FLIP ROBO

Project Report On
E-mail spam classifier

Submitted By:
B. SHEEBARANI

BATCH NO.:1844

ACKNOWLEDGMENT

I express my sincere gratitude to Flip Robo Technologies for giving me the opportunity to work on this project on Email Spam classifier using machine learning algorithms.I would also like to thank Flip Robo Technologies for providing me with the requisite datsets to work with .I acknowledge my indebtedness to the authors of papers titled: Naive Bayes spam classifier and some other article based on spam detection and modelling process such as 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.

INTRODUCTION

BUSINESS PROBLEM FRAMING:

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.

Conceptual Background of the domain problem

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.

Review of literature

Machine learning techniques now days used to automatically filter the spam e-mail in a very successful rate. In this project we review some of the most popular machine learning methods (Bayesian classification, k-NN, ANNs, SVMs, Artificial immune system and Rough sets) and of their applicability to the problem of spam Email classification.

Descriptions of the algorithms are presented, and the comparison of their performance on the Spam Assassin spam corpus is presented. Electronic mail has eased communication methods for many organizations as well as individuals. This method is exploited for fraudulent gain by spammers through sending unsolicited emails. This article aims to present a method for detection of spam emails with machine learning algorithms that are optimized with bio-inspired methods. A literature review is carried to explore the efficient methods applied on different datasets to achieve good results. An extensive research was done to implement machine learning models using Naïve

Bayes, Support Vector Machine, Random Forest, Decision Tree and Multi-Layer Perceptron on seven different email datasets, along with feature extraction and pre- processing.

Some Useful Links-

1-https://www.javatpoint.com/nlp

2-https://www.educative.io/answers/preprocessing-steps-in-natural-language-processing-nlp

3-https://www.youtube.com/watch?v=5ctbvkAMQO4

4-https://www.youtube.com/watch?v=X2vAabgKiuM

Analytical Problem Framing:

Mathematical /Analytical modelling of the problem

Models used: **Naive Bayes and SVM**: Email spam classification done using traditional machine learning techniques comprise Baive Bayes and SVM (support vector machines), due to not having sufficient hardware resources, takes less time to train. Also, not opting for neural algorithms due to less data and computing resources.

Reason for choosing SVM and Naïve Bayes: Both are good at handling large number of features; in the case of text classification each word is a feature and we have thousands of words based on the vocabulary of the corpus. SVM works best with high dimensional data, a vocabulary with 1000 words means each text in the corpus will be represented with a vector of 1000 dimension.

When we have a sufficient number of features, both SVM and Naïve Bayes can work with less data as well.

Naïve Bayes does not suffer from curse-of-dimensionality because it treats all features as independent of one another. Also, one of the benefits of features being independent is: For example, most spam emails contain words such as money and investment, etc, but it is not necessary that all the mails containing both words money and investment are considered to be spam.

Feature representation: Word embeddings can be broadly classified into two categories: Frequency and prediction-based. I have chosen a count vector that shows the count of occurrence of a feature in the given document, thus it is a matrix of document vs vocabulary (containing all the features as a column).

In our case, the size of the Count vector matrix is 5728 x 20114, where 5728 represents the number of documents in the corpus and 20114 represents the number of features in the vocabulary.

Splitting Training and Testing Data: Splitting the data into training and test datasets, where training data contains 80 percent and test data contains 20 percent.

Applying model SVM and Naïve Bayes: I trained the model for both SVM and Naive without tuning hyperparameters as I got results with default parameter settings.

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.

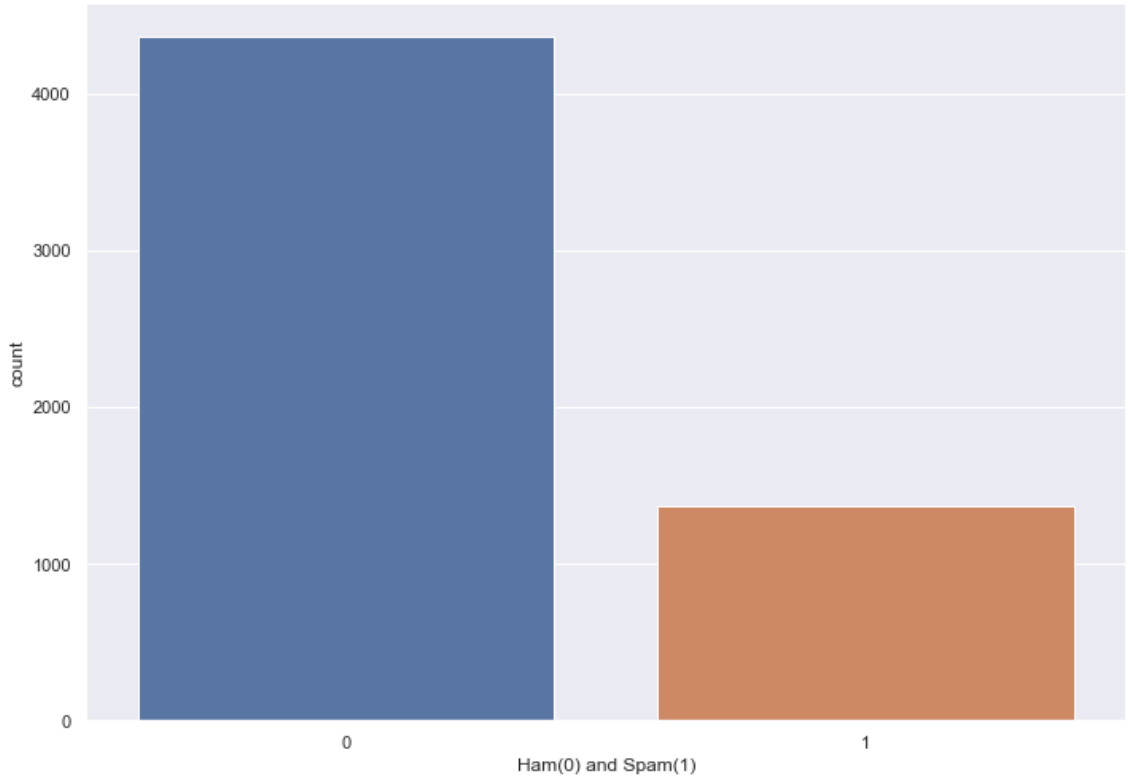
Data Description

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).

	v1	v2	Label
0	ham	Go until jurong point, crazy.. Available only ...	43
1	ham	Ok lar... Joking wif u oni...	43
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	43
3	ham	U dun say so early hor... U c already then say...	43
4	ham	Nah I don't think he goes to usf, he lives aro...	43
...	...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...	43
5568	ham	Will Ì_ b going to esplanade fr home?	43
5569	ham	Pity, * was in mood for that. So...any other s...	43
5570	ham	The guy did some bitching but I acted like i'd...	43
5571	ham	Rofl. Its true to its name	43

Data exploration:

The bar graph given below depicts the percentage of Ham and Spam emails in the given dataset. The blue bar represents the count of ham emails and the red bar shows the count of spam emails in the dataset.



Data preparation

The following steps we used for data preparation.

- Identifying Missing values.
- Converting all text to lower case.
- Performing tokenization.
- Removing Stop words.
- Labelling classes: ham/spam: {0;1}
- Splitting Train and Test Data: 80% and 20%.

Preprocess Data

In this step, we will preprocess our data to a format that can be fed into our machine training process.

The Multinomial NB classifier we will be training later requires the input data to be in word vector count or tf-idf vectors. We need a way to turn our email text into vectors of numbers that represent the frequency of each word in the text documents. We can manually write code to do it, but this is a common task and sklearn has modules that make these tasks really easy. We can use Count Vectorizer to create word vector count or we can use TfidfVectorizer to create tf-idf vectors.

Tf-idf stands for term frequency - inverse document frequency. The main idea is this will give more weight to words that occur less frequently.

```
from sklearn.feature_extraction.text import TfidfVectorizer

tf_vec=TfidfVectorizer()
feature1=tf_vec.fit_transform(df['v2'],df['v1'])
```

```
x=feature1
```

Model Training

In this step, we will train our model. Sklearn has many classification algorithms we can choose from. In this problem, we are using Naive Bayes algorithm. It's popular in text classification because of its relative simplicity. In sklearn, the Naive Bayes classifier is implemented in MultinomialNB. MultinomialNB needs the input data in word vector count or tf-idf vectors which we have prepared in data preparation steps. Below is the code that we will need in the model training step. You can see how easy it is to train a Naïve Bayes classifer in sklearn.

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.

Data Retrieval Flow

The program flow also shows how we retrieved the data, we have used CSV file, which we directly downloaded from the Kaggle email spam data set link. The below steps mention how we read and done operation on csv file using pandas.

Creating object on the parent class: data_read_write

```
data_obj = data_read_write("emails.csv")
```

We create an object by initializing it using the dataset file emails.csv which is passed to the constructor. It will read the name of the file and store it in file_link variable which is a string type and return the object reference. We can now read the CSV file by calling read_csv_file() function defined in our parent class data_read_write by accessing it through the object. The function will return the content of the file as pandas dataframe

```
data_frame = data_obj.read_csv_file()
```

Now, we can print the top 5 rows of our dataframe

```
data_frame.head()
```

The below table shows the text and spam, as two columns, the text feature is the descriptive feature which contains the email: subject and body content. The

spam column contains two ham and spam class labels, where 0 refers to ham and 1 refers to spam

	v1	v2	Label	length	sub_length
0	ham	Go until jurong point, crazy.. Available only ...	43	111	3
1	ham	Ok lar... Joking wif u oni...	43	29	3
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	43	155	4
3	ham	U dun say so early hor... U c already then say...	43	49	3
4	ham	Nah I don't think he goes to usf, he lives aro...	43	61	3

Then we plotted histogram for distribution of mean word length used in spam and ham category using the below code snippet.

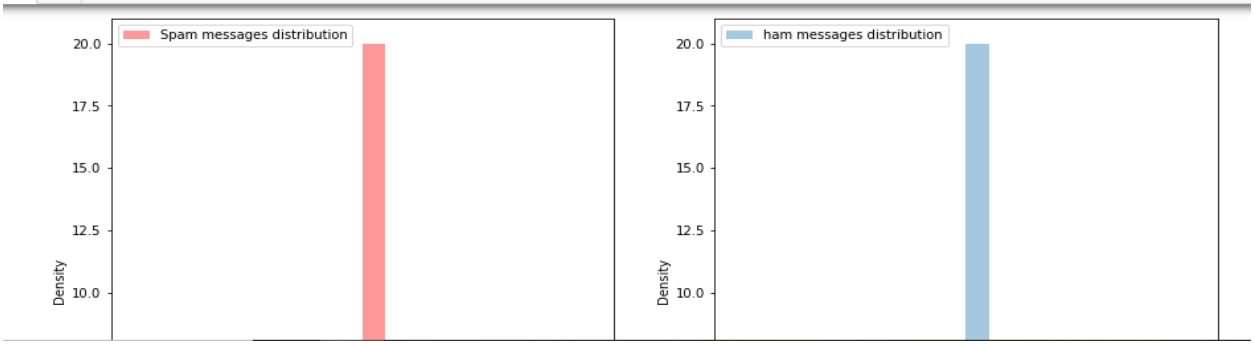
```
#Remove stopwords
import string
import nltk
from nltk.corpus import stopwords

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

df['v2'] = df['v2'].apply(lambda x: ' '.join(term for term in x.split() if term not in stop_words))
```

```
df['clean_length'] = df.v2.str.len()
```

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

df['v1'] = df['v1'].apply(lambda x: ' '.join(term for term in x.split() if term not in stop_words))
```

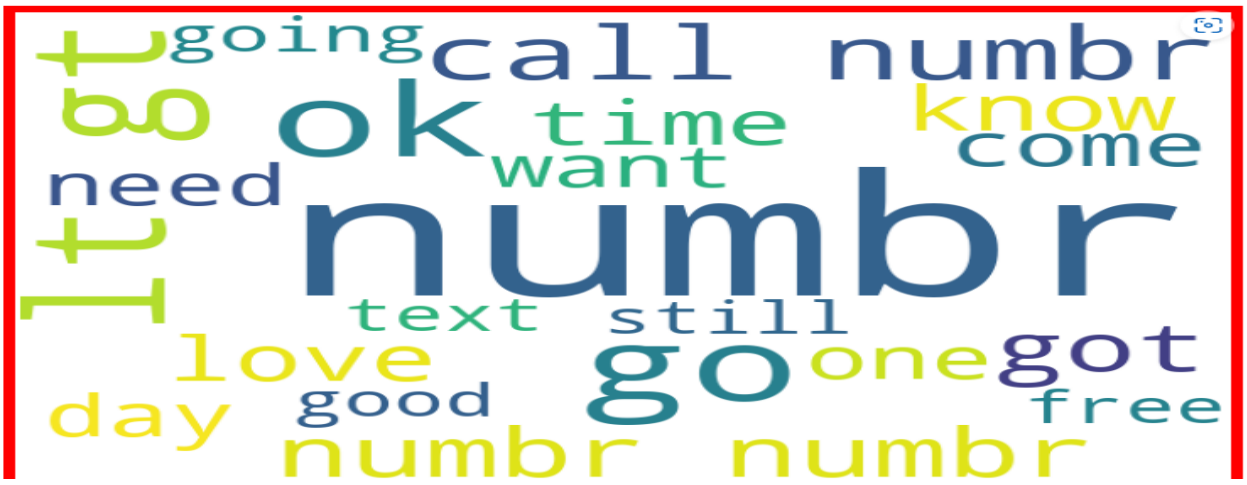
```
df['clean_sublength'] = df.v1.str.len()
df.head()
```

```
2]:
#message distribution after cleaning

f,ax = plt.subplots(1,2,figsize=(15,8))

sns.distplot(df[df['Label']==1]['clean_length'],bins=20,ax=ax[0],label='Spam messages distribution',color='r')
ax[0].set_xlabel('Spam sms length')
ax[0].legend()

sns.distplot(df[df['Label']==0]['clean_length'],bins=20,ax=ax[1],label='ham messages distribution')
ax[1].set_xlabel('ham sms length')
ax[1].legend()

plt.show()
f.savefig('sms distribution')
```



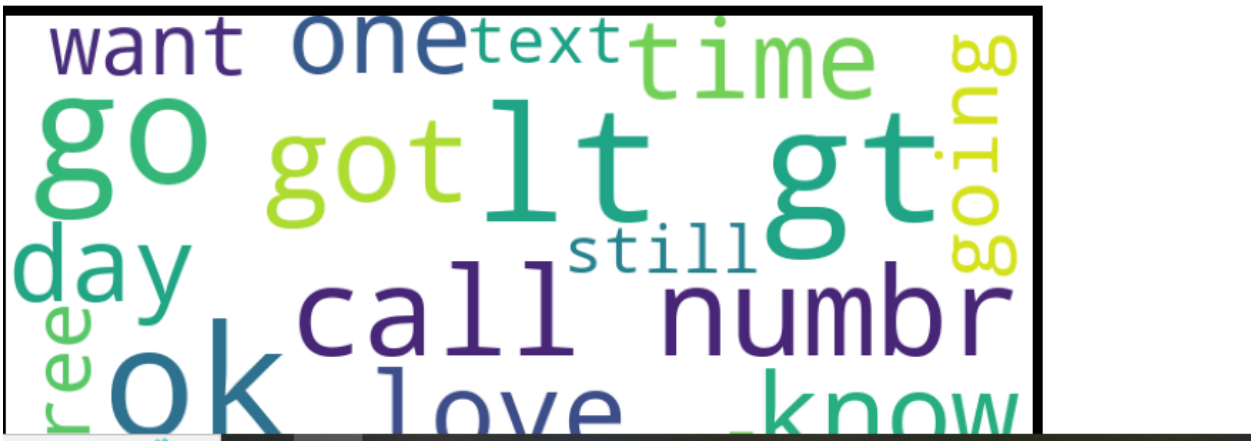
Word cloud for spam emails

```
#getting sense of message Loud words in spam
from wordcloud import WordCloud

spams=df['v2']

spam_cloud = WordCloud(width=700,height=500,background_color='white',max_words=20).generate(' '.join(spams))

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

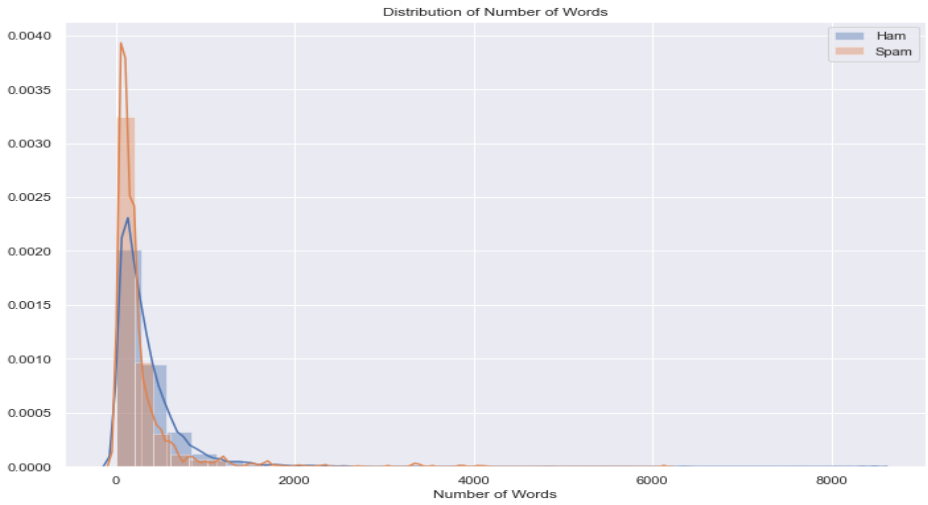


Wordcloud for ham emails:

```
#getting sense of message Loud words in ham

spams=df['v2']

spam_cloud = WordCloud(width=700,height=500,background_color='white',max_words=20).generate(' '.join(spams))

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



Distribution of the number of words: From the below figure we can see that, there is a spike in spam emails with a smaller number of words, even when our dataset includes 24 percent of spam emails out of total emails.



The model algorithms used were as follows:

- Logistic Regression: It is a classification algorithm used to find the probability of event success and event failure. It is used when the dependent variable is binary(0/1, True/False, Yes/No) in nature. It supports categorizing data into discrete classes by studying the relationship from a given set of labelled data. It learns a linear relationship from the given dataset and then introduces a non-linearity in the form of the Sigmoid function. It not only provides a measure of how appropriate a predictor(coefficient size)is, but also its direction of association (positive or negative).
- DecisionTree Classifier: Decision Tree solves the problem of machine learning by transforming the data into a tree representation. Each internal node of the tree representation denotes an attribute and each leaf node denotes a class label. A decision tree does not require normalization of data. A decision tree does not require normalization of data.
- XGBClassifier: XGBoost uses decision trees as base learners; combining many weak learners to make a strong learner. As a result it is referred to as an ensemble learning method since it uses the output of many models in the final prediction. It uses the power of parallel processing and supports regularization.
- RandomForestClassifier: A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. A random forest produces good predictions that can be understood easily. It reduces overfitting and can handle large datasets efficiently. The random forest algorithm provides a higher level of accuracy in predicting outcomes over the decision tree algorithm.
- AdaBoost Classifier: The basis of this algorithm is the Boosting main core: give more weight to the misclassified observations. the meta-learner adapts based upon the results of the weak classifiers, giving more weight to the misclassified observations of the last weak learner. The individual learners can be weak, but as long as the performance of each weak learner is better than random guessing, the final model can converge to a strong learner (a learner not influenced by outliers and with a

```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,roc_curve,auc
from sklearn.model_selection import GridSearchCV,cross_val_score
```

```
KNN=KNeighborsClassifier(n_neighbors=6)
LR=LogisticRegression()
DT=DecisionTreeClassifier(random_state=6)
XGB=XGBClassifier()
RF=RandomForestClassifier()
ADA=AdaBoostClassifier()
MNB=MultinomialNB()

```

```
models= []
models.append(('KNeighborsClassifier', KNN))
models.append(('LogisticRegression', LR))
models.append(('DecisionTreeClassifier', DT))
models.append(('XGBClassifier', XGB))
models.append(('RandomForestClassifier', RF))
models.append(('AdaBoostClassifier', ADA))
models.append(('MultinomialNB', MNB))
```

bservation:we can conclude that RandomForest Classifier as our best fitting model which is giving very less difference compare to other models

```
Model= []
score= []
cvs=[]
rocscore=[]
for name,model in models:
```

```
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, pos_label=1)
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.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')
```

```
**************************************** KNeighborsClassifier **********************************************************


KNeighborsClassifier(n_neighbors=6)


Accuracy_score =  0.9928251121076234


Cross_Val_Score =  0.9910265567588785


roc_auc_score =  nan


classification_report
              precision    recall  f1-score   support

          16       0.00      0.00      0.00         1
```

All models are trained.

Analyzing the accuracy of the models and Hyper parameter tuning:

```
# Decision Tree Classifier

parameters = {'criterion':['gini','entropy'],
              'max_features':['auto','sqrt','log2'],
              'max_depth':[10,20,30,40,50],
              'splitter':['best','random']}
```

```
GCV=GridSearchCV(DecisionTreeClassifier(),parameters,cv=5)
GCV.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [10, 20, 30, 40, 50],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'splitter': ['best', 'random']})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
GCV.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 40,
 'max_features': 'log2',
 'splitter': 'best'}
```

```
ESC = DecisionTreeClassifier(criterion='entropy', max_depth=20, max_features='sqrt',splitter='best')
ESC.fit(x_train,y_train)
pred = ESC.predict(x_test)
acc=accuracy_score(y_test,pred)
```

```
GCV.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 40,
 'max_features': 'log2',
 'splitter': 'best'}
```

```
ESC = DecisionTreeClassifier(criterion='entropy', max_depth=20, max_features='sqrt',splitter='best')
ESC.fit(x_train,y_train)
pred = ESC.predict(x_test)
acc=accuracy_score(y_test,pred)
print(acc*100)
```

```
99.19282511210761
```

So here we can see the accuracy of the best model is increased after tuning.

Conclusion:

In this project, I have done some feature engineering by replacing the unwanted entries by suitable values, found no null values, and renamed the columns by giving new names. Visualized the data using count plot, factor plot, pie plot and distribution plot, also encoded the object data into numerical using label encoding method. Checked the statistical summary of the dataset and checked for skewness, outliers and correlation between the features.

Learning outcomes of the study in respect of data science:

Data cleaning was a very important step in removing plenty of anomalous data from the huge dataset that was provided. Visualizing data helped identify outliers and the relationships between target and feature columns as well as analyzing the strength of correlation that exists between them.

Limitations of this work and scope for future work

While the huge dataset to work with enabled the building of highly accurate models. The presence of anomalous entries in the numbers heavily disorted the data distributions and may have had a huge impact on model learning. Predictive model yield more accurate values.