FLIP ROBO

Spam Project

Submitted by:

Roshan Kumar Verma

ACKNOWLEDGMENT

This Project would not have been possible without the kind support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

I would like to thank Flip-Robo Technologies Bangalore for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents & my SME of Flip-Robo’s Mohd. Kashif for their kind co-operation and encouragement which help me in completion of this project.

My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

INTRODUCTION

Problem Statement:

The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged according being ham (legitimate) or spam.

Conceptual Background of the Domain Problem

Spam Detector is used to detect unwanted, malicious and virus infected texts and helps to separate them from the non-spam texts. It uses a binary type of classification containing the labels such as 'ham' (non-spam) and spam. Application of this can be seen in Google Mail (GMAIL) where it segregates the spam emails in order to prevent them from getting into the user's inbox.

The files contain one message per line. Each line is composed by two columns: v1 contains the label (ham or spam) and v2 contains the raw text. This corpus has been collected from free or free for research sources at the Internet:

Review of Literature

A collection of 5573 rows SMS spam messages was manually extracted from the Grumble text 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. The identification of the text of spam messages in the claims is a very hard and time-consuming task, and it involved carefully scanning hundreds of web pages.

Motivation for the Problem Undertaken

A subset of 3,375 SMS randomly chosen ham messages of the NUS SMS Corpus (NSC), which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore. The messages largely originate from Singaporeans and mostly from students attending the University. These messages were collected from volunteers who were made aware that their contributions were going to be made publicly available.

These information are Gathered from Different Sources:- Spam Email , become a big trouble over the internet. Spam is waste of time, storage space and communication bandwidth. The problem of spam e-mail has been increasing for years. In recent statistics, 40% of all emails are spam which about 15.4 billion email per day and that cost internet users about $355 million per year Knowledge engineering and machine learning are the two general approaches used in e-mail filtering In knowledge engineering approach a set of rules has to be specified according to which emails are categorized as spam or ham. Machine learning approach is more efficient than knowledge engineering approach; it does not require specifying any rules . Instead, a set of training samples, these samples is a set of pre classified e-mail messages. A specific algorithm is then used to learn the classification rules from these e-mail messages. Machine learning approach has been widely studied and there are lots of algorithms can be used in e-mail filtering. They include Naive Bayes, support vector machines, Neural Networks, K-nearest neighbour, Rough sets and the artificial immune system.

Analytical Problem Framing

Mathematical/ Analytical Modelling of the Problem

There are multiple mathematical and analytical analytics can be done before moving forward to the proper Exploratory Data Analysis.

Data contains 5572 entries each having 5 columns. Data contains Null values. We need to treat them using the domain knowledge and our own understanding. Extensive EDA has to be performed to gain relationships of important variable and price.

```
In [7]:  df.shape
Out[7]: (5572, 2)
```

```
In [8]:  df.columns
Out[8]: Index(['class_label', 'message'], dtype='object')
```

```
In [54]: # check for missing values
         df.isnull().sum()
Out[54]: class_label    0
         message        0
         length         0
         word_count     0
         dtype: int64
```

```
In [6]:  df.drop(columns=['Unnamed: 2','Unnamed: 3', 'Unnamed: 4'], inplace=True)
         df.rename(columns= {'v1':'class_label', 'v2':'message'}, inplace=True)
         df.head()
```

Out[6]:

	class_label	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...

We have to build Machine Learning models, apply regularization and determine the optimal values of Hyper Parameters. We need to find important features which affect the price positively or negatively.

Data Sources and their formats

A collection of 5573 rows SMS spam messages was manually extracted from the Grumble text 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. The identification of the text of spam messages in the claims is a very hard and time-consuming task, and it involved carefully scanning hundreds of web pages.

The data is provided in the CSV file .

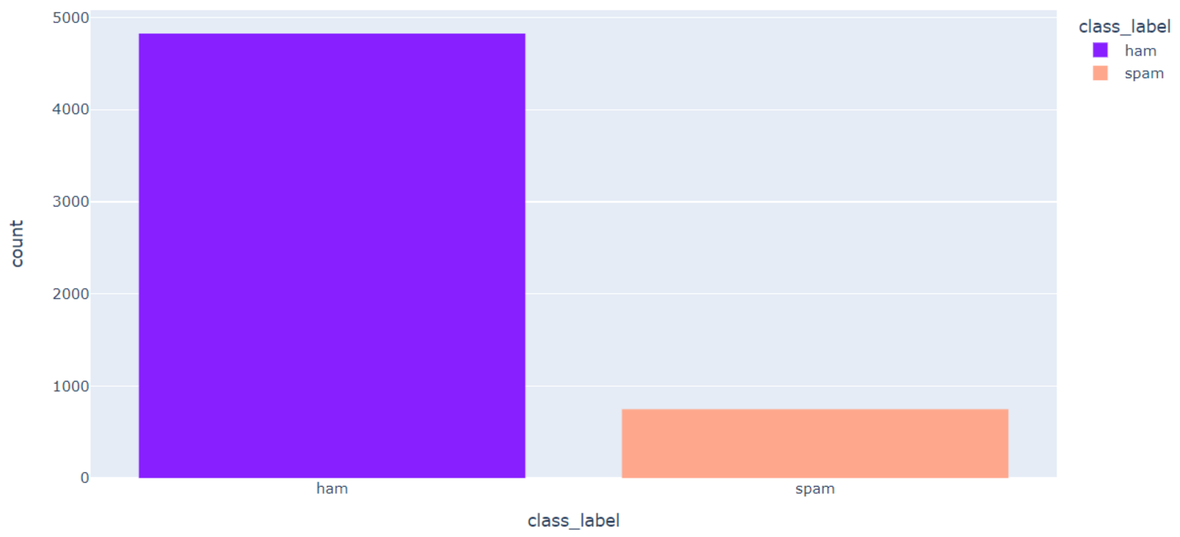
In [5]:
```
# read in the dataset and display the first few rows
df = pd.read_csv('spam.csv', encoding = 'latin-1')
df
```

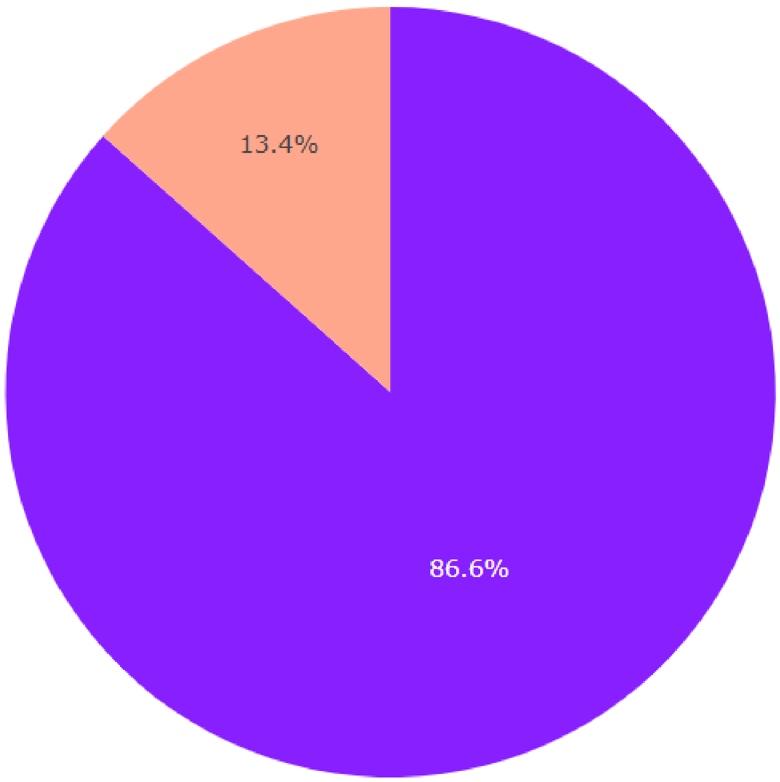
Out[5]:

	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

Univariate analysis





87%ham and 13%spam messages present in the dataset

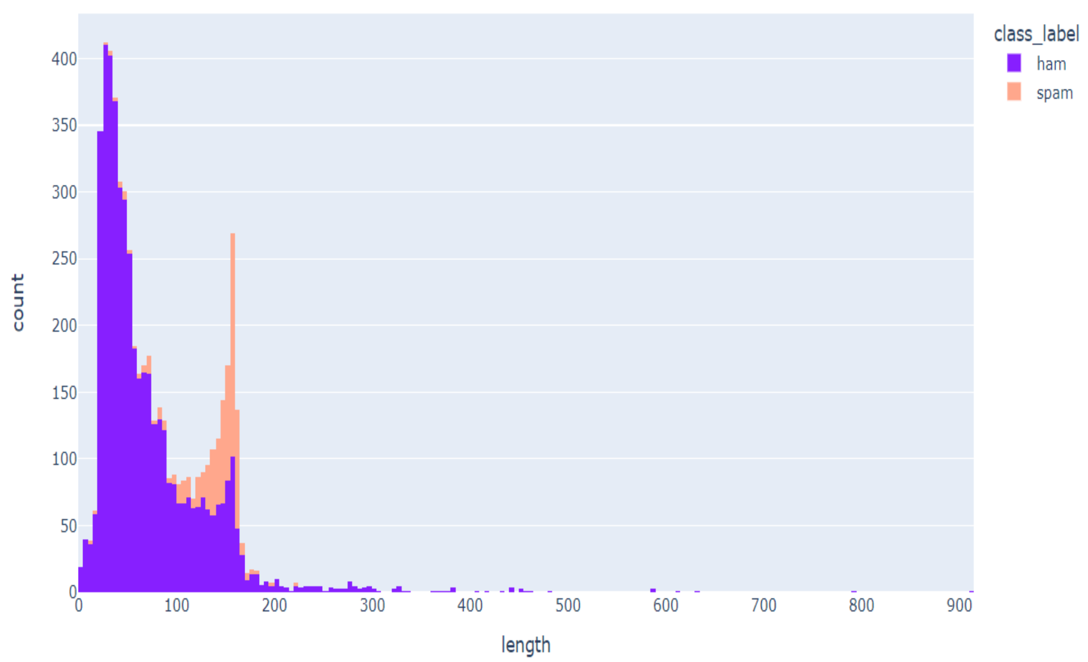
Analysing vs Length of the messages

In [12]:
```
df['length'] = df['message'].apply(len)
df.head()
```

Out[12]:

	class_label	message	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

In [17]:
```
fig = px.histogram(df, x="length", color="class_label", color_discrete_sequence=["#871fff","#ffa78c"] )
fig.show()
```



Spam messages found to lengthier than ham messages

Data Pre-Processing

Impute missing values

Dropping columns which has missing values.

In [6]:
```python
df.drop(columns=['Unnamed: 2','Unnamed: 3', 'Unnamed: 4'], inplace=True)
df.rename(columns= {'v1':'class_label', 'v2':'message'}, inplace=True)
df.head()
```

Out[6]:

	class_label	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...

Selecting categorical features and level encoding

Label encoding of input features

Data pre-processing

In [32]:
```python
df['class_label'] = df['class_label'].map( {'spam': 1, 'ham': 0})
```

```python
# Replace email address with 'emailaddress'
df['message'] = df['message'].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$', 'emailaddress')

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

# Replace money symbol with 'money-symbol'
df['message'] = df['message'].str.replace(r'£|\$', 'money-symbol')

# Replace 10 digit phone number with 'phone-number'
df['message'] = df['message'].str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$', 'phone-number')

# Replace normal number with 'number'
df['message'] = df['message'].str.replace(r'\d+(\.\d+)?', 'number')

# remove punctuation
df['message'] = df['message'].str.replace(r'[^\w\d\s]', ' ')

# remove whitespace between terms with single space
df['message'] = df['message'].str.replace(r'\s+', ' ')

# remove leading and trailing whitespace
df['message'] = df['message'].str.replace(r'^\s+|\s*?$', ' ')

# change words to lower case
df['message'] = df['message'].str.lower()

```

Pre-processing using NLP

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
df['message'] = df['message'].apply(lambda x: ' '.join(term for term in x.split() if term not in stop_words))
```

```
ss = nltk.SnowballStemmer("english")
df['message'] = df['message'].apply(lambda x: ' '.join(ss.stem(term) for term in x.split()))
```

```
sms_df = df['message']
from nltk.tokenize import word_tokenize

# creating a bag-of-words model
all_words = []
for sms in sms_df:
    words = word_tokenize(sms)
    for w in words:
        all_words.append(w)

all_words = nltk.FreqDist(all_words)
```

```
print('Number of words: {}'.format(len(all_words)))
```

```
Number of words: 6526
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_model = TfidfVectorizer()
tfidf_vec=tfidf_model.fit_transform(sms_df)
import pickle
#serializing our model to a file called model.pkl
pickle.dump(tfidf_model, open("tfidf_model.pkl","wb"))
tfidf_data=pd.DataFrame(tfidf_vec.toarray())
tfidf_data.head()
```

	0	1	2	3	4	5	6	7	8	9	...	6496	6497	6498	6499	6500	6501	6502	6503	6504	6505
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	0.0	0.0	0.0
1	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	0.0	0.0
2	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	0.0	0.0
3	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	0.0	0.0
4	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	0.0	0.0

5 rows × 6506 columns

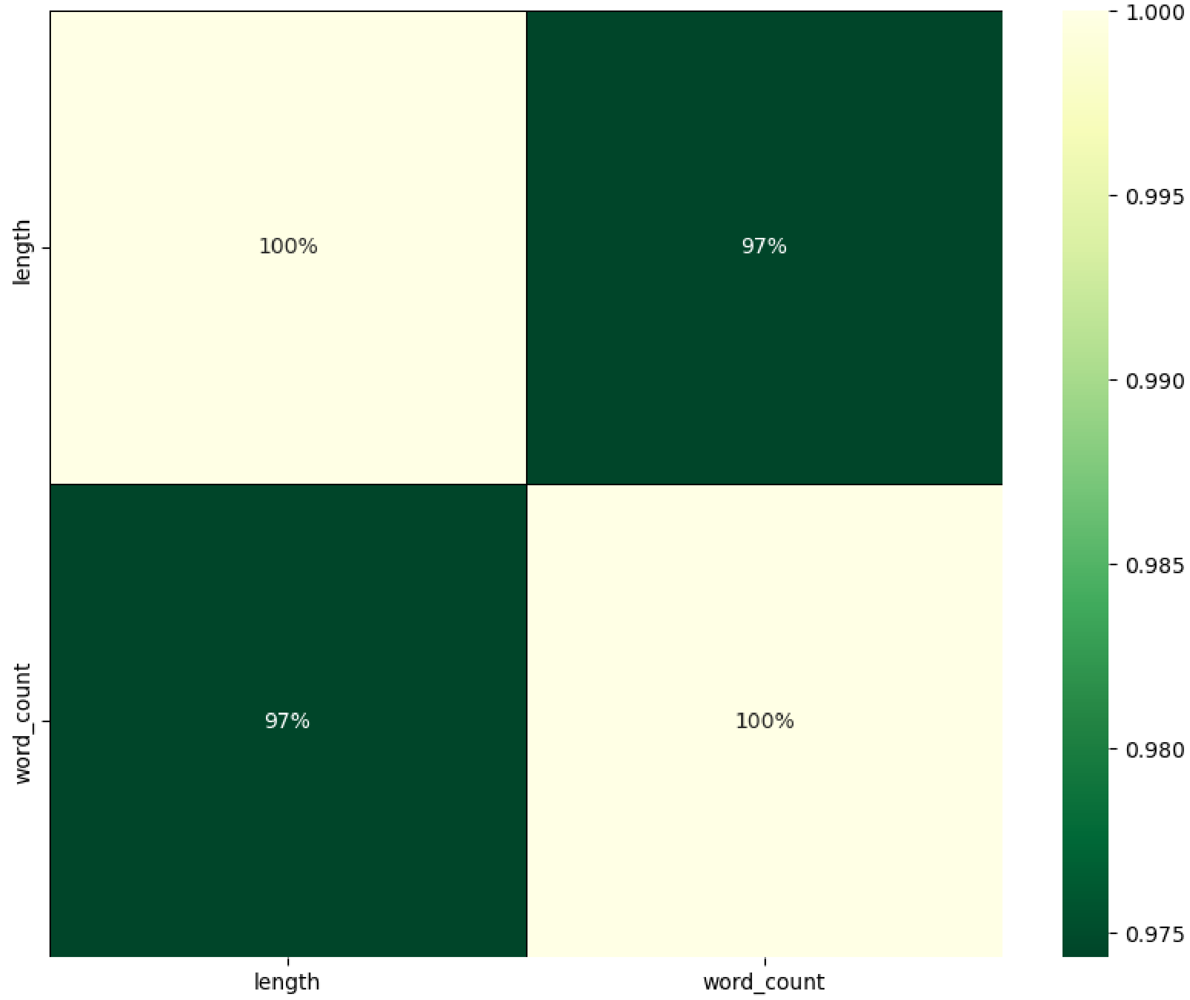
Data Inputs- Logic- Output Relationships

In [27]:
```
#Checking correlation of the dataset
corr=df.corr()  #corr() function provides the correlation value of each column
corr
```

Out[27]:

	length	word_count
length	1.000000	0.974318
word_count	0.974318	1.000000

In [28]:
```
#Plotting heatmap for visualizing the correlation
plt.figure(figsize=(10,8))
sns.heatmap(corr,linewidth=0.5,linecolor='black',fmt='.0%',cmap='YlGn_r',annot=True)
plt.show()
```

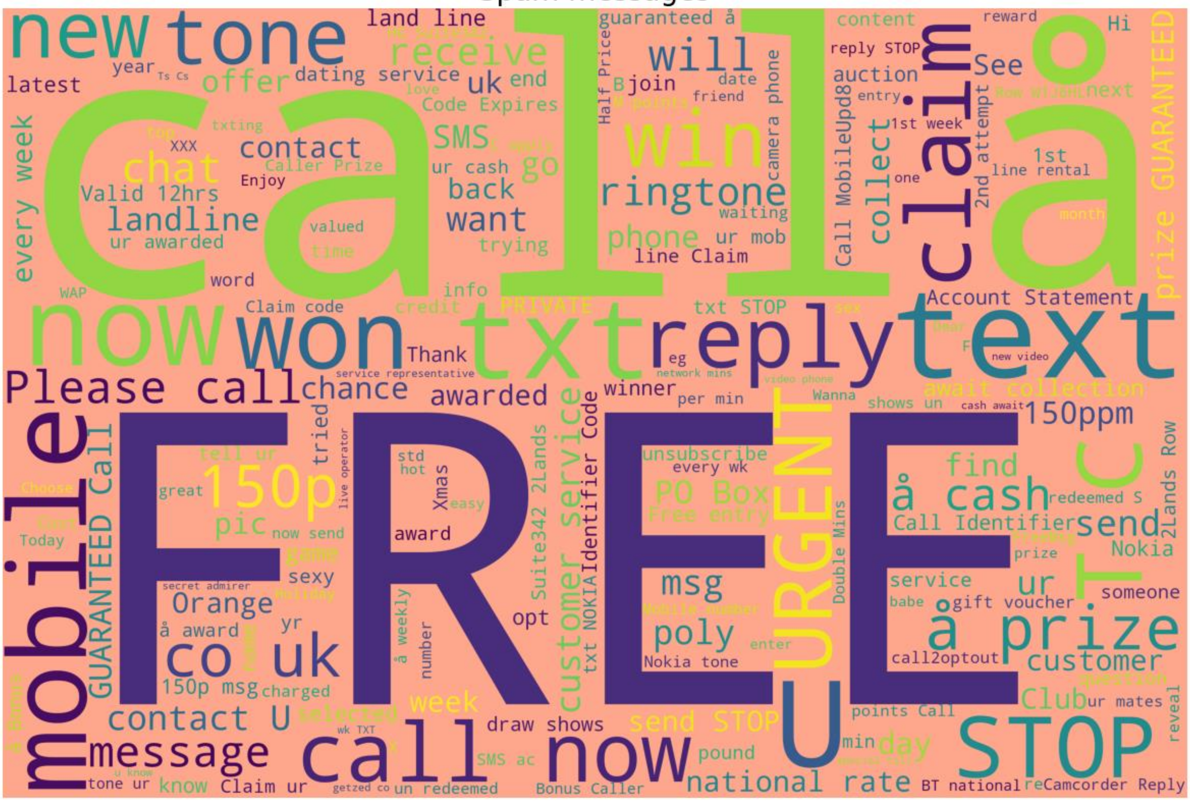


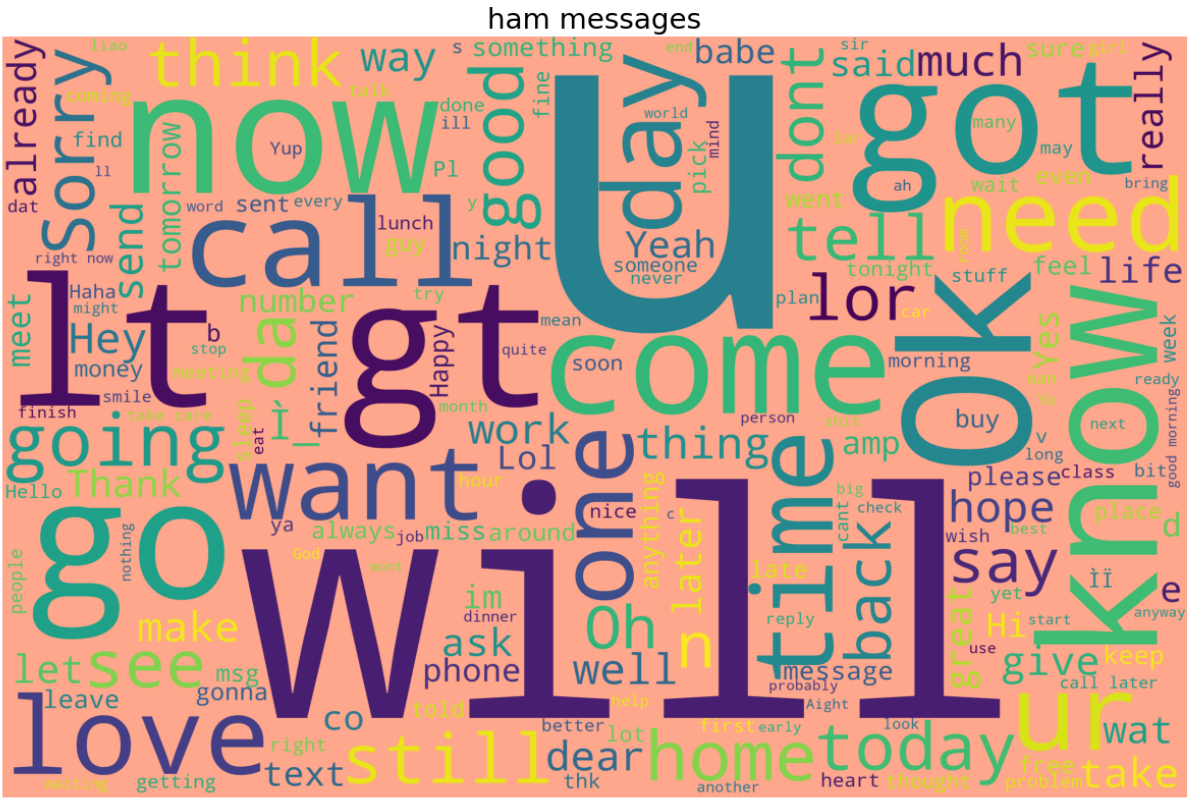
Displaying the Wordcloud

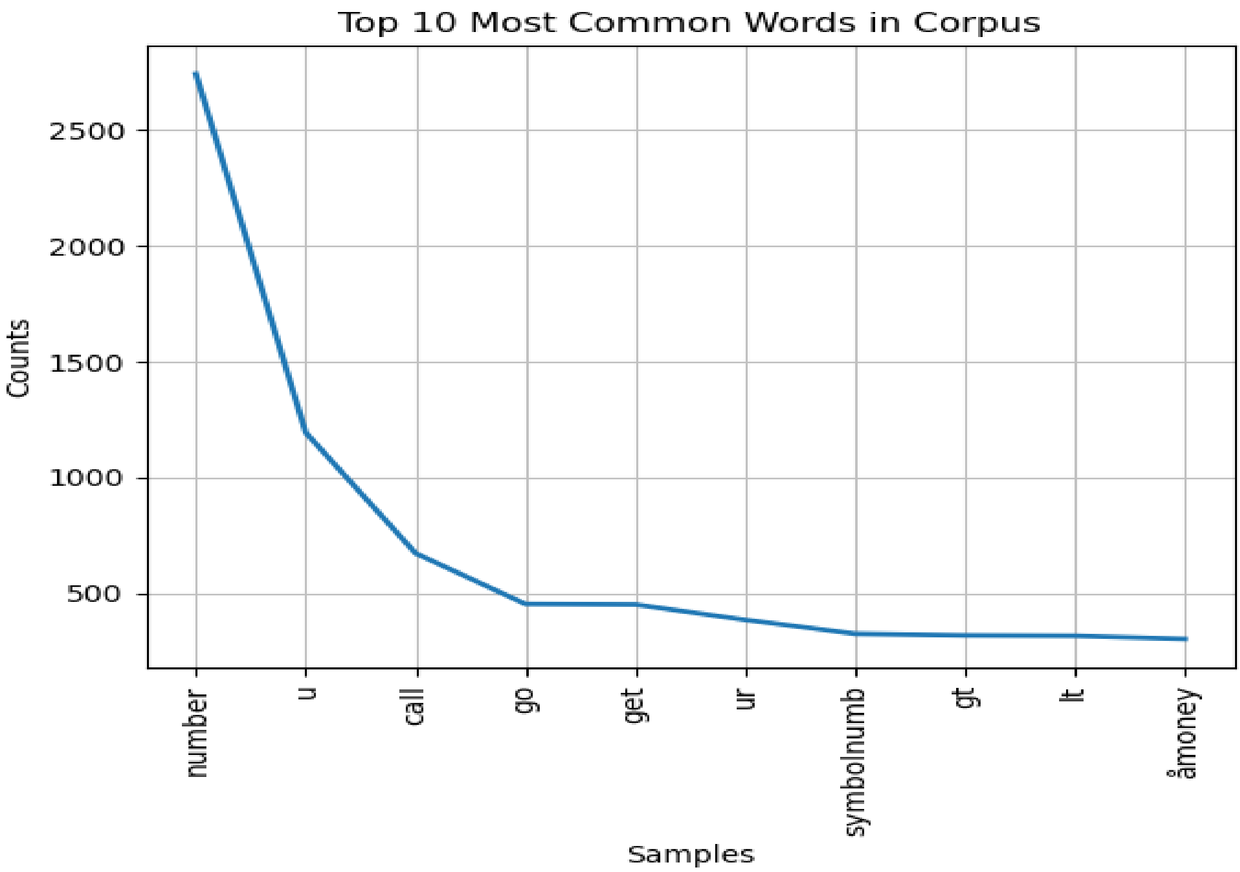
In [29]:

```python
import wordcloud
data_ham  = df[df['class_label'] == "ham"].copy()
data_spam = df[df['class_label'] == "spam"].copy()

def show_wordcloud(df, title):
    text = ' '.join(df['message'].astype(str).tolist())
    stopwords = set(wordcloud.STOPWORDS)
    fig_wordcloud = wordcloud.WordCloud(stopwords=stopwords, background_color="#ffa78c",
                                        width = 3000, height = 2000).generate(text)
    plt.figure(figsize=(15,15), frameon=True)
    plt.imshow(fig_wordcloud)
    plt.axis('off')
    plt.title(title, fontsize=20)
    plt.show()
```

Spam messages




ham messages
u
will
now
got
gt
lt
call
go
come
ok
know
love
want
one
time
day
good
think
need
ur
tell
still
home
today
going
Sorry
see
say
back
lor
later
thing
dont
much
said
really
Yeah
Oh
well
Hey
make
hope
take
dear
work
night
way
babe
phone
text
ask
give
wat


Top 10 Most Common Words in Corpus
Counts
2500
2000
1500
1000
500
Samples
number
u
call
go
get
ur
symbolnumb
gt
lt
åmoney

Hardware and Software Requirements and Tools Used

The General Hardware used for this project is :-

8 GB RAM

512GB SSD

Intel i5 processor

So for doing these project the hardware use is a laptop with high end specification, an internet connection. While coming to software I have used anaconda navigator in that I have used **Jupyter notebook** to do my python programming and analysis, for csv file excel is needed.

So in Jupyter notebook I have used lots of python libraries to carry out this project I will be pasting down below with proper justification.

1. Pandas- pandas is used to read the data, visualization and analysis of data.

2. Numpy - used for working with array and various mathematical techniques.

3. Seaborn - I used seaborn for plotting different types of plot.

4. Ploty - Is also used to plot the different types of plot.

5. Matplotlib - It provides an object-oriented API for embedding plots into applications

6. zscore - To remove outliers.

7. skew- to treat skewed data using various transformation like sqrt, log, boxcox.

8. PCA- I used this to remove the data columns to 10.

9. Standard-Scaler- I used this data to scale my data before sending it to model.

10. train-test-split - to split the test and train data.

11. joblib - this is used to save the model pickle file.

Model/s Development and Evaluation

Testing of Identified Approaches (Algorithms)

1) from sklearn.neighbors import KNeighborsClassifier
2) from sklearn.linear_model import LogisticRegression
3) from sklearn.tree import DecisionTreeClassifier
4) from sklearn.naive_bayes import GaussianNB
5) from sklearn.ensemble import RandomForestClassifier
6) from sklearn.preprocessing import StandardScaler
7) from sklearn.metrics import
8) classification_report , confusion_matrix,accuracy_score,roc_curve,auc

Run and Evaluate selected models

In [58]:

```
models = []
models.append(('KNeighborsClassifier', KNN))
models.append(('LogisticRegression', LR))
models.append(('DecisionTreeClassifier',DT))
models.append(('GaussianNB', GNB))
models.append(('RandomForestClassifier', RF))
```

In [59]:

```
# lets make the for loop and call the algorithm one by one and save data to respective model using append function.
Model=[]
score=[]
cvs=[]
rocscore=[]
for name,model in models:
    print('******************************',name,'***************************')
    print('\n')
    Model.append(name)
    model.fit(x_train,y_train)
    print(model)
    pre=model.predict(x_test)
    print('\n')
    AS=accuracy_score(y_test,pre)
    print('accuracy_score=',AS)
    score.append(AS*100)
    print('\n')
    sc=cross_val_score(model,x,y,cv=10,scoring='accuracy').mean()
    print('cross_val_score',sc)
    cvs.append(sc*100)
    print('\n')
    false_positive_rate,true_positive_rate,thresholds=roc_curve(y_test,pre)
    roc_auc= auc(false_positive_rate,true_positive_rate)
    print('roc_auc_score = ',roc_auc)
    rocscore.append(roc_auc*100)
```

Key Metrics for success in solving problem under consideration

We can observe that I imported the metrics to find the accuracy score, roc_auc_curve, confusion_matrix, classification_report, in order to interpret the models output. Then I also selected the model to find the cross_validation_score and cross validation prediction.

```
LogisticRegression()


accuracy_score:  0.9805680119581465


cross_val_score:  0.9667939987820408


roc_auc_score:  0.951479583796922


Hamming_loss: 0.01943198804185351


Log_loss : 0.6711630659506583


Classification report:

              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1157
           1       0.94      0.91      0.93       181

    accuracy                           0.98      1338
   macro avg       0.96      0.95      0.96      1338
weighted avg       0.98      0.98      0.98      1338



Confusion matrix:

[[1147   10]
 [  16  165]]
```

```
MultinomialNB()

accuracy_score:  0.976831091180867

cross_val_score:  0.9688131942242556

roc_auc_score:  0.9702913326043253

Hamming_loss: 0.023168908819133034

Log_loss : 0.8002401035729126

Classification report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      1157
           1       0.88      0.96      0.92       181

    accuracy                           0.98      1338
   macro avg       0.94      0.97      0.95      1338
weighted avg       0.98      0.98      0.98      1338


Confusion matrix:

[[1133   24]
 [   7  174]]
```

```
DecisionTreeClassifier()


accuracy_score:  0.9723467862481315


cross_val_score:  0.9719524593216672


roc_auc_score:  0.9234231222871114


Hamming_loss: 0.027653213751868

46
Log_loss : 0.9551147400474056


Classification report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      1157
           1       0.93      0.86      0.89       181

    accuracy                           0.97      1338
   macro avg       0.96      0.92      0.94      1338
weighted avg       0.97      0.97      0.97      1338


Confusion matrix:

[[1146   11]
 [  26  155]]
```

```
KNeighborsClassifier()

accuracy_score:  0.9446935724962631

cross_val_score:  0.9062131026256587

roc_auc_score:  0.8002406681405998

Hamming_loss: 0.05530642750373692

Log_loss : 1.9102175279658082

Classification report:

              precision    recall  f1-score   support

           0       0.94      1.00      0.97      1157
           1       0.98      0.60      0.75       181

    accuracy                           0.94      1338
   macro avg       0.96      0.80      0.86      1338
weighted avg       0.95      0.94      0.94      1338


Confusion matrix:

[[1155    2]
 [  72  109]]
```

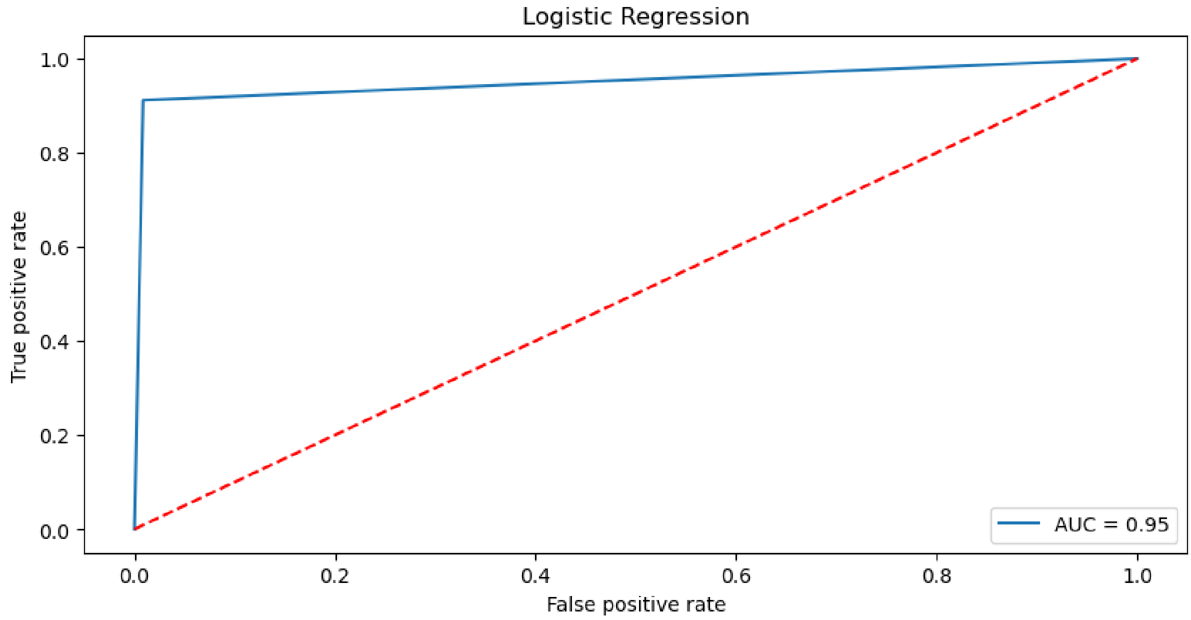
```
RandomForestClassifier()

accuracy_score:  0.9805680119581465

cross_val_score:  0.9820509529777093

roc_auc_score:  0.9305070744017916

Hamming_loss: 0.01943198804185351

Log_loss : 0.6711576874926078

Classification report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1157
           1       0.99      0.86      0.92       181

    accuracy                           0.98      1338
   macro avg       0.99      0.93      0.96      1338
weighted avg       0.98      0.98      0.98      1338


Confusion matrix:

[[1156    1]
 [  25  156]]
```

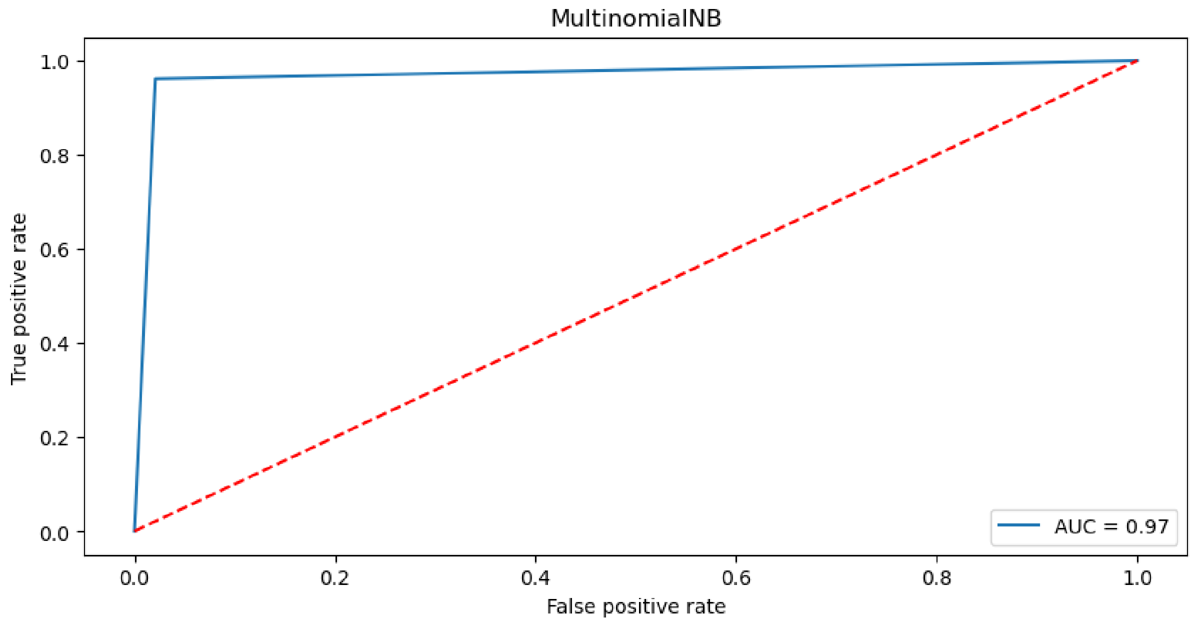
```
AdaBoostClassifier()

accuracy_score:  0.9701046337817638

cross_val_score:  0.975543929579804

roc_auc_score:  0.936108338864562

Hamming_loss: 0.029895366218236172

Log_loss : 1.0325613211846283

Classification report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1157
           1       0.89      0.89      0.89       181

    accuracy                           0.97      1338
   macro avg       0.94      0.94      0.94      1338
weighted avg       0.97      0.97      0.97      1338


Confusion matrix:
[[1137   20]
 [  20  161]]
```

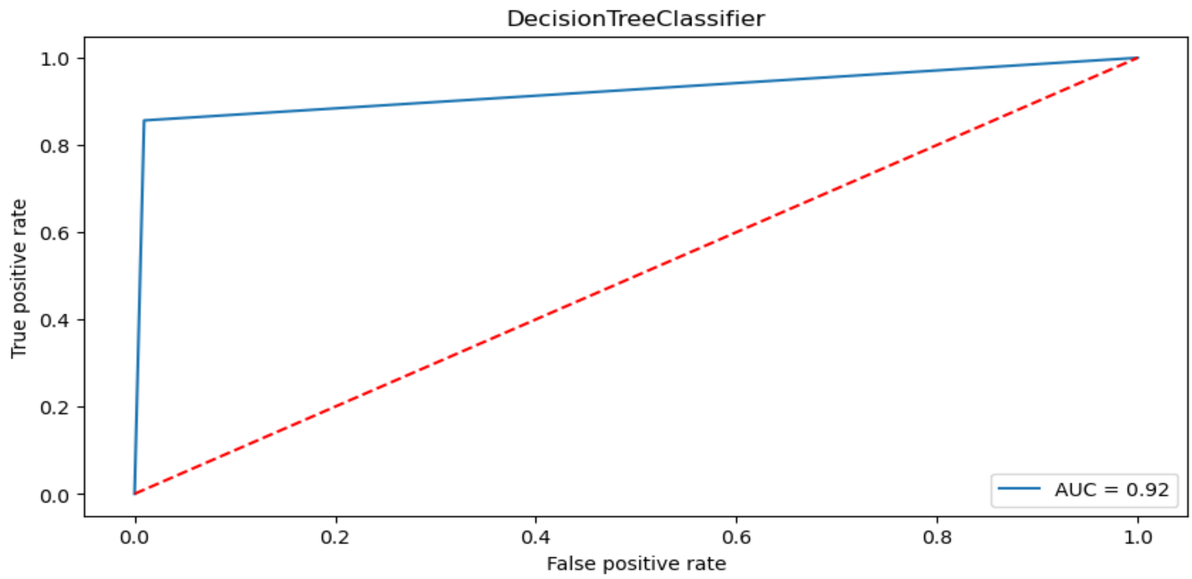
```
GradientBoostingClassifier()

accuracy_score:  0.9723467862481315

cross_val_score:  0.975991856784084

roc_auc_score:  0.9513864681472851

Hamming_loss: 0.02765321375186846

Log_loss : 0.9551219113248065

Classification report:

              precision    recall  f1-score   support

           0       0.99      0.98      0.98      1157
           1       0.88      0.92      0.90       181

    accuracy                           0.97      1338
   macro avg       0.93      0.95      0.94      1338
weighted avg       0.97      0.97      0.97      1338


Confusion matrix:

[[1134   23]
 [  14  167]]
```

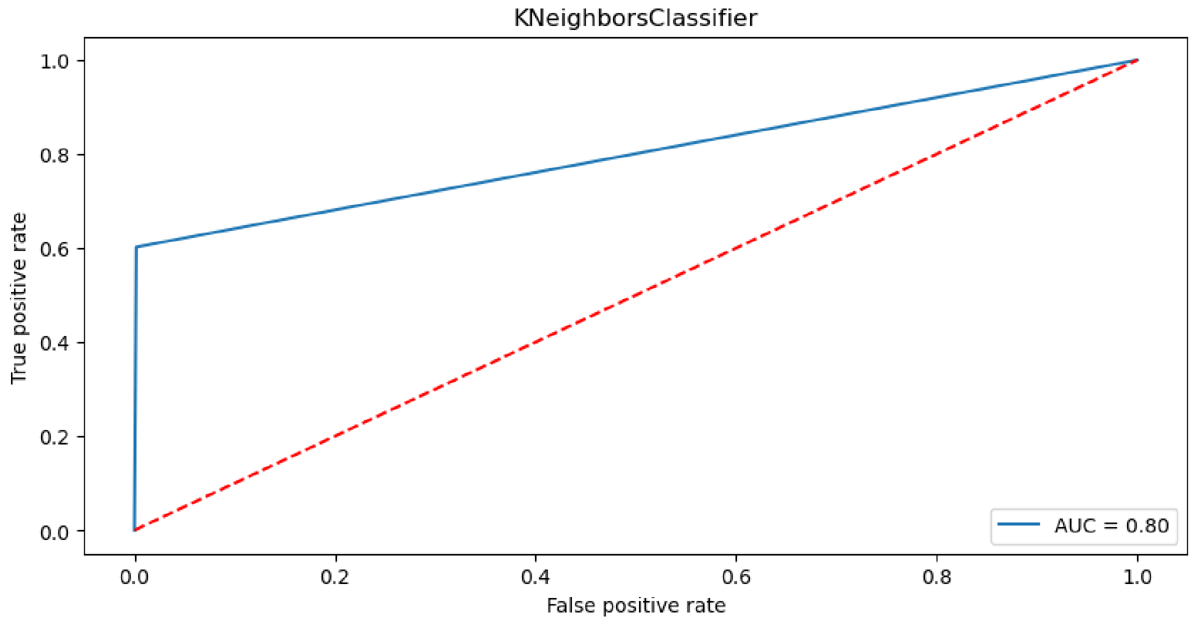
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, ...)


accuracy_score:  0.9798206278026906


cross_val_score:  0.9793565953506164


roc_auc_score:  0.9533777105010577


Hamming_loss: 0.020179372197309416


Log_loss : 0.6969779953899492


Classification report:

              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1157
           1       0.93      0.92      0.92       181

    accuracy                           0.98      1338
   macro avg       0.96      0.95      0.96      1338
weighted avg       0.98      0.98      0.98      1338



Confusion matrix:

[[1145   12]
 [  15  166]]
```

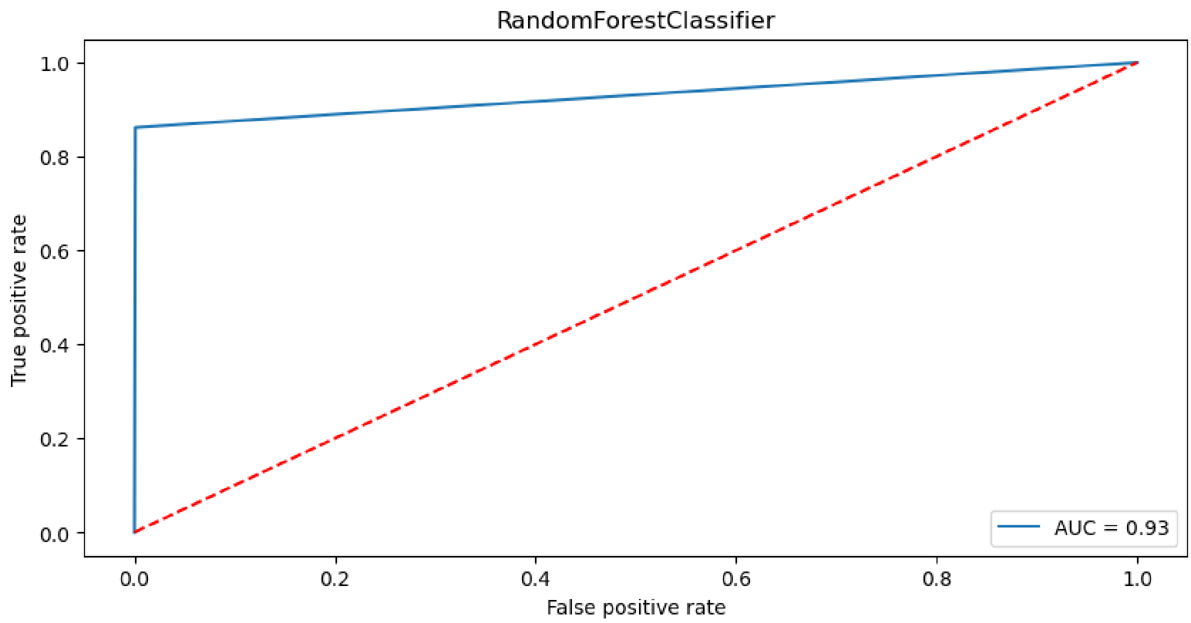
Visualizations

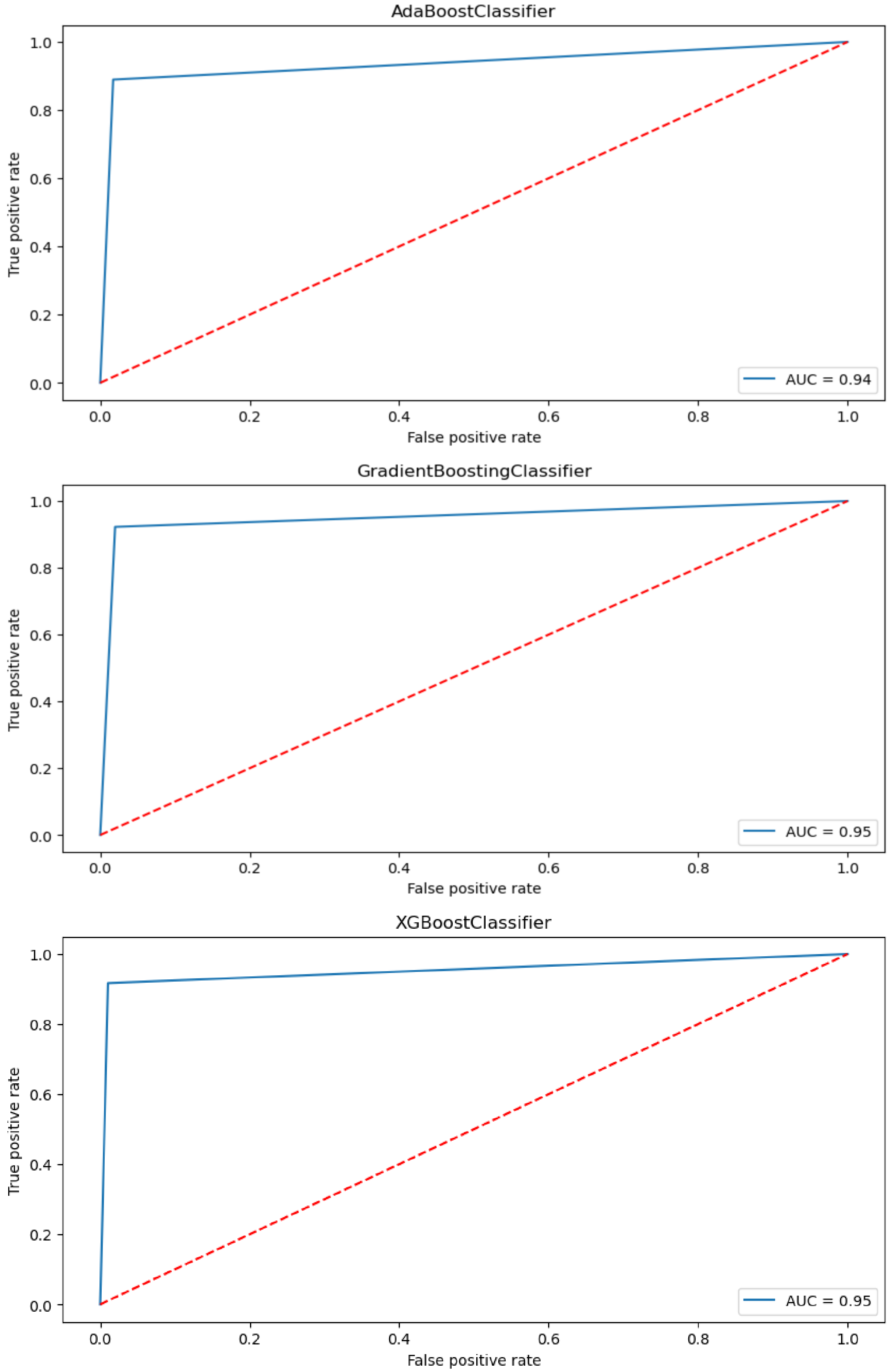





DecisionTreeClassifier
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.92


KNeighborsClassifier
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.80


RandomForestClassifier
True positive rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.93


AdaBoostClassifier
True positive rate
False positive rate
AUC = 0.94
GradientBoostingClassifier
True positive rate
False positive rate
AUC = 0.95
XGBoostClassifier
True positive rate
False positive rate
AUC = 0.95
0.0
0.2
0.4
0.6
0.8
1.0

Interpretation of the Results

```
#Finalizing the result
result=pd.DataFrame({'Model':Model, 'Accuracy_score': score,'Cross_val_score':cvs,'roc_auc_score':rocscore,
                    'Hamming_loss':h_loss, 'Log_loss':l_loss})
result
```

	Model	Accuracy_score	Cross_val_score	roc_auc_score	Hamming_loss	Log_loss
0	Logistic Regression	98.056801	96.679400	95.147958	0.019432	0.671163
1	MultinomialNB	97.683109	96.881319	97.029133	0.023169	0.800240
2	DecisionTreeClassifier	97.234679	97.195246	92.342312	0.027653	0.955115
3	KNeighborsClassifier	94.469357	90.621310	80.024067	0.055306	1.910218
4	RandomForestClassifier	98.056801	98.205095	93.050707	0.019432	0.671158
5	AdaBoostClassifier	97.010463	97.554393	93.610834	0.029895	1.032561
6	GradientBoostingClassifier	97.234679	97.599186	95.138647	0.027653	0.955122
7	XGBoostClassifier	97.982063	97.935660	95.337771	0.020179	0.696978

Logistic Regression and Random Forest Classifier showed the best accuracy (98.25%). After running the for loop of classification algorithms and the required metrics, we can see that the best 2 performing algorithms are RandomForestClassifier because the loss values are less and their scores are the best among all. Now, we will try Hyperparameter Tuning to find out the best parameters and using them to improve the scores and metrics values.

Hyper-parameter Tuning

Random Forest Classifier

In [51]:
```
#Creating parameter list to pass in GridSearchCV
parameters={'min_samples_leaf': [1, 2, 4], 'min_samples_split': [2, 5, 10],'n_estimators': [50, 100, 500]}
```

In [52]:
```
rfc = RandomForestClassifier()
rfc.fit(x_train_os,y_train_os)
```

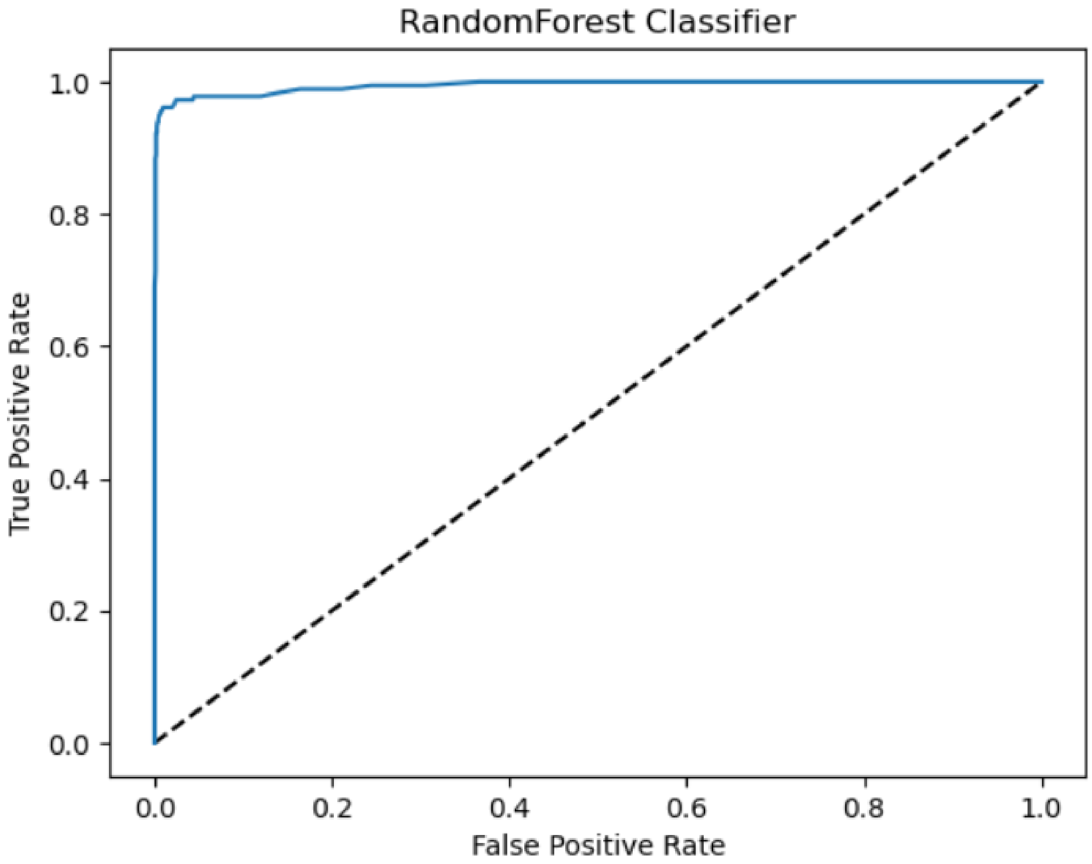
Out[52]: RandomForestClassifier()

In [53]:

```
pred=rfc.predict(x_test)
print('Accuracy score: ',accuracy_score(y_test,pre)*100)
print('Cross validation score: ',cross_val_score(rfc,X,y,cv=5,scoring='accuracy').mean()*100)
false_positive_rate,true_positive_rate,thresholds=roc_curve(y_test,pre)
roc_auc= auc(false_positive_rate,true_positive_rate)
print('roc_auc_score: ',roc_auc)
hloss = hamming_loss(y_test, pre)
print("Hamming_loss:", hloss)
loss = log_loss(y_test, pre)
print("Log loss:", loss)
print('Classification report: \n')
print(classification_report(y_test,pre))
print('Confusion matrix: \n')
print(confusion_matrix(y_test,pre))
```

```
Accuracy score:  97.98206278026906
Cross validation score:  98.16022708399392
roc_auc_score:  0.9533777105010577
Hamming_loss: 0.020179372197309416
Log loss: 0.6969779953899492
Classification report:

              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1157
           1       0.93      0.92      0.92       181

    accuracy                           0.98      1338
   macro avg       0.96      0.95      0.96      1338
weighted avg       0.98      0.98      0.98      1338

Confusion matrix:

[[1145   12]
 [  15  166]]
```

```python
#AUC_ROC Curve of Randomforest Classifier with oversampled data
y_pred_prob=rfc.predict_proba(x_test)[:,1]
fpr,tpr,thresholds=roc_curve(y_test,y_pred_prob)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr,label='RandomForest Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('RandomForest Classifier')
plt.show()

auc_score=roc_auc_score(y_test,rfc.predict(x_test))
print(auc_score)
```




```
0.9415567981586977
```

Conclusion

We converted all the text to lower case and made a bag of words. Counted the frequency of each word and chose 3,000 most frequent word Make a feature set that is simply a sequence of True, False based on whether the data set contains word in "frequent word" set or not.

Logistic Regression and RandomForestClassifier showed the best accuracy (98.25%).

Things to work on in the future

Do some more extensive analysis on the text.

Get rid of all the punctuation and stop words to see the difference in the result.