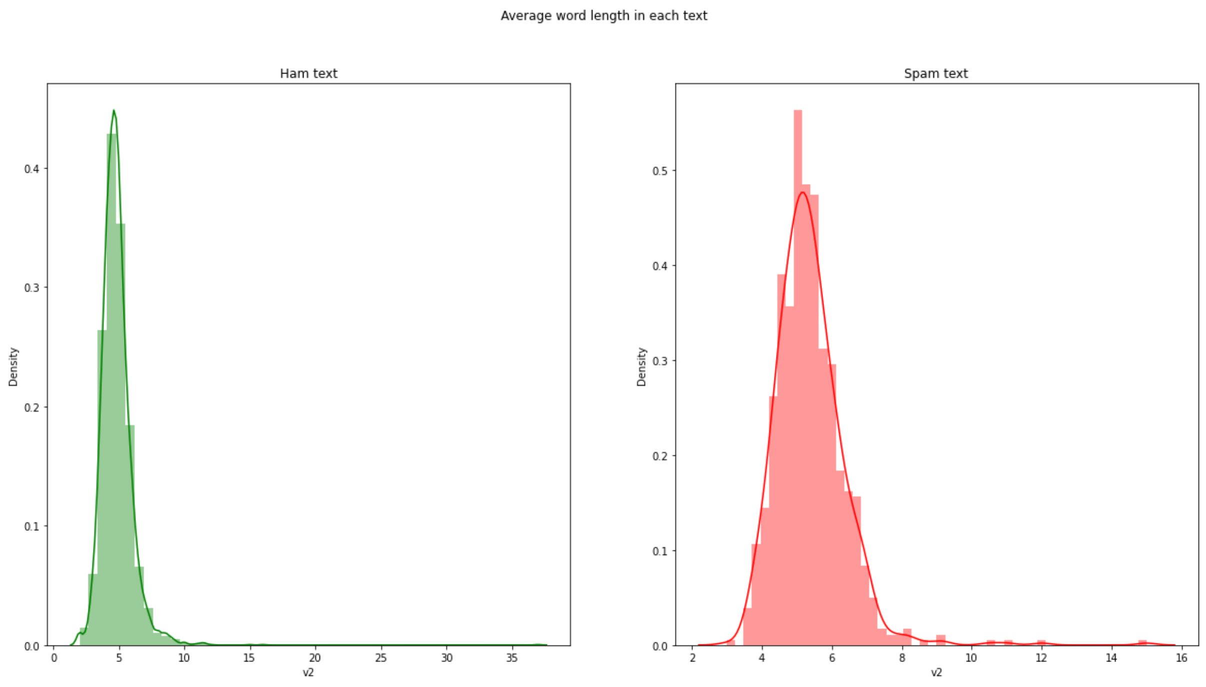
c) Character Count



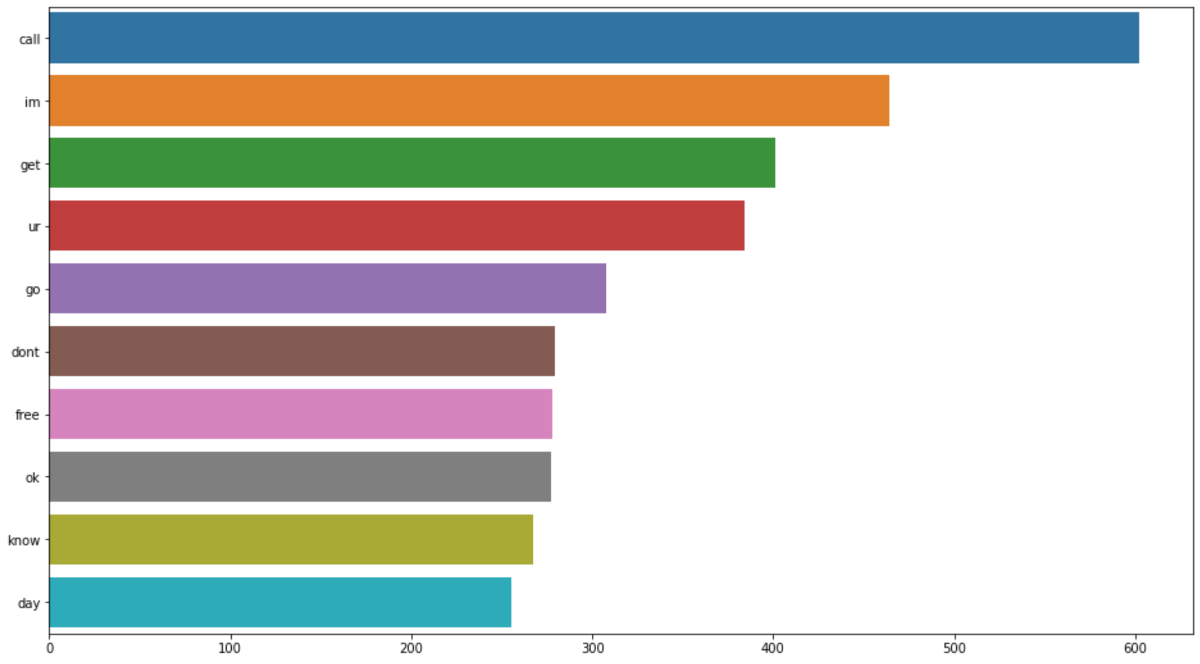
d) Words in texts



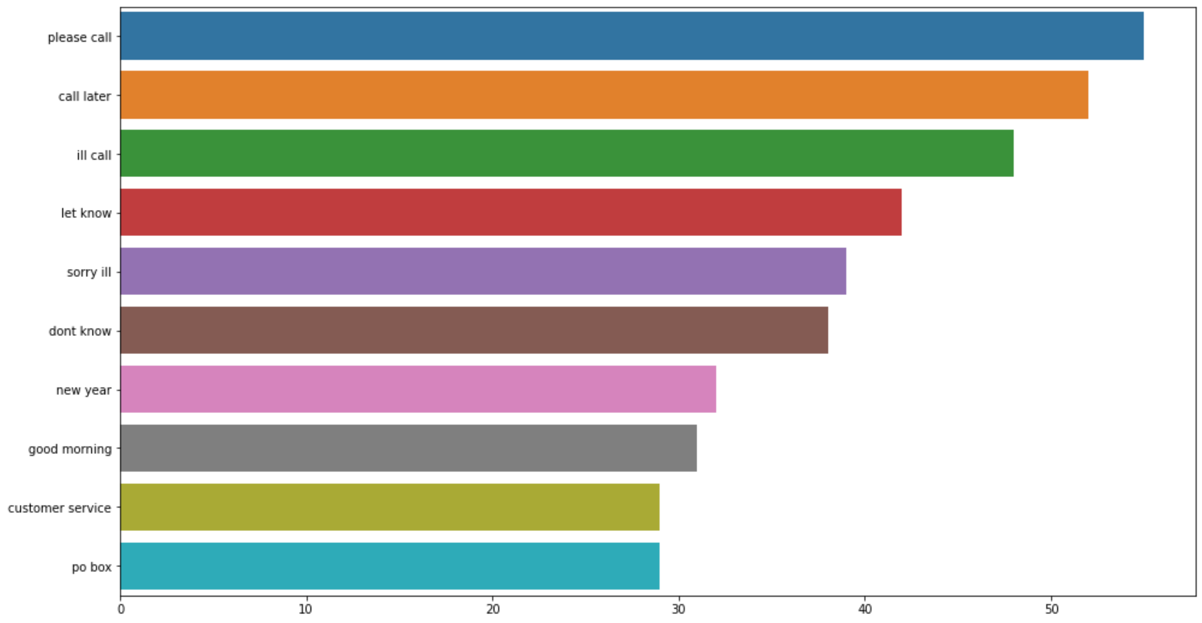
e) Average word length



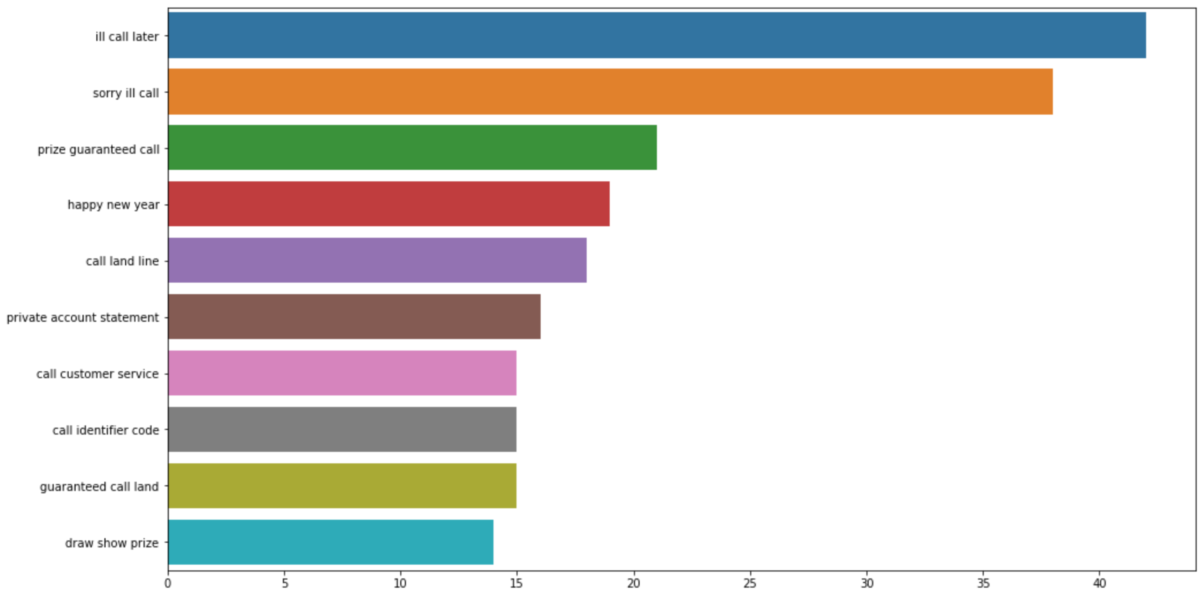
f) Unigram Analysis



g) Bigram Analysis



h) Trigram Analysis



- 13% of the SMS in the dataset are spam whereas the remaining 87% are ham SMS.
- 70 characters in text is most common in ham SMS category while approx 90 characters in text is most common in spam SMS category.

- 10 words in text is most common in ham SMS category
  while approx 15 words in text is most common in spam SMS category.
- The average word length distribution in ham SMS is marginally less than spam SMS.
- In unigram analysis, word 'call' is found to have highest number of repetition.
- In bigram analysis, 'please call' is found to have highest number of occurrence.
- In Trigram analysis, 'ill call later' is found to have highest number of occurrence.

CONCLUSION

- Key Findings and Conclusions of the Study

  - In this study we found that support vector classifier performs slightly better than rest of the algorithm tested.
  - Call is the most repeated word in entire dataset.
  - Average word count and character count in spam SMS is more than ham SMS

Thus we were successfully able to detect the spam SMS with high accuracy (97.39 %) and minimum error (log loss of 0.08).