FLIP ROBO

Email Spam Detection Project

Submitted by:

Shama Tanweer

INTRODUCTION

- **Business Problem Framing**

  A start up company wants a system which can identify if the mail is a spam or not. This is called Spam Detection, and it is a binary classification problem. The reason to do this is that identification of spam helps us to ensure data security.

Mathematical/ Analytical Modeling of the Problem

We'll be using the **messages.csv** file provided by the Fliprobo technologies

- The dataset consists of 2893 rows & 3 columns ,
- The 'label ' attribute is the target column showing whether the mail is spam(1) or not spam(0).
- We observe that the dataset is imbalanced as only 481 mails(17%) is spam ,while rest 2412(83%) are regular mails

• Identification of possible problem-solving approaches (methods)
Used the following process for problem solving

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

Data pre-processing:

Data usually comes from a variety of sources & is often inconsistent,inaccurate.
Datapreprocessing helps to enhace the quality of data & make it ready for ML model.

steps involved In pre-processing are:

- Cleaning the raw data-It involves deletion of words or special character that do not add meaning to the text.
- **Important cleaning steps are:**

- Lowering case
- Handling of special characters
- Removal of stopwords
- Handling of hyperlinks
- Handling of numbers
- REPLACING one or 2 character length word BY WHITE SPACE
- Remove leading and trailing whitespace
- Replace URLs with 'webaddress'

➢ Converting all the messages to lower case:

All text in given data was converted to small case since keeping words in large case does not make sense as same word with small and large case conveys same meaning.

```
# Convert all messages to lower case
emails['subject'] = emails['subject'].str.lower()
emails['message'] = emails['message'].str.lower()
```

➢ Performing regex operation

By using regex we removed unwanted words, characters, numbers etc.

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

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

    # Replace money symbols with 'moneysymb'
    emails[i] = emails[i].str.replace(r'£|\$', 'dollers')

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


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

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

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

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

   #REPLACING SPECIAL CHARACTERS  BY WHITE SPACE
    emails[i]=emails[i].str.replace(r"[^a-zA-Z0-9]+", " ")

     #REPLACING one or 2  character length word   BY WHITE SPACE
    emails[i]=emails[i].str.replace(r'\b[a-zA-Z]{1,2}\b', '')
```

➢ Removing stopwords.

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

stop_words = set(stopwords.words('english') + ['u', 'ü', 'ur', '4', '2', 'im', 'don
t', 'doin', 'ure'])

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

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

Adding additional attribute :

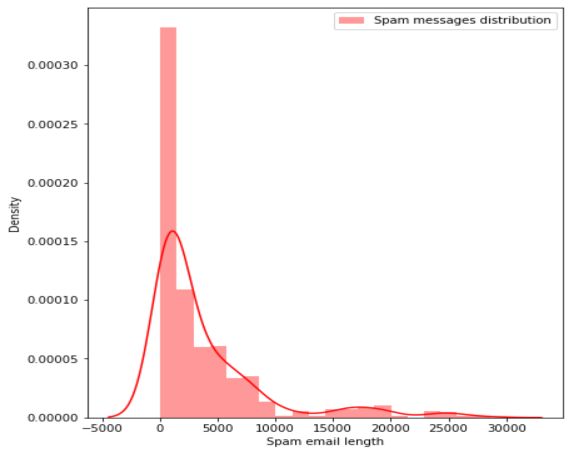
To compare the length of subject & message before preprocessing and after preprocessing an addition column 'subject_Cleaned_length' & message_clean_length wasadded.

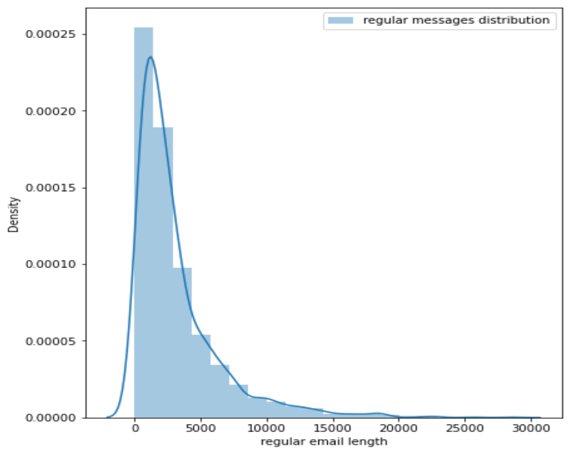
```
# New column (clean_length) after puncuations,stopwords removal
emails['subject_clean_length'] = emails.subject.str.len()
emails['message_clean_length'] = emails.message.str.len()
emails.head()
# Total subject length removal
print ('Origian subject Length', emails.length_subject.sum())
print ('Clean subject Length', emails.subject_clean_length.sum())
Origian subject Length 91725
Clean subject Length 77452

# Total message length removal
print ('Origian message Length', emails.length_message.sum())
print ('Clean message Length', emails.message_clean_length.sum())
Origian message Length 9344743
Clean message Length 6498264
```

After executing all these steps it was found that all the words & special characters were removed from the dataset which were of no use and consuming memory.

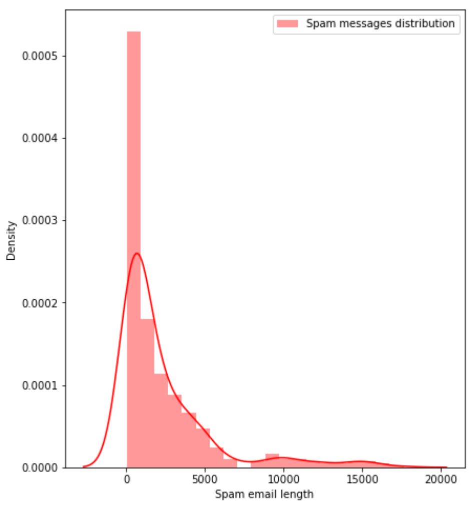
- **Then we have plotted graph to show distribution of world count before cleaning for distribution between spam emails and regular emails message**

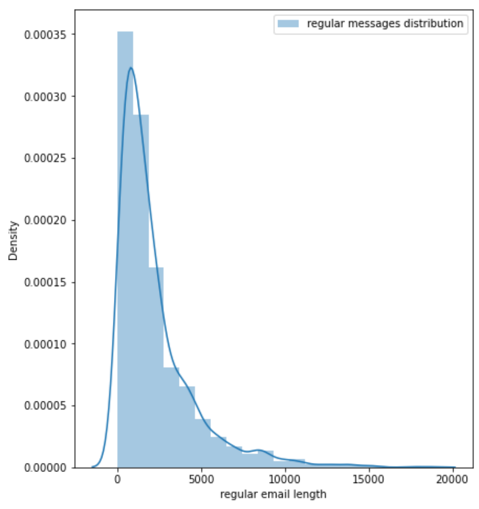
  - Message distribution before cleaning





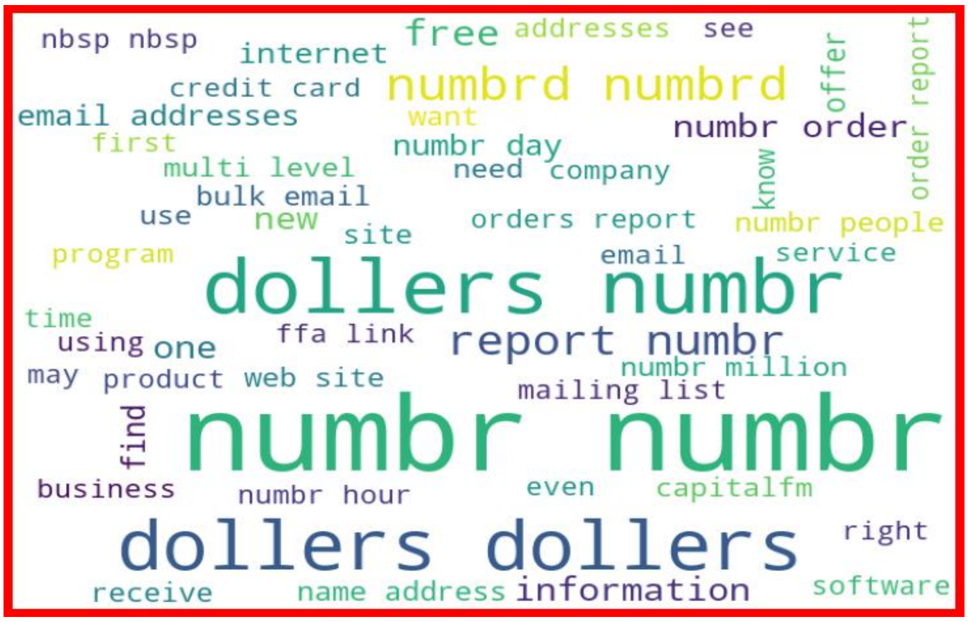
  - Message distribution after cleaning



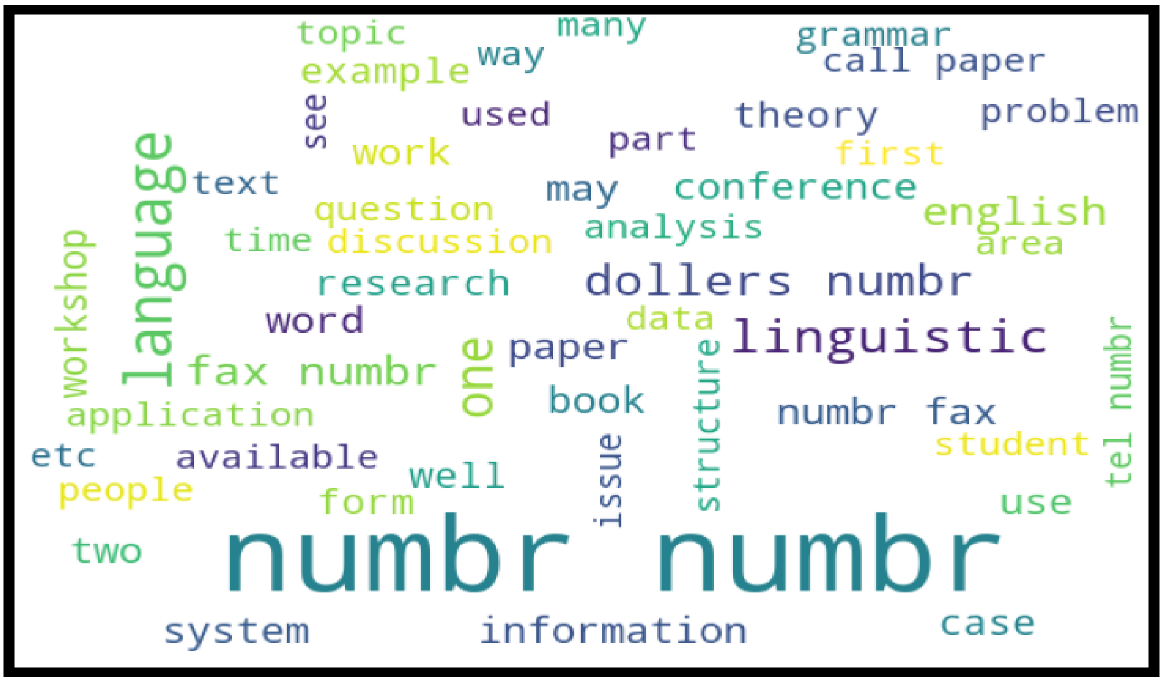


- **To get better view of words contained in spam mails . A word dictionary (wordcloud) was made showing the first 50 words highly occurred words in SPAM mails.**

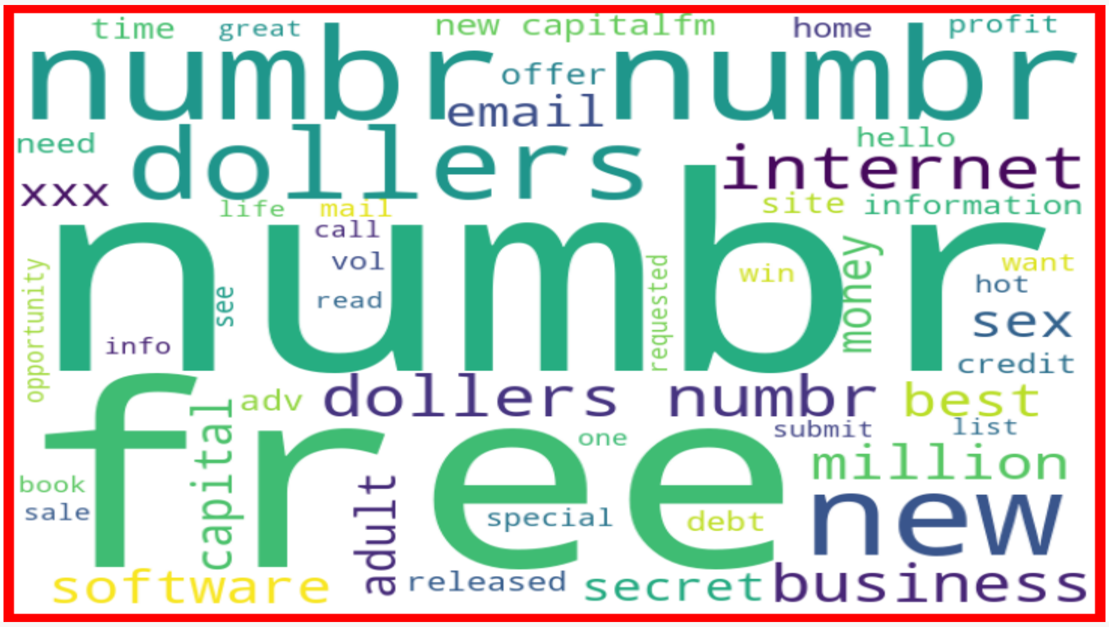
```
#Getting sense of loud words in spam messages
from wordcloud import WordCloud

spams = emails['message'][emails['label']==1]

spam_cloud = WordCloud(width=700,height=500,background_color='white',max_words=50).generate(' '.join(spams))

plt.figure(figsize=(10,8),facecolor='r')
plt.imshow(spam_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



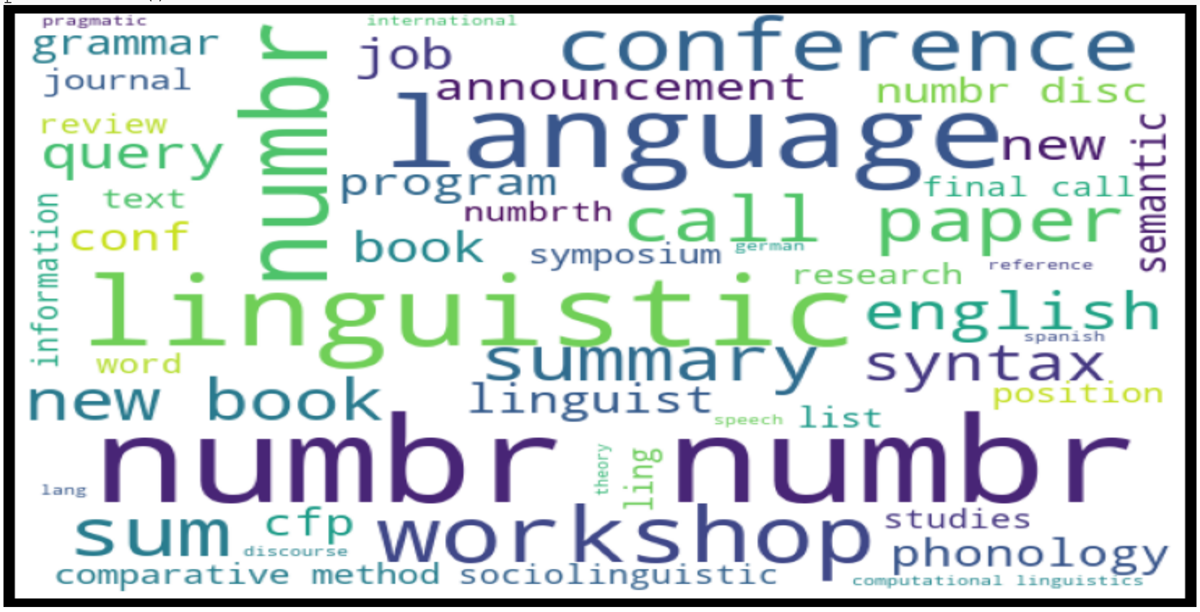
```
#Getting sense of loud words in regular message

hams = emails['message'][emails['label']==0]
spam_cloud = WordCloud(width=600,height=400,background_color='white',max_words=50).
generate(' '.join(hams))
plt.figure(figsize=(10,8),facecolor='k')
plt.imshow(spam_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



```
#Getting sense of loud words in spam emails subject

spams = emails['subject'][emails['label']==1]

spam_cloud = WordCloud(width=700,height=500,background_color='white',max_words=50).
generate(' '.join(spams))

plt.figure(figsize=(10,8),facecolor='r')
plt.imshow(spam_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



```
#Getting sense of loud words in regular email subject

hams = emails['subject'][emails['label']==0]
spam_cloud = WordCloud(width=600,height=400,background_color='white',max_words=50).
generate(' '.join(hams))
plt.figure(figsize=(10,8),facecolor='k')
plt.imshow(spam_cloud)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



Hardware and Software Requirements and Tools Used

- Framework-Annconda
- IDE-Jupyter –Notebook
- Coding Language-Python
- Hardware used: system memory 8GB,
- Processor: core i3

Libraries used: Below are the python library used in this project.

- **Pandas**: To read the Data file in form of data.
- **Matplotlib**: This library is typically used to plot the figures for better visualisation of data.
- **Seaborn**: A advanced version of Matplotlib
- **Scikit Learn**: This is the most important library for Machine Learning since it contains various Machine Learning Algorithms which are used in this project. Scikit Learn also contains Preprocessing library which is used in data preprocessing. Apart from this it contains very useful joblib library for serialization purpose using which final model have been saved in this project.
- **NLTK**: Natural language took kit is one of the most used library for building NLP projects.

Model/s Development and Evaluation

Identification of possible problem

Understanding the problem is a crucial first step in solving any problem.Since it was clear that it’s a binary classification problem. Therefore I decided to run my preprocessed dataset on 6 classification algorithms and then to get observationsfrom their results

Training Classifier:

All tokenized emails are converted into vectors . We have converted text into vector using TF-IDF. Then we have split features and label.

```
# Convert text into vectors using TF-IDF
# Split feature and label
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

tf_vec = TfidfVectorizer()


features = tf_vec.fit_transform(emails['message']+emails['subject'])
x = features
y = emails['label']
```

```
# Train_train_split
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=42,stratify=y
```

Testing of Identified Approaches (Algorithms)

Below are the algorithms used for this problem:

- from sklearn.linear_model import LogisticRegression
- from sklearn.naive_bayes import MultinomialNB
- from sklearn.tree import DecisionTreeClassifier
- from sklearn.ensemble import RandomForestClassifier
- from sklearn.neighbors import KNeighborsClassifier
- from sklearn.ensemble import AdaBoostClassifier

Run and Evaluate selected models

In my approach I have first prepared a method which gives all necessary classification metrics of an algorithm like classification metrics , auc_roc score, confusion matrix

```
models=[]
models.append(('KneighborsClassifier',KNN))
models.append(('LogisticRegression',LR))
models.append(('DecisionTreeClassifier',DT))
models.append(('RandomForestClassifier',RF))
models.append(('AdaBoostClassifier',AD))
models.append(('MultinomialNB',MNB))
```

```python
model_list=[]
score=[]
cvs=[]
rocscore=[]

for name, model in models:
    print('************************',name,'************************',end='\n\n')

    model_list.append(name)
    model.fit(x_train,y_train)
    print(model,end='\n\n')
    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 validation 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)
    print('\n')
    print('classification_report\n',classification_report(y_test,pre))
    print('\n')
    cm=confusion_matrix(y_test,pre)
    print(cm)
    print('\n')
    plt.figure(figsize=(10,40))
    plt.subplot(911)
    plt.title(name)
    print(sns.heatmap(cm,annot=True))
    plt.subplot(912)
    plt.title(name)
    plt.plot(false_positive_rate,true_positive_rate, label='AUC= %0.2f'%roc_auc)
    plt.plot([0,1],[0,1],'r--')
    plt.legend(loc='lower right')
    plt.ylabel('True positive rate')
    plt.xlabel('False positive rate')
    print('\n\n')
```

```
result=pd.DataFrame({'Model': model_list, 'Accuracy_score': score, 'Cross_val_score
':cvs,'Roc_auc_score': rocscore})
result
```

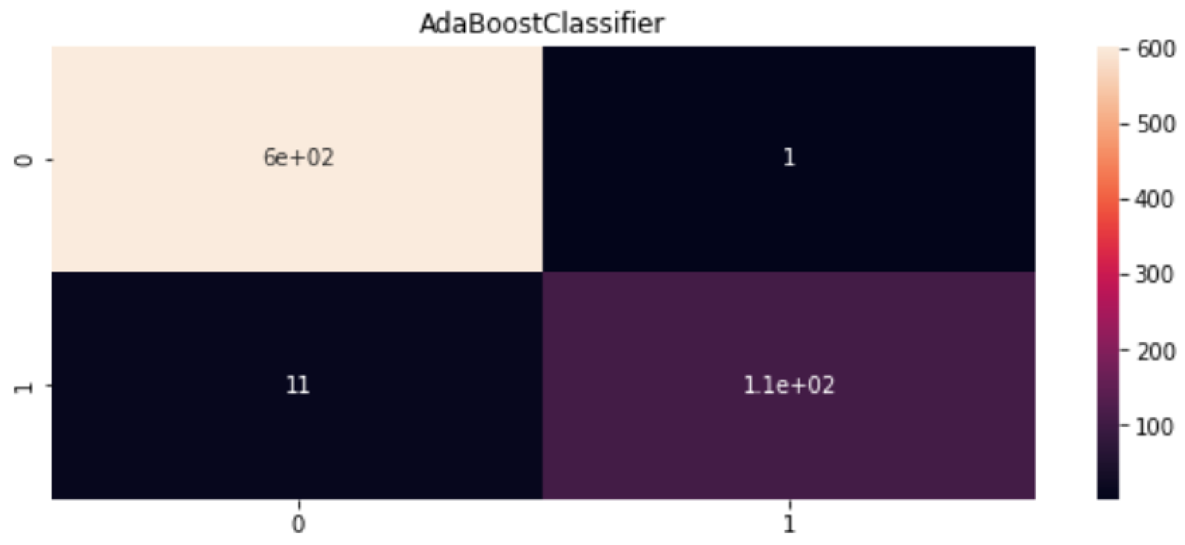
	Model	Accuracy_score	Cross_val_score	Roc_auc_score
0	KneighborsClassifier	95.994475	96.024221	94.594371
1	LogisticRegression	95.303867	95.194607	85.833333
2	DecisionTreeClassifier	94.060773	94.573440	90.430464
3	RandomForestClassifier	97.651934	96.992722	92.916667
4	AdaBoostClassifier	98.342541	97.857416	95.333885
5	MultinomialNB	85.911602	86.035079	57.500000

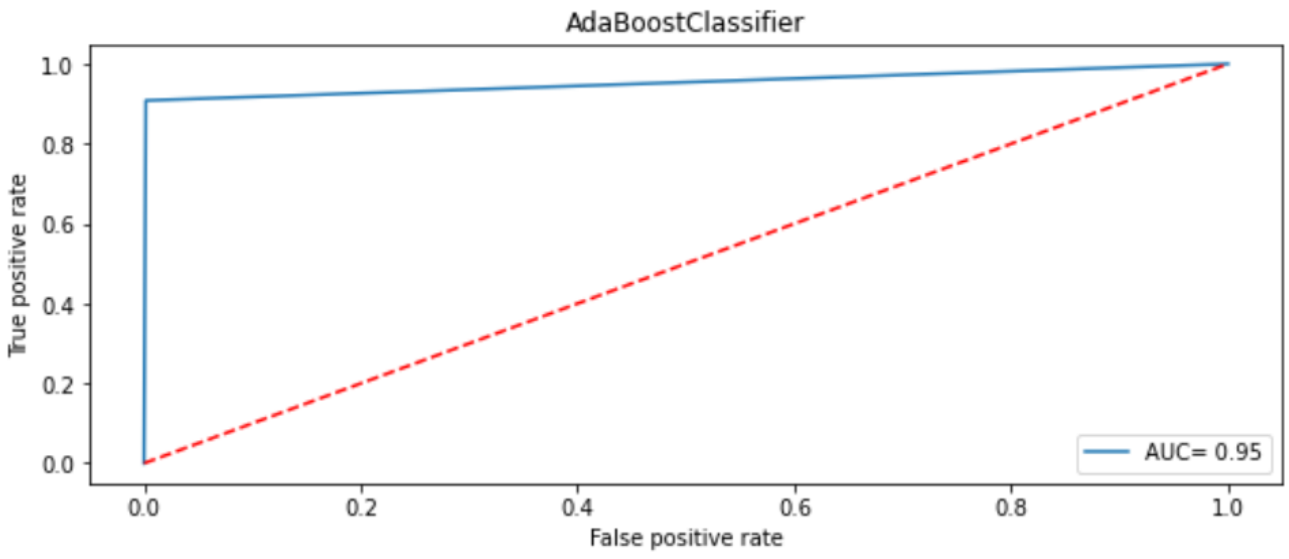
After comparing all models we have observed that Adaboost is the most suitable model for our project.

```
#lets make our ADABOOST CLASSIFIER as final model

AD=AdaBoostClassifier()
AD.fit(x_train,y_train)
ad_pred=AD.predict(x_test)
print(accuracy_score(y_test,ad_pred))
print(confusion_matrix(y_test,ad_pred))
print(classification_report(y_test,ad_pred))
```

```
0.9834254143646409
[[603   1]
 [ 11 109]]
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       604
           1       0.99      0.91      0.95       120

    accuracy                           0.98       724
   macro avg       0.99      0.95      0.97       724
weighted avg       0.98      0.98      0.98       724
```

It only classified 12 out of 724 entries incorrectly.





After that we have saved our Model in pickle file:

```
#save model as a pickle file
import joblib
joblib.dump(AD,'Spam.pkl')

['Spam.pkl']
```

Interpretation of the Results

- The confusion matrix gives the number of TP (603) ,FP(1) ,FN(11) and TN (109).
- It implies total 724 rows were considered for testing out of which only 12were incorrect and 712 were correctly predicted by our model.
- It means98% data was correctly predicted.
- This number are the best among all used algorithms
- The ROC-AUC curve shows the ability to distinguish between 0 and 1 labels. In my final model I got ROC-AUC score of 97% which implies that probabilityof my model to distinguish between 0 and 1 is 0.95

CONCLUSION

Key Findings and Conclusions of the Study

- Dataset was imbalanced with very high number of non-spams in comparison tospam
- feature in the dataset contained NAN values.

- **Learning Outcomes of the Study in respect of Data Science**

I found visualisation a very useful technique to infer insights from dataset.

The ROC AUC plot gives large info about the false positive rate and True positive rate at various thresholds.