Spam Classifier using Email Data for classifying emails as Spam / Ham

1) Introduction: -

In recent times, unwanted commercial bulk emails called spam has become a huge problem on the internet. The person sending the spam messages is referred to as the spammer. Such a person gathers email addresses from different websites, chatrooms, and viruses. Spam prevents the user from making full and good use of time, storage capacity and network bandwidth. The huge volume of spam mails flowing through the computer networks have destructive effects on the memory space of email servers, communication bandwidth, CPU power and user time. The menace of spam email is on the increase on yearly basis and is responsible for over 77% of the whole global email traffic. Users who receive spam emails that they did not request find it very irritating. It is also resulted to untold financial loss to many users who have fallen victim of internet scams and other fraudulent practices of spammers who send emails pretending to be from reputable companies with the intention to persuade individuals to disclose sensitive personal information like passwords, Bank Verification Number (BVN) and credit card numbers.

To effectively handle the threat posed by email spams, leading email providers such as Gmail, Yahoo mail and Outlook have employed the combination of different machine learning (ML) techniques such as Decision Tree, Support Vector Machine, Naive Bayes in its spam filters.

Though there are several email spam filtering methods in existence, the state-of-the-art approaches are discussed in this paper. I compared Decision Tree, Support Vector Machine, Naive Bayes algorithms on the given dataset to get the maximum accuracy out of it.

2) Problem Statement: -

In this assignment, you are going to build your own custom spam detection system. Assume that you are working as security engineer at some organization and you are asked to solve the problem of rampant email spam affecting employees in the organization. For whatever reason, you are instructed to develop a custom solution instead of using commercial options. Provided with administrator access to the private email servers, you can extract a body of emails for analysis. All the emails are properly tagged by recipients as either “spam” or “ham” (non-spam), so you do not need to spend too much time cleaning the data.

An intuitive method to deal with this problem is to scan through all the email messages and check for words which are very often used by spam messages. There is a pattern associated with spam messages. If an email contains such words, then you can identify or classify it as a spam otherwise as ham. For instance, you notice that the word “lottery” appears in the spam data a lot, but seldom appears in regular emails.

Before using any known Machine Learning models, you are required to first develop a naïve algorithm which will classify a message between spam or ham using a “blacklist” of words.

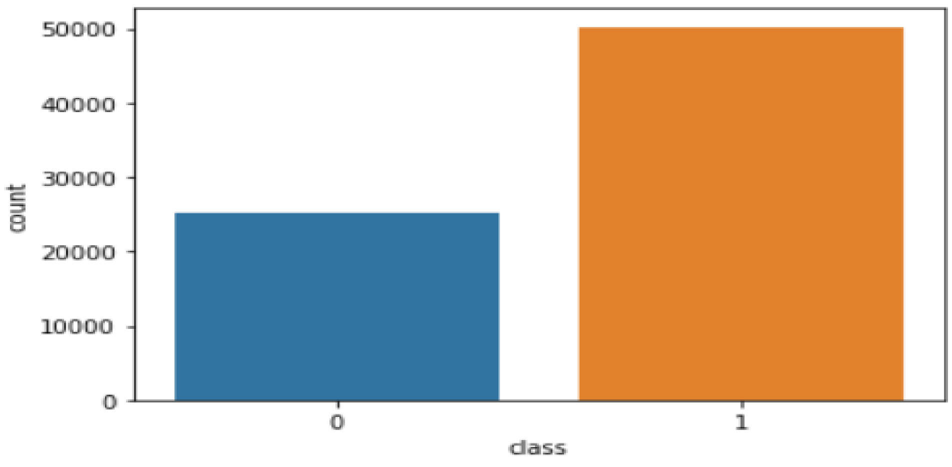
I have taken dataset from 2007 TREC Public Spam Corpus. which is having 75000+ mails on which I am going to test three very popular machine learning algorithms Decision Tree, Support Vector Machine, Naive Bayes.

3) Exploratory Data Analysis: -

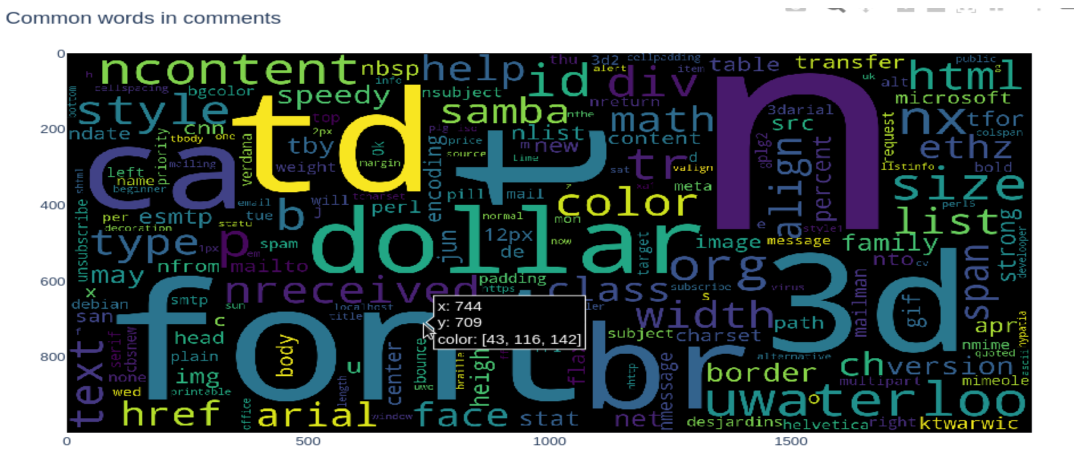
3.1) count plot: - Number of 0’s means ham and number of 1’s means spam.

```
sns.countplot(x="class",data=data)
```

```
<AxesSubplot:xlabel='class', ylabel='count'>
```

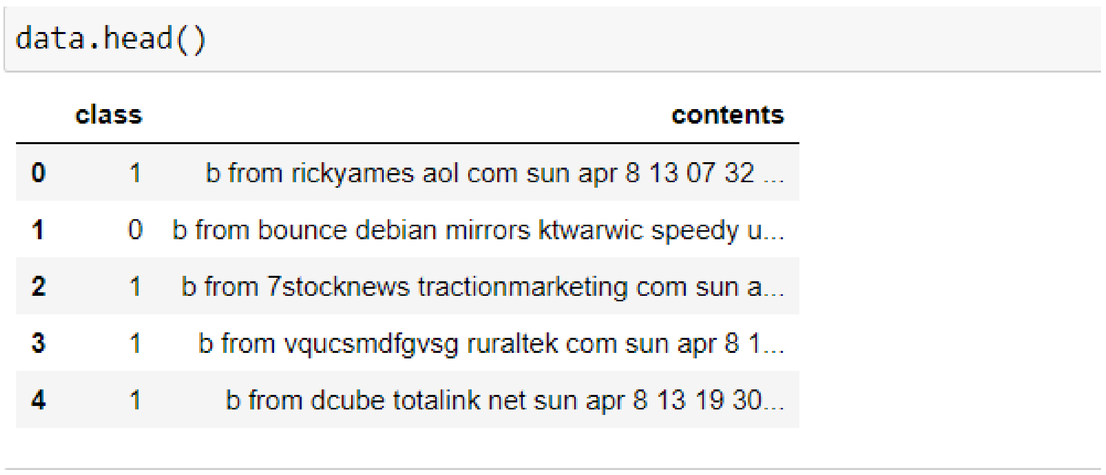


3.2) Word cloud: - A word cloud is a collection, or cluster, of words depicted in different sizes. The bigger and bolder the word appears, the more often it’s mentioned within a given text and the more important it is.



4) Pre-processing of Text:-

- Delete all the duplicates rows
- Removing html tags
- Removing Punctuations
- Performing stemming
- Delete all the duplicates rows



```
data.head()
```

	class	contents
0	1	b from rickyames aol com sun apr 8 13 07 32 ...
1	0	b from bounce debian mirrors ktwarwic speedy u...
2	1	b from 7stocknews tractionmarketing com sun a...
3	1	b from vqucsmdfgvsg ruraltek com sun apr 8 1...
4	1	b from dcube totalink net sun apr 8 13 19 30...

5) What is Natural Language Processing (NLP) ?

Natural Language processing or NLP is a subset of Artificial Intelligence (AI), where it is basically responsible for the understanding of human language by a machine or a robot.

One of the important subtopics in NLP is Natural Language Understanding (NLU) and the reason is that it is used to understand the structure and meaning of human language, and then with the help of computer science transform this linguistic knowledge into algorithms of Rules-based machine learning that can solve specific problems and perform desired tasks.

5.1) Apply tf-idf NLP algo to convert text data to vector –

TF-IDF :- Term Frequency Inverse Document Frequency

EX- If we have totel N documents

N=5 r = documents(comments)

r1 : w1 w2 w3 w2 w5

r2 : w1 w3 w4 w5 w6 w2

r3 : w1 w4 w7 w3 w4

r4 : w1 w6 w9

r5 : w2 w1 w3 w5 w8

step 1 :- Find all distinct(unique) words and count like this-

	w1	w2	w3	w4	w5	w6	w7	w8	w9
r1	1	2	1	0	1	0	0	0	0
r2	1	1	1	1	1	1	0	0	0
r3	1	0	1	2	0	0	1	0	0
r4	1	0	0	0	0	1	0	0	1
r5	1	1	1	0	1	0	0	1	0

TF :-

$$TF(w_i , r_j) = \frac{\text{number if times } w_i \text{ occurd in } r_j}{\text{totel number if words in } r_j}$$

EX-

$$TF(w2 , r1) = \frac{2}{5} = 0.4$$

Note :- $0 <= tf(wi , rj) <= 1$ always

IDF :-

Dc = r1 :
r2 :
r3 :
¦
rN :

Totel we have N documents

DATA CORPAS Dc = { r1 , r2 , r3, ------- ,rN }

$$IDF(W_i , D_c) = \log\left(\frac{N}{n_i}\right)$$

where N = number of documents

ni = number of documents which contain Wi

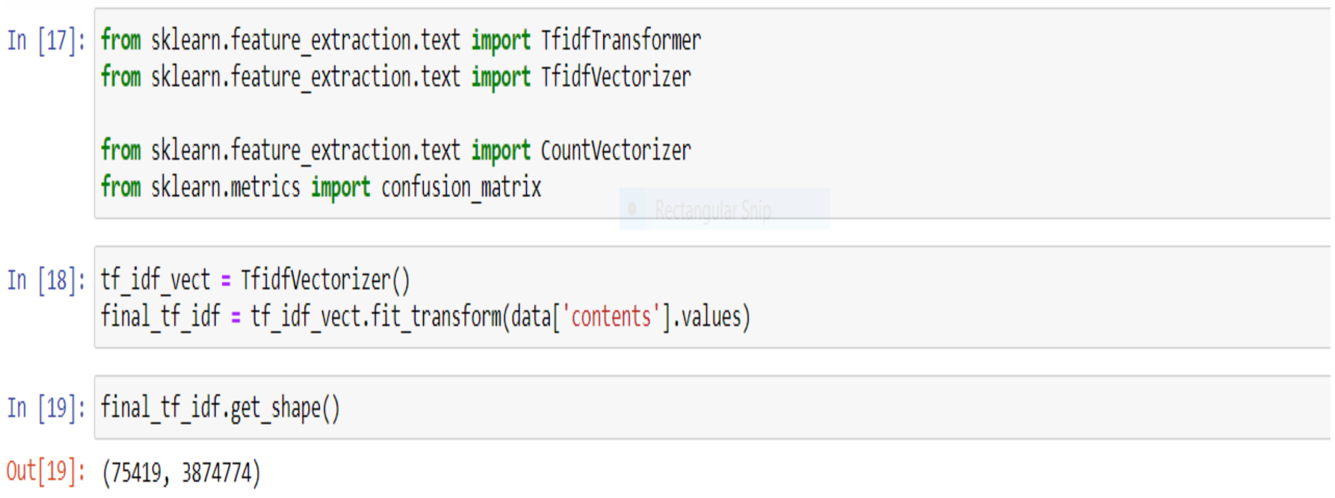
Ex-

IDF(W1 , Dc) = log (5/5) = 0

IDF(W2 , Dc) = log (5/3) = 0.22

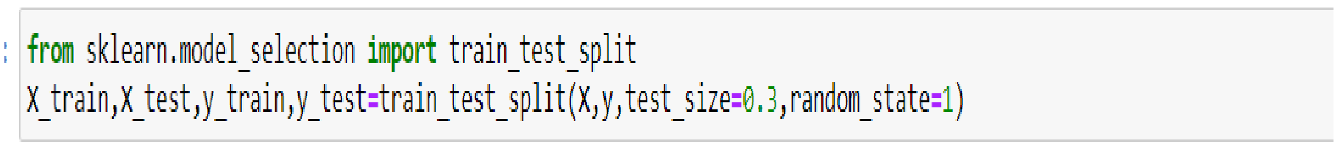
IDF(W9 , Dc) = log (5/1) = 0.69

$$\boxed{TF\text{-}IDF(w_i) = TF(w_i, r_j) * IDF(w_i, D_c)}$$



```
In [17]: from sklearn.feature_extraction.text import TfidfTransformer
         from sklearn.feature_extraction.text import TfidfVectorizer

         from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.metrics import confusion_matrix

In [18]: tf_idf_vect = TfidfVectorizer()
         final_tf_idf = tf_idf_vect.fit_transform(data['contents'].values)

In [19]: final_tf_idf.get_shape()

Out[19]: (75419, 3874774)
```

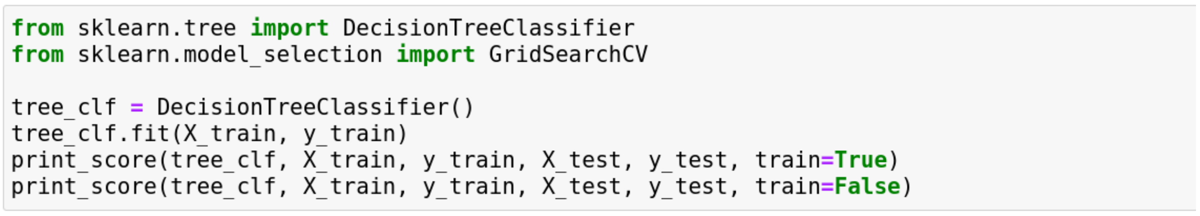
5.2) Split data 70:30 ratio randomly: -



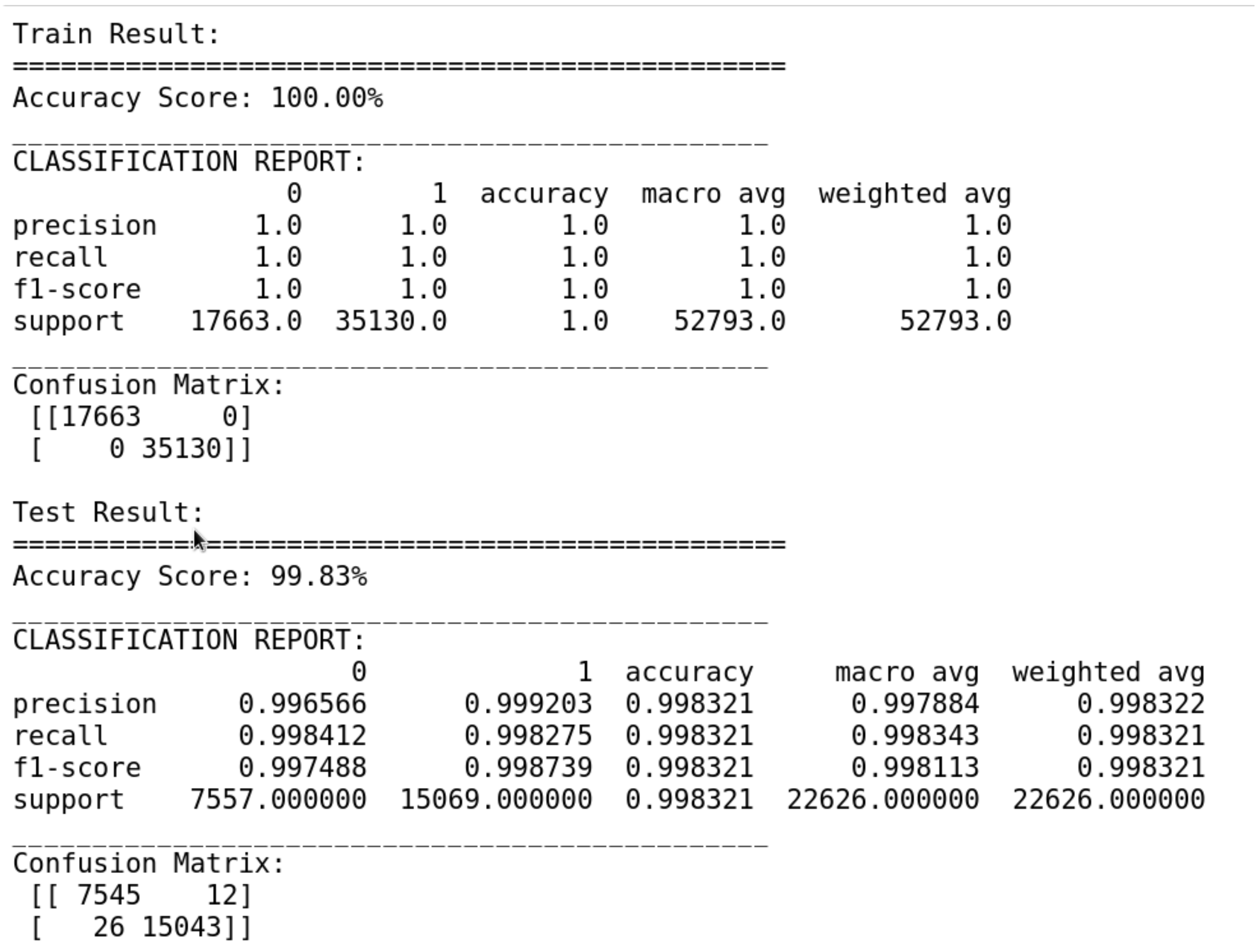
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=1)
```

6) Applying different ML algorithms to analyse which one is best for this dataset: -

6.1) Apply Decision Tree to binary classification: -

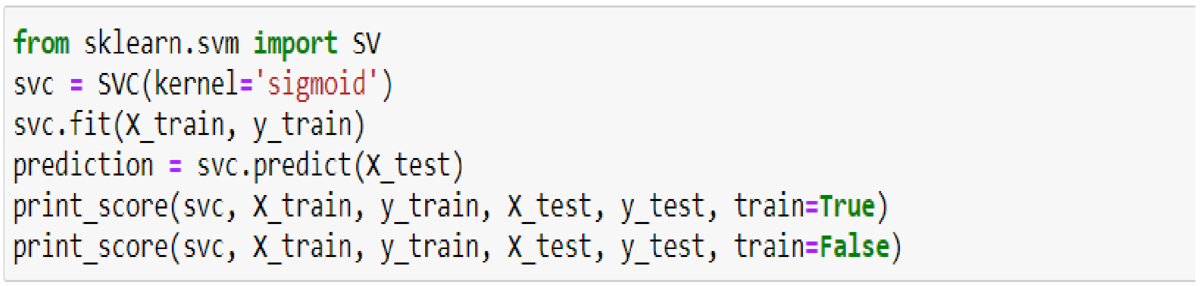


```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

tree_clf = DecisionTreeClassifier()
tree_clf.fit(X_train, y_train)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=True)
print_score(tree_clf, X_train, y_train, X_test, y_test, train=False)
```

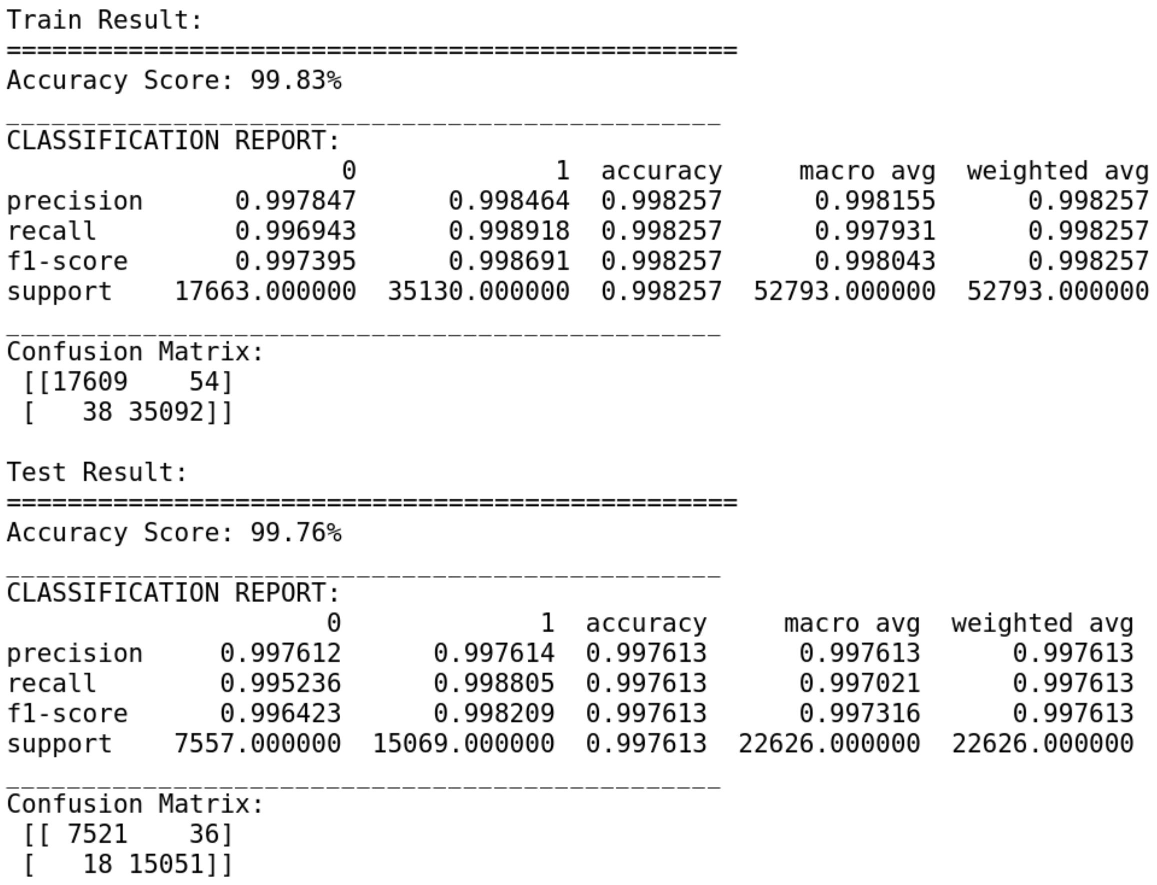


```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
                 0        1  accuracy  macro avg  weighted avg
precision      1.0      1.0       1.0        1.0           1.0
recall         1.0      1.0       1.0        1.0           1.0
f1-score       1.0      1.0       1.0        1.0           1.0
support    17663.0  35130.0       1.0    52793.0       52793.0
_______________________________________________
Confusion Matrix:
 [[17663     0]
 [    0 35130]]

Test Result:
================================================
Accuracy Score: 99.83%
_______________________________________________
CLASSIFICATION REPORT:
                     0             1  accuracy     macro avg  weighted avg
precision     0.996566      0.999203  0.998321      0.997884      0.998322
recall        0.998412      0.998275  0.998321      0.998343      0.998321
f1-score      0.997488      0.998739  0.998321      0.998113      0.998321
support    7557.000000  15069.000000  0.998321  22626.000000  22626.000000
_______________________________________________
Confusion Matrix:
 [[ 7545    12]
 [   26 15043]]
```

6.2) Apply SVM to binary classification: -

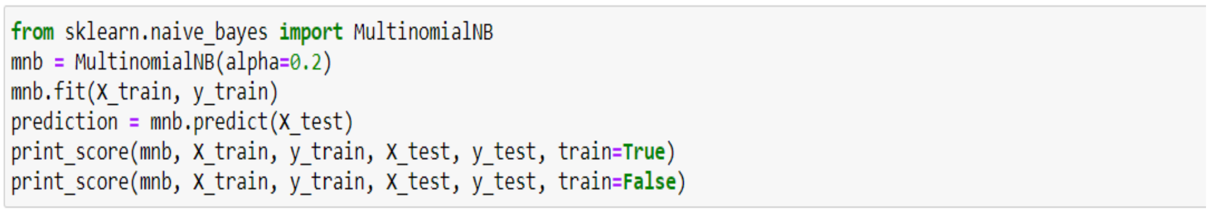


```
from sklearn.svm import SV
svc = SVC(kernel='sigmoid')
svc.fit(X_train, y_train)
prediction = svc.predict(X_test)
print_score(svc, X_train, y_train, X_test, y_test, train=True)
print_score(svc, X_train, y_train, X_test, y_test, train=False)
```

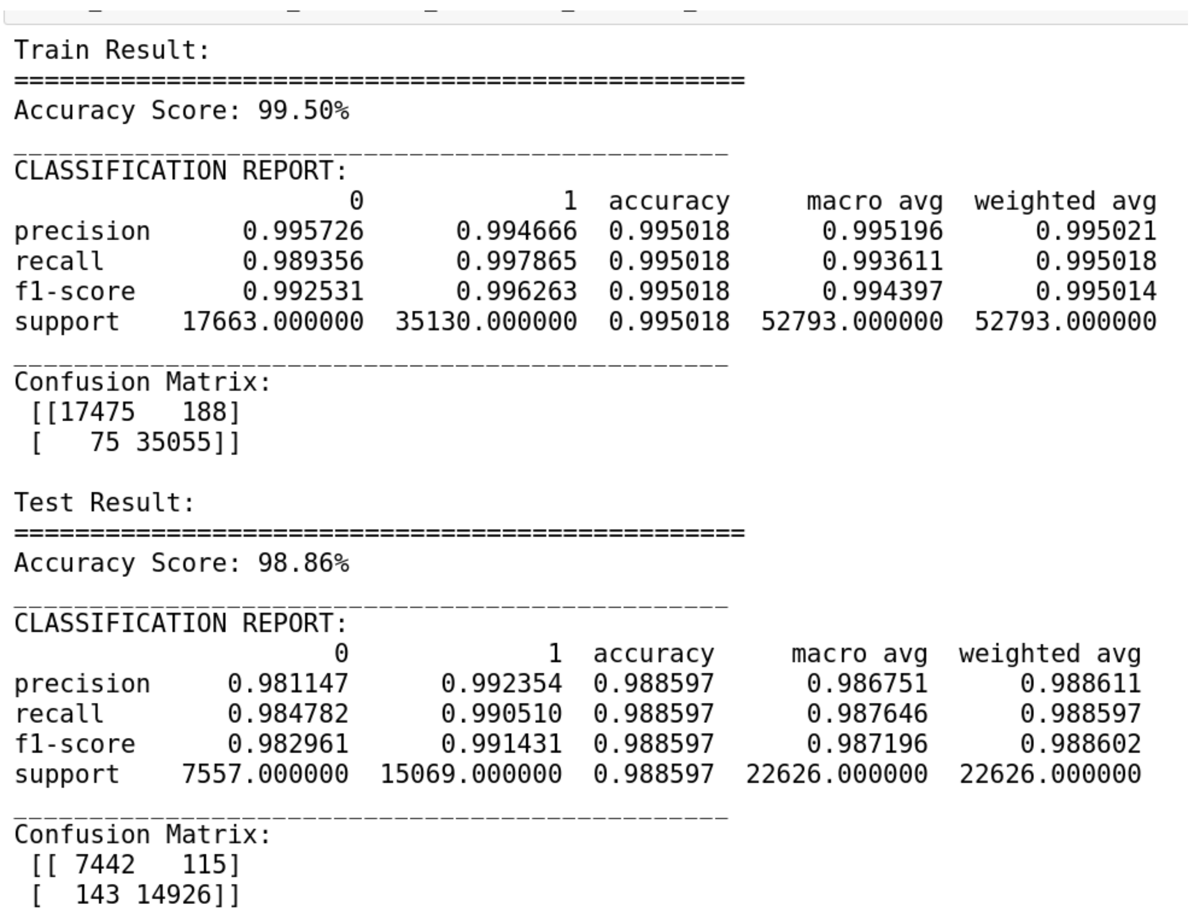


```
Train Result:
================================================
Accuracy Score: 99.83%
_______________________________________________
CLASSIFICATION REPORT:
                      0             1  accuracy     macro avg  weighted avg
precision      0.997847      0.998464  0.998257      0.998155      0.998257
recall         0.996943      0.998918  0.998257      0.997931      0.998257
f1-score       0.997395      0.998691  0.998257      0.998043      0.998257
support    17663.000000  35130.000000  0.998257  52793.000000  52793.000000
_______________________________________________
Confusion Matrix:
 [[17609    54]
 [   38 35092]]

Test Result:
================================================
Accuracy Score: 99.76%
_______________________________________________
CLASSIFICATION REPORT:
                     0             1  accuracy     macro avg  weighted avg
precision     0.997612      0.997614  0.997613      0.997613      0.997613
recall        0.995236      0.998805  0.997613      0.997021      0.997613
f1-score      0.996423      0.998209  0.997613      0.997316      0.997613
support    7557.000000  15069.000000  0.997613  22626.000000  22626.000000
_______________________________________________
Confusion Matrix:
 [[ 7521    36]
 [   18 15051]]
```

6.3) Apply Naive Bayes to binary classification: -



```python
from sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB(alpha=0.2)
mnb.fit(X_train, y_train)
prediction = mnb.predict(X_test)
print_score(mnb, X_train, y_train, X_test, y_test, train=True)
print_score(mnb, X_train, y_train, X_test, y_test, train=False)
```



```
Train Result:
================================================
Accuracy Score: 99.50%
_______________________________________________
CLASSIFICATION REPORT:
                     0             1  accuracy     macro avg  weighted avg
precision      0.995726      0.994666  0.995018      0.995196      0.995021
recall         0.989356      0.997865  0.995018      0.993611      0.995018
f1-score       0.992531      0.996263  0.995018      0.994397      0.995014
support    17663.000000  35130.000000  0.995018  52793.000000  52793.000000
_______________________________________________
Confusion Matrix:
 [[17475   188]
 [   75 35055]]

Test Result:
================================================
Accuracy Score: 98.86%
_______________________________________________
CLASSIFICATION REPORT:
                    0             1  accuracy     macro avg  weighted avg
precision     0.981147      0.992354  0.988597      0.986751      0.988611
recall        0.984782      0.990510  0.988597      0.987646      0.988597
f1-score      0.982961      0.991431  0.988597      0.987196      0.988602
support    7557.000000  15069.000000  0.988597  22626.000000  22626.000000
_______________________________________________
Confusion Matrix:
 [[ 7442   115]
 [  143 14926]]
```

7) **Conclusion :-** we got best accuracy and precision value using Decision Tree algo.

Accuracy :-99.83%

Precision :-99.8321%