Assignment-4 - Apply-Naive Bayes-On-Amazon-Review-Dataset

March 30, 2019

1 Assignment-4: Apply Naive Bayes On Amazon Fine Food Reviews DataSet

1.1 Introduction

(i).Naive Bayes is a simple, yet effective and commonly-used, machine learning classifier. It is a probabilistic classifier that makes classifications using the Maximum A Posteriori decision rule in a Bayesian setting

(ii).Naive Bayes classifiers have been especially popular for text classification, and are a traditional solution for problems such as spam detection.

1.2 Objective

To Predict the Polarity of Amazon Fine Food Review Using Naive Bayes Algorithm.

1.3 Importing All Required Library

```
In [1]: %matplotlib inline
        import sqlite3
        import pandas as pd
        import numpy as np
        import nltk
        import string
        import matplotlib.pyplot as plt
        import seaborn as sns
        import math

        from sklearn.model_selection import GridSearchCV
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.naive_bayes import MultinomialNB


        from sklearn.metrics import classification_report
        from sklearn.feature_extraction.text import TfidfTransformer
        from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.model_selection import TimeSeriesSplit

from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing

import pickle

from tqdm import tqdm
import os
import warnings
warnings.filterwarnings("ignore")
```

1.4 Importing Amazon Fine Food Review Dataset

```
In [2]: if os.path.isfile("final.sqlite"):
            conn=sqlite3.connect("final.sqlite")
            Data=pd.read_sql_query("select * from NaiveBayes where Score!=3",conn)
            conn.close()
        else :
            print("Error Importing the file")
```

1.5 Taking 150K Random Points

```
In [ ]: Data=Data.sample(n=150000)

In [3]: # Printing some data of DataFrame

        Data['Score'].value_counts()

Out[3]: 1    126439
        0     23561
        Name: Score, dtype: int64
```

1.6 Information About DataSet

```
In [4]: print("\nNumber of Reviews: ",Data["Text"].count())
        print("\nNumber of Users: ",len(Data["UserId"].unique())) # Unique returns 1-D array o
        print("\nNumber of Products: ",len(Data["ProductId"].unique()))
        print("\nShape of Data: ", Data.shape)
        print("\nColumn Name of DataSet : ",Data.columns)
        print("\n\nNumber of Attributes/Columns in data: 12")
        print("\nNumber of Positive Reviews : ", Data['Score'].value_counts()[1])
        print("\nNumber of Negative Reviews : ", Data['Score'].value_counts()[0])
```

```
Number of Reviews:  150000
```

```
Number of Users:  115632

Number of Products:  43130

Shape of Data:  (150000, 12)

Column Name of DataSet :  Index(['index', 'Id', 'ProductId', 'UserId', 'ProfileName',
       'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score', 'Time',
       'Summary', 'Text', 'CleanedText'],
      dtype='object')


Number of Attributes/Columns in data: 12

Number of Positive Reviews :  126439

Number of Negative Reviews :  23561
```

In [5]:
```python
print("\nNumber of Reviews: ",Data["Text"].count())
```

```
Number of Reviews:  150000
```

1.7 Attribute Information About DataSet

1.Id - A unique value starts from 1
2.ProductId - A unique identifier for the product
3.UserId - A unqiue identifier for the user
4.ProfileName - Name of user profile
5.HelpfulnessNumerator - Number of users who found the review helpful
6.HelpfulnessDenominator - Number of users who indicated whether they found the review helpful or not
7.Score - Rating 0 or 1
8.Time - Timestamp for the review
9.Summary - Brief summary of the review
10.Text - Text of the review
11.Cleaned Text - Text that only alphabets

In [6]:
```python
# Sorting on the basis of Time Parameter
Data.sort_values('Time',inplace=True)
```

In [7]:
```python
Y = Data['Score'].values
X = Data['CleanedText'].values
```

1.8 Splitting DataSet into Train and Test Data

```
In [8]: from sklearn.model_selection import train_test_split
        # X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Fl
        X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33) # this is ra

        print("Shape of Train and Test Dataset for 50k points")
        print(X_train.shape, Y_train.shape)
        print(X_test.shape, Y_test.shape)
```

```
Shape of Train and Test Dataset for 50k points
(100500,) (100500,)
(49500,) (49500,)
```

1.9 Defining Some Function

1.9.1 Train Data Confusion Matrix Plot

```
In [9]: def trainconfusionmatrix(model,X_train,y_train):
            print("Confusion Matrix for Train set")
            cm=confusion_matrix(y_train, model.predict(X_train))
            class_label = ["negative", "positive"]
            df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
            sns.heatmap(df_cm, annot = True, fmt = "d")
            plt.title("Train Confusiion Matrix")
            plt.xlabel("Predicted Label")
            plt.ylabel("True Label")
            plt.show()
```

1.9.2 Test Data Confusion Matrix Plot

```
In [10]: def testconfusionmatrix(model,X_test,y_test):
             print("Confusion Matrix for Test set")
             cm=confusion_matrix(y_test, model.predict(X_test))
             class_label = ["negative", "positive"]
             df_cm = pd.DataFrame(cm, index = class_label, columns = class_label)
             sns.heatmap(df_cm, annot = True, fmt = "d")
             plt.title("Test Confusiion Matrix")
             plt.xlabel("Predicted Label")
             plt.ylabel("True Label")
             plt.show()
```

1.9.3 ROC-AUC Curve Plot

```
In [12]: def plot(Alpha,gsv):

             Res=gsv.cv_results_
             train_auc=Res['mean_train_score']
```

```
train_auc_std=Res['std_train_score']
cv_auc=Res['mean_test_score']
cv_auc_std=Res['std_test_score']


log_alpha=[math.log10(x) for x in Alpha ]
plt.plot(log_alpha, train_auc, label='Train AUC')
plt.gca().fill_between(log_alpha,train_auc - train_auc_std,train_auc + train_auc_s

plt.plot(log_alpha, cv_auc, label='CV AUC')
plt.gca().fill_between(log_alpha,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.
plt.legend()
plt.xlabel("Log(Alpha): hyperparameter")
plt.ylabel("AUC")
plt.title("Plot Between AUC & Log(Alpha)")
plt.show()
```

1.9.4 GridSearchCV

```
In [13]: def Grid_SearchCV(X_train,Y_train,alpha):

             tscv = TimeSeriesSplit(n_splits=10)
             M_NB = MultinomialNB()

             gsv=GridSearchCV(M_NB,alpha,cv=tscv,verbose=1,scoring='roc_auc')
             gsv.fit(X_train,Y_train)

             return gsv
```

1.9.5 30 Informative Feature

```
In [36]:  def show_30_informative_feature(vectorizer,model,n=30):

             # For Negative Class
             neg_class_prob_sorted = model.feature_log_prob_[0, :].argsort()
             neg_feat=[vectorizer.get_feature_names()[x] for x in neg_class_prob_sorted[-n:]]
             neg_prob=[model.feature_log_prob_[0, :][x] for x in neg_class_prob_sorted[-n:]]

             neg_zip=list(zip(neg_feat,neg_prob))
             neg_zip.sort()

             # For Positive Class
             pos_class_prob_sorted = model.feature_log_prob_[1, :].argsort()
             pos_feat=[vectorizer.get_feature_names()[x] for x in pos_class_prob_sorted[-n:]]
             pos_prob=[model.feature_log_prob_[0, :][x] for x in pos_class_prob_sorted[-n:]]
             pos_zip=list(zip(pos_feat,pos_prob))
             pos_zip.sort()
```

```
        top=zip(pos_zip,neg_zip)

        print("{0:20}{1:55}{2:20}".format("S.N","Positive","Negative"))
        print("_"*90)
        i=1
        for (fn_1,coef_1), (fn_2,coef_2) in top:
            print("%d.\t\t%.3f\t%-30s\t\t%.3f\t%s" % (i,coef_1, fn_1, coef_2, fn_2))
            i+=1
```

1.10 Bags of Words

```
In [15]: vectorizer = CountVectorizer()
         vectorizer.fit(X_train) # fit has to happen only on train data

         # we use the fitted CountVectorizer to convert the text to vector
         X_train_bow = vectorizer.transform(X_train)
         X_train_bow=preprocessing.normalize(X_train_bow)

         X_test_bow = vectorizer.transform(X_test)
         X_test_bow=preprocessing.normalize(X_test_bow)

         print("Shape of Train and Test Data After vectorizations")
         print(X_train_bow.shape, Y_train.shape)
         print(X_test_bow.shape, Y_test.shape)
```

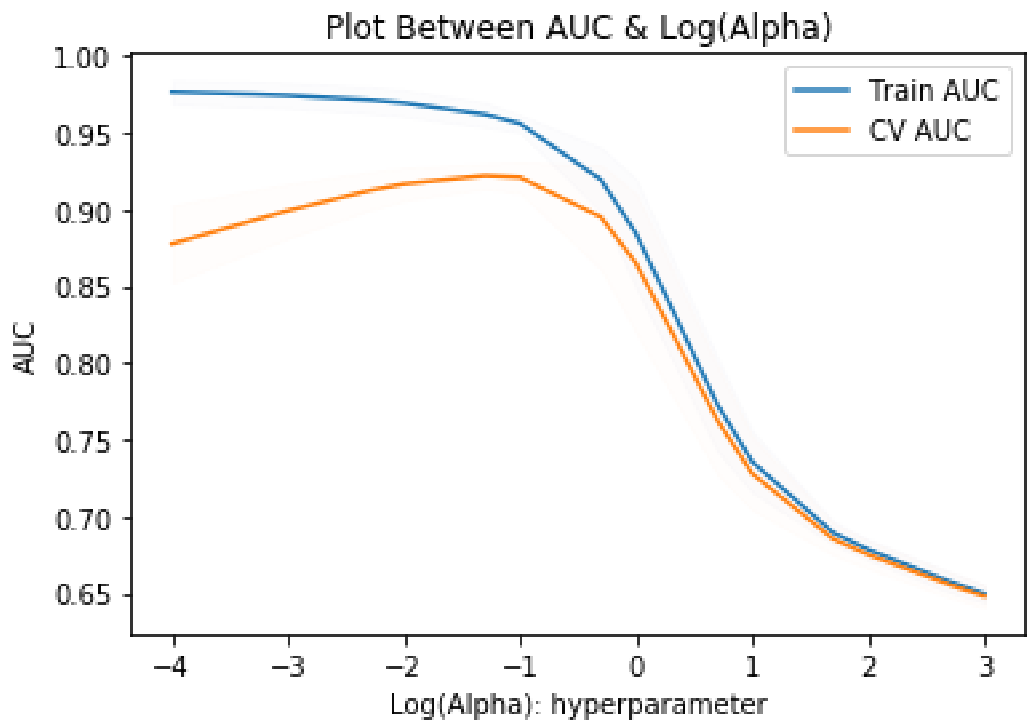
```
Shape of Train and Test Data After vectorizations
(100500, 38189) (100500,)
(49500, 38189) (49500,)
```

1.10.1 Finding the best value Of hyperparameter (Alpha)

```
In [70]: Alpha={'alpha':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]}
         gsv=Grid_SearchCV(X_train_bow,Y_train,Alpha)

         print("Best HyperParameter: ",gsv.best_params_)
         print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits


[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 150 out of 150 | elapsed:   15.1s finished


Best HyperParameter:  {'alpha': 0.05}
Best Accuracy: 92.22%
```

```
In [71]: plot(Alpha['alpha'],gsv)
```



1.10.2 Training the model

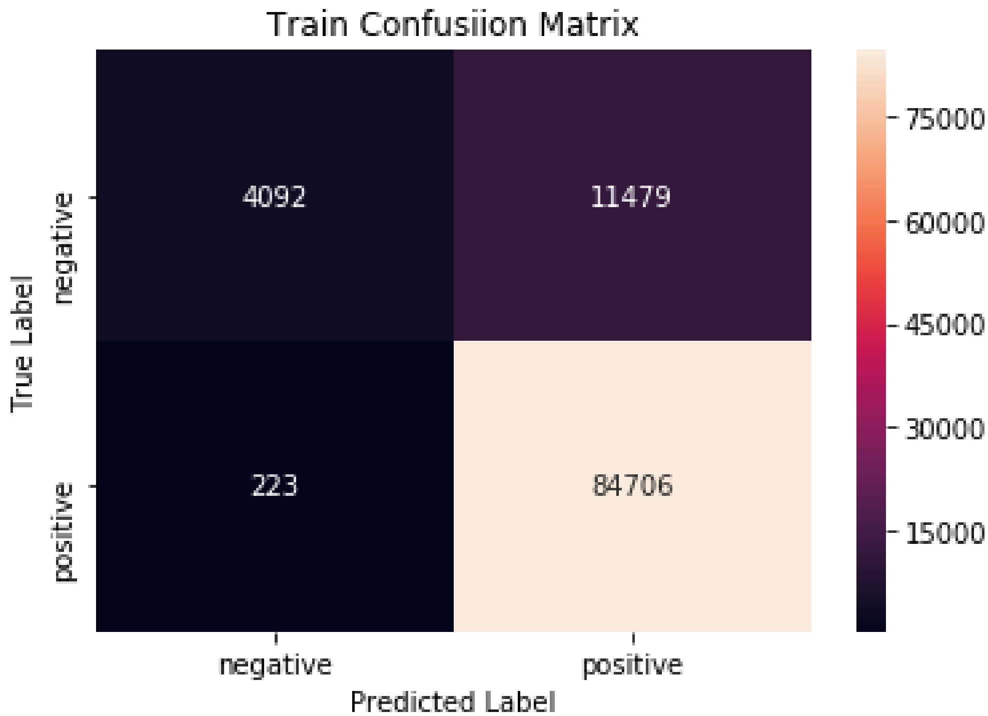
```
In [72]: model_FBOW=MultinomialNB(alpha=gsv.best_params_['alpha'])
         model_FBOW.fit(X_train_bow,Y_train)
```

```
Out[72]: MultinomialNB(alpha=0.05, class_prior=None, fit_prior=True)
```

1.10.3 Evaluating the performance of model

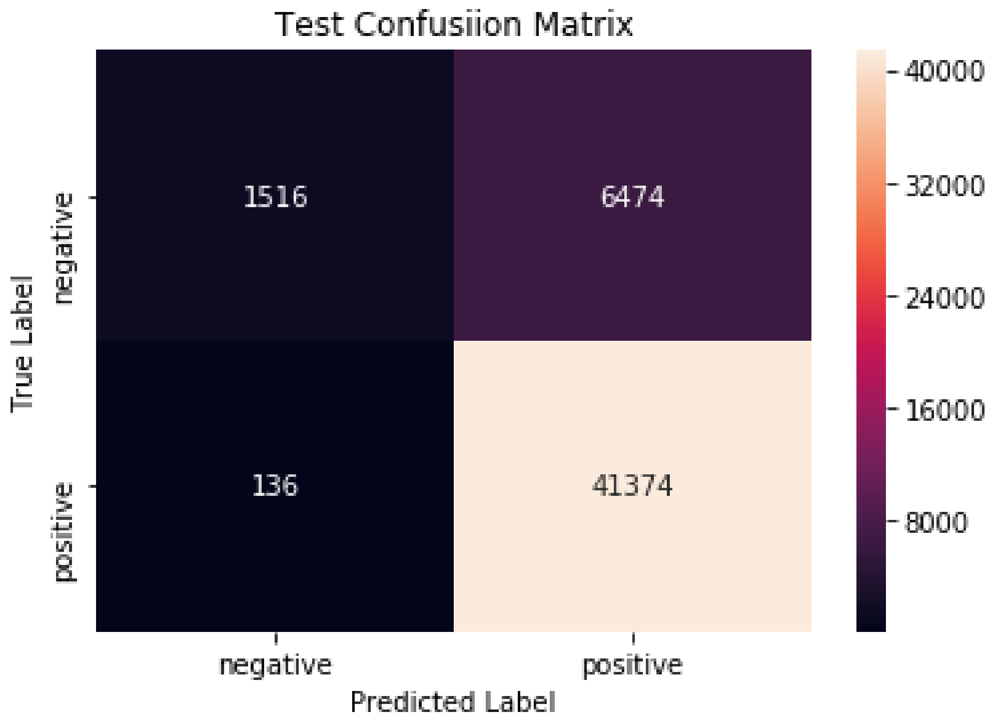
```
In [73]: trainconfusionmatrix(model_FBOW,X_train_bow,Y_train)
```

```
Confusion Matrix for Train set
```

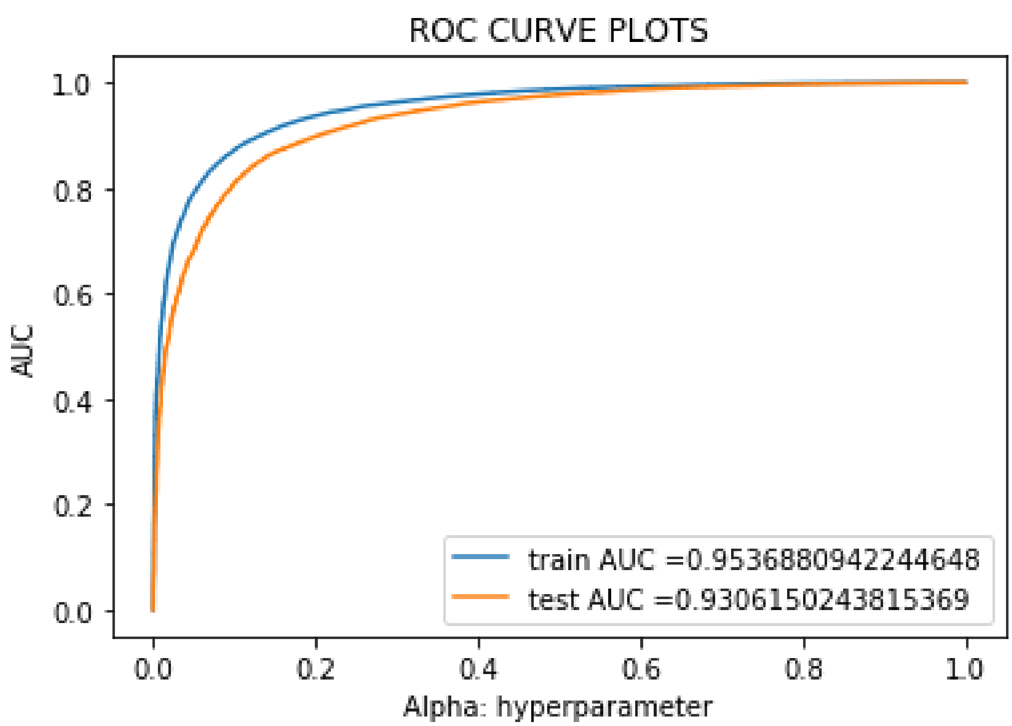



```
In [74]: testconfusionmatrix(model_FBOW,X_test_bow,Y_test)
```

```
Confusion Matrix for Test set
```




```
In [75]: plot_auc_roc(model_FBOW,X_train_bow,X_test_bow,Y_train,Y_test)
```



```
In [76]: print("Classification Report: \n")
         y_pred=model_FBOW.predict(X_test_bow)

         print(classification_report(Y_test, y_pred))
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.92      0.19      0.31      7990
           1       0.86      1.00      0.93     41510

   micro avg       0.87      0.87      0.87     49500
   macro avg       0.89      0.59      0.62     49500
weighted avg       0.87      0.87      0.83     49500
```

1.10.4 Displaying 30 most informative feature

```
In [77]: show_30_informative_feature(vectorizer,model_FBOW)
```

```
S.N                    Positive                                                 Negative

------------------------------------------------------------------------------------------
1.                 -5.943       also                                      -5.412        ama
2.                 -5.412       amazon                                    -5.505        bag
3.                 -6.689       best                                      -5.319        bor
4.                 -5.004       buy                                       -5.004        buy
5.                 -5.043       coffe                                     -5.043        cof
6.                 -5.238       dont                                      -5.536        dis
7.                 -5.507       eat                                       -5.238        don
8.                 -6.103       find                                      -5.507        eat
9.                 -4.750       flavor                                    -5.372        eve
10.                 -5.395       food                                      -4.750        fl
11.                 -5.145       get                                       -5.395        fo
12.                 -5.004       good                                      -5.145        ge
13.                 -5.886       great                                     -5.004        go
14.                 -4.260       like                                      -4.260        li
15.                 -6.045       littl                                     -5.588        lo
16.                 -5.524       love                                      -5.524        lo
17.                 -5.544       make                                      -5.544        ma
18.                 -5.474       much                                      -5.474        mu
19.                 -4.746       one                                       -4.746        on
20.                 -5.065       order                                     -5.065        or
21.                 -5.688       price                                     -5.579        pa
22.                 -4.366       product                                   -4.366        pr
```

23.	-5.509	realli	-5.609	pu
24.	-5.986	store	-5.509	re
25.	-4.148	tast	-4.148	ta
26.	-5.423	tea	-5.423	te
27.	-5.485	time	-5.485	ti
28.	-4.885	tri	-4.885	ti
29.	-5.072	use	-5.072	us
30.	-4.857	would	-4.857	wo

1.11 TF-IDF

In [83]:
```
vectorizer_tfidf=TfidfVectorizer()
vectorizer_tfidf.fit(X_train)
```

Out[83]:
```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

In [84]:
```
X_Train_Tfidf=vectorizer_tfidf.transform(X_train)
X_Train_Tfidf=preprocessing.normalize(X_Train_Tfidf)

X_Test_Tfidf=vectorizer_tfidf.transform(X_test)
X_Test_Tfidf=preprocessing.normalize(X_Test_Tfidf)
```

In [85]:
```
print("Shape of Train and Test Data After vectorizations")
print(X_Train_Tfidf.shape, Y_train.shape)
print(X_Test_Tfidf.shape, Y_test.shape)
```

```
Shape of Train and Test Data After vectorizations
(100500, 38440) (100500,)
(49500, 38440) (49500,)
```

1.11.1 Finding the best value of hyperparameter(Alpha)

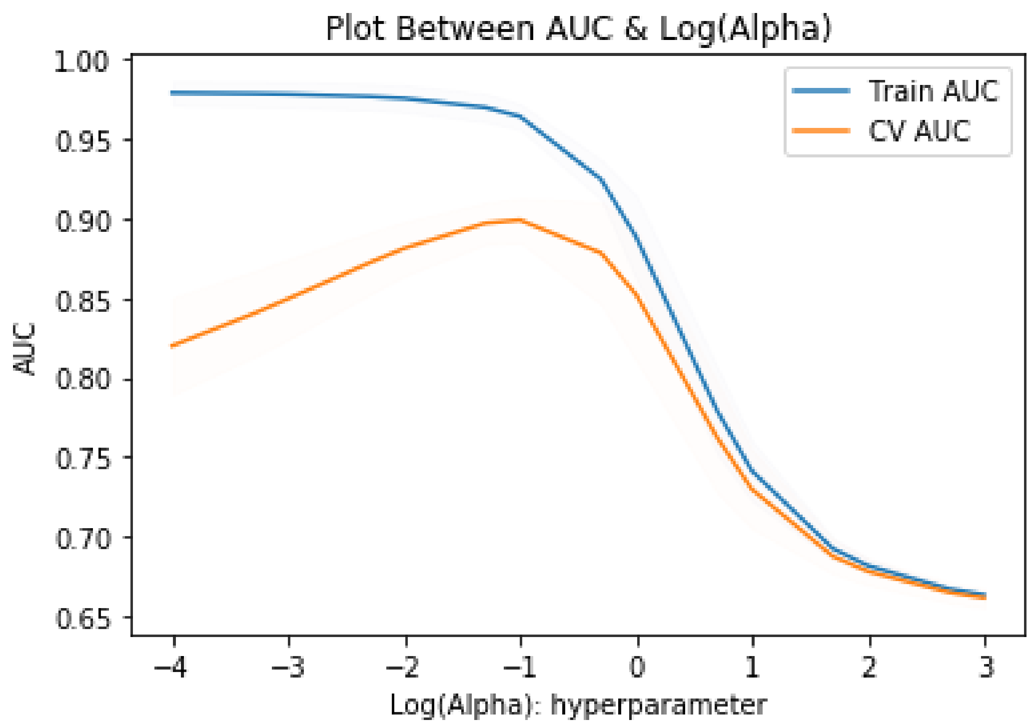
In [86]:
```
Alpha={'alpha':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]}
gsv=Grid_SearchCV(X_Train_Tfidf,Y_train,Alpha)

print("Best HyperParameter: ",gsv.best_params_)
print("Best Accuracy: %.2f%%"%(gsv.best_score_*100))
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 150 out of 150 | elapsed:   15.3s finished
```

```
Best HyperParameter:  {'alpha': 0.1}
Best Accuracy: 89.89%
```

In [87]:
```
plot(Alpha['alpha'],gsv)
```



1.11.2 Training the model

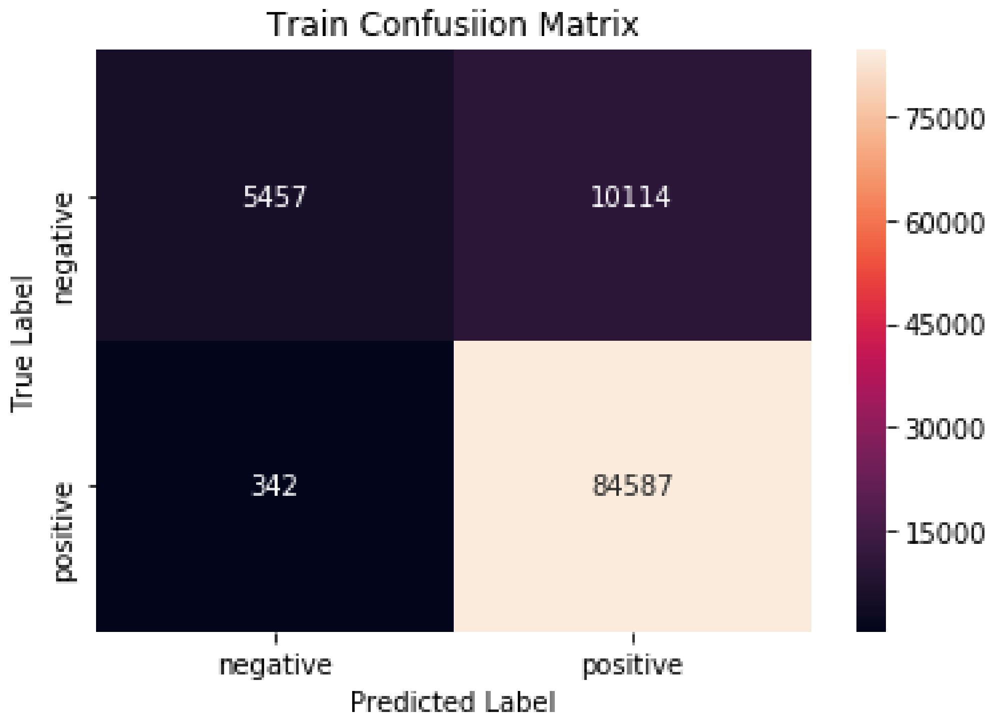
In [88]:
```
model_Tfidf=MultinomialNB(alpha=gsv.best_params_['alpha'])
model_Tfidf.fit(X_Train_Tfidf,Y_train)
```

Out[88]:
```
MultinomialNB(alpha=0.1, class_prior=None, fit_prior=True)
```

1.11.3 Evaluating the performance of model

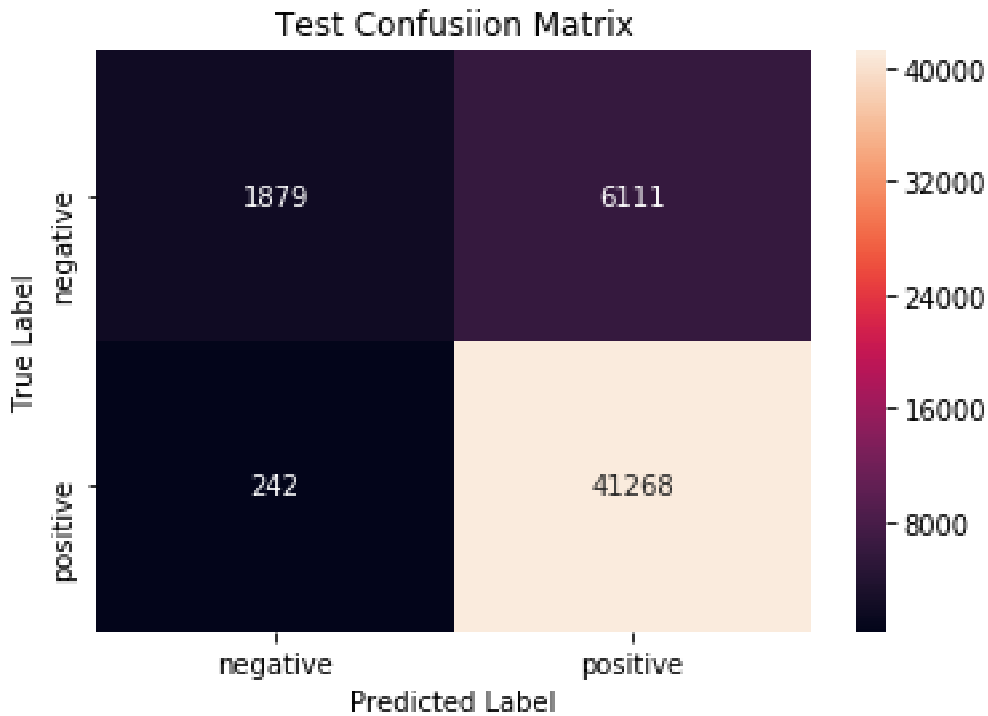
In [89]:
```
trainconfusionmatrix(model_Tfidf,X_Train_Tfidf,Y_train)
```

```
Confusion Matrix for Train set
```

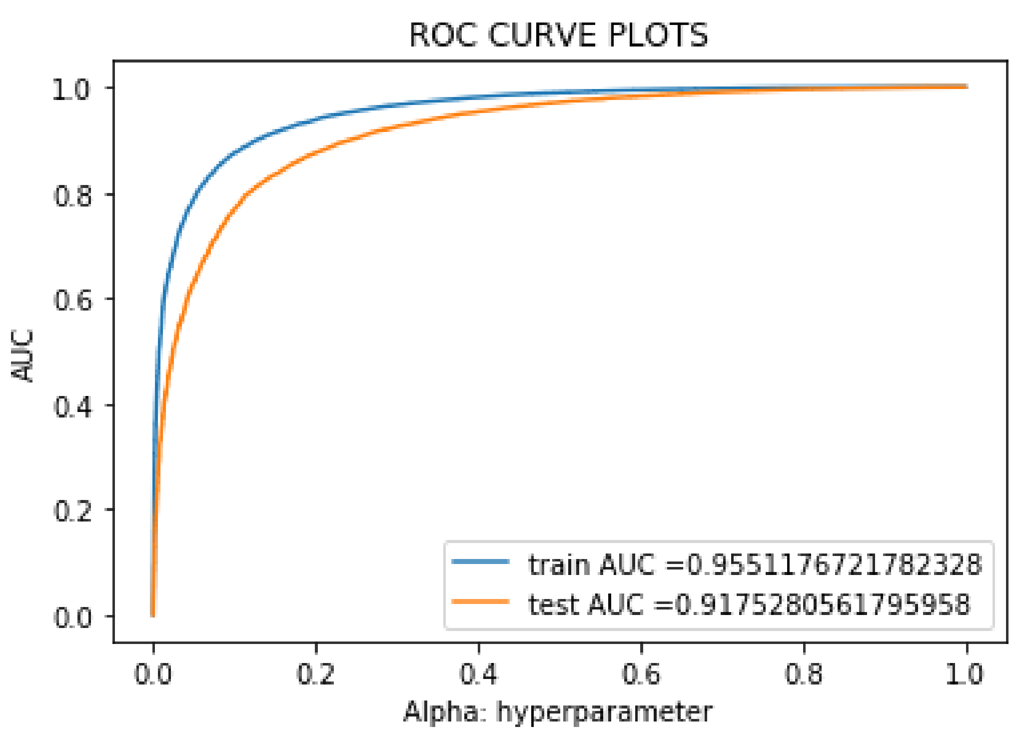


In [90]: testconfusionmatrix(model_Tfidf,X_Test_Tfidf,Y_test)

```
Confusion Matrix for Test set
```



In [91]: `plot_auc_roc(model_Tfidf,X_Train_Tfidf,X_Test_Tfidf,Y_train,Y_test)`



```
In [92]: print("Classification Report: \n")
         y_pred=model_Tfidf.predict(X_Test_Tfidf)

         print(classification_report(Y_test, y_pred))
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.89      0.24      0.37      7990
           1       0.87      0.99      0.93     41510

   micro avg       0.87      0.87      0.87     49500
   macro avg       0.88      0.61      0.65     49500
weighted avg       0.87      0.87      0.84     49500
```

1.11.4 Displaying 30 most informative feature

```
In [93]: show_30_informative_feature(vectorizer_tfidf,model_Tfidf)
```

```
S.N                    Positive                                                     Negative

-----------------------------------------------------------------------------------------
1.                 -5.902        amazon                                       -5.902        ama
2.                 -7.158        best                                         -5.850        bad
3.                 -5.529        buy                                          -5.856        bag
4.                 -5.401        coffe                                        -5.951        bou
5.                 -5.940        dog                                          -5.642        bor
6.                 -6.222        drink                                        -5.529        buy
7.                 -5.965        eat                                          -5.401        co
8.                 -6.562        find                                         -5.657        dis
9.                 -5.409        flavor                                       -5.940        dog
10.                 -5.827        food                                         -5.711        do
11.                 -5.749        get                                          -5.965        ea
12.                 -5.729        good                                         -5.813        ev
13.                 -6.584        great                                        -5.409        fl
14.                 -5.054        like                                         -5.827        fo
15.                 -6.489        littl                                        -5.749        ge
16.                 -6.247        love                                         -5.729        go
17.                 -6.146        make                                         -5.054        li
18.                 -5.945        much                                         -5.953        lo
19.                 -5.428        one                                          -5.968        mo
20.                 -5.528        order                                        -5.945        mu
21.                 -6.138        price                                        -5.428        on
22.                 -5.048        product                                      -5.528        or
```

23.	-5.980	realli	-5.914	pa
24.	-6.387	store	-5.048	pr
25.	-4.916	tast	-5.938	pu
26.	-5.728	tea	-4.916	ta
27.	-5.999	time	-5.728	te
28.	-5.528	tri	-5.528	ti
29.	-5.754	use	-5.754	us
30.	-5.390	would	-5.390	wo

1.12 Addition of another column length

In [16]:
```
Train_len=[]
Test_len=[]
for i in X_train:
    Train_len.append(len(i))

for i in X_test:
    Test_len.append(len(i))
```

In [17]:
```
Train_len=np.array(Train_len)
Test_len=np.array(Test_len)
```

In [18]:
```
Train_len=Train_len[:,np.newaxis]
Test_len=Test_len[:,np.newaxis]
```

1.13 Bag Of Words

In [19]:
```
X_Train_BOW=X_train_bow.todense()
```

In [20]:
```
X_Train_New=np.append(X_Train_BOW,Train_len,axis=1)
```

In [21]:
```
from scipy.sparse import csr_matrix
X_Train_New= csr_matrix(X_Train_New)
```

In [22]:
```
print("Shape of Train Data Before Adding length column ")
print(X_train_bow.shape)

print("\nShape of Train Data After Adding length column ")
print(X_Train_New.shape)
```

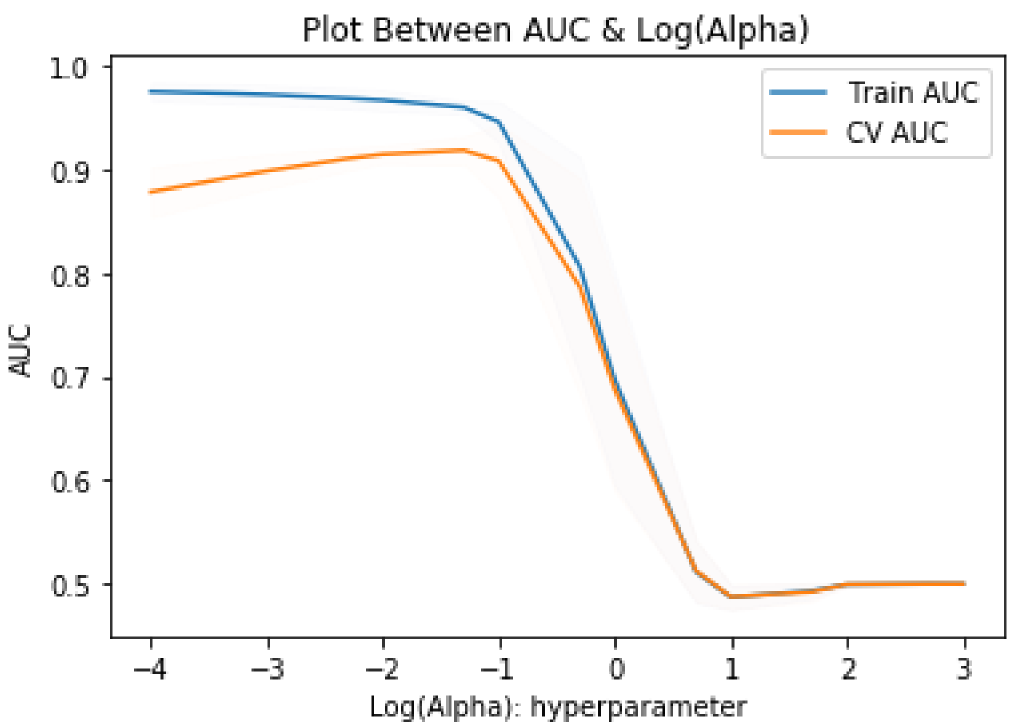
```
Shape of Train Data Before Adding length column
(100500, 38189)

Shape of Train Data After Adding length column
(100500, 38190)
```

1.13.1 Finding the best value of hyperparameter (Alpha)

In [23]:
```
Alpha={'alpha':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]}
gsv1=Grid_SearchCV(X_Train_New,Y_train,Alpha)

print("Best HyperParameter: ",gsv1.best_params_)
print("Best Accuracy: %.2f%%"%(gsv1.best_score_*100))
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits


[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 150 out of 150 | elapsed:   31.2s finished


Best HyperParameter:  {'alpha': 0.05}
Best Accuracy: 91.83%
```

In [24]:
```
plot(Alpha['alpha'],gsv1)
```



In [25]:
```
X_Test_Bow=X_test_bow.todense()
```

In [26]:
```
X_Test_New=np.append(X_Test_Bow,Test_len,axis=1)
```

In [27]:
```
X_Test_New=csr_matrix(X_Test_New)
```

In [28]:
```
print("Shape of Test Data Before Adding length column ")
print(X_test_bow.shape)

print("\nShape of Test Data After Adding length column ")
print(X_Test_New.shape)
```

```
Shape of Test Data Before Adding length column
(49500, 38189)

Shape of Test Data After Adding length column
(49500, 38190)
```

1.13.2 Training the model

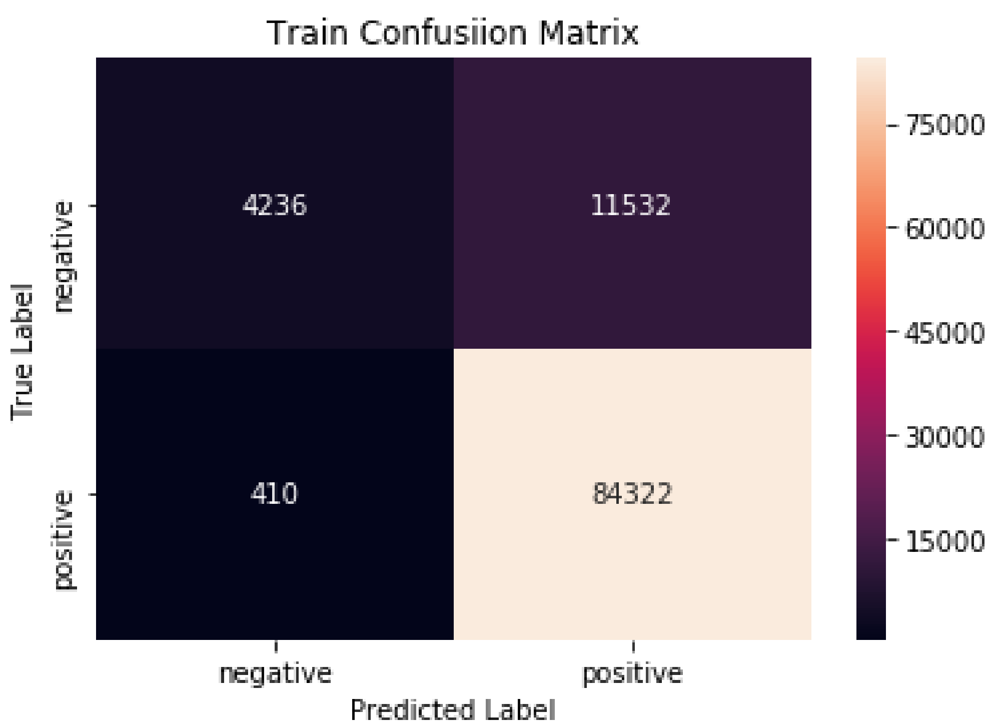
In [29]:
```
model_New_Bow=MultinomialNB(alpha=gsv1.best_params_['alpha'])
model_New_Bow.fit(X_Train_New,Y_train)
```

Out[29]:
```
MultinomialNB(alpha=0.05, class_prior=None, fit_prior=True)
```

1.13.3 Evaluating the performance of model

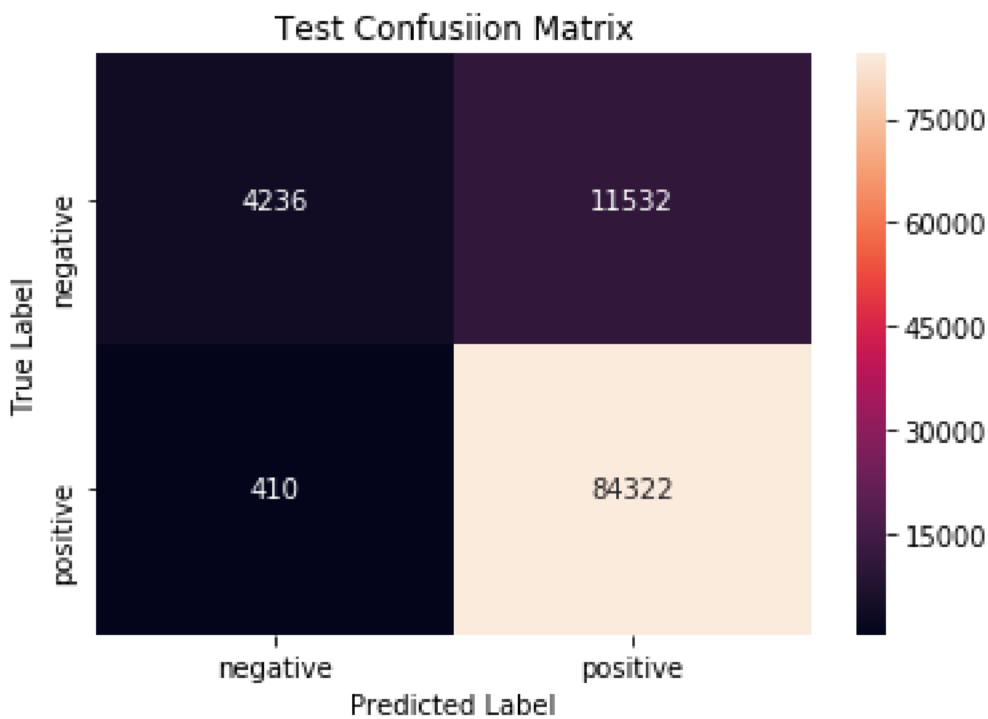
In [30]:
```
trainconfusionmatrix(model_New_Bow,X_Train_New,Y_train)
```

```
Confusion Matrix for Train set
```

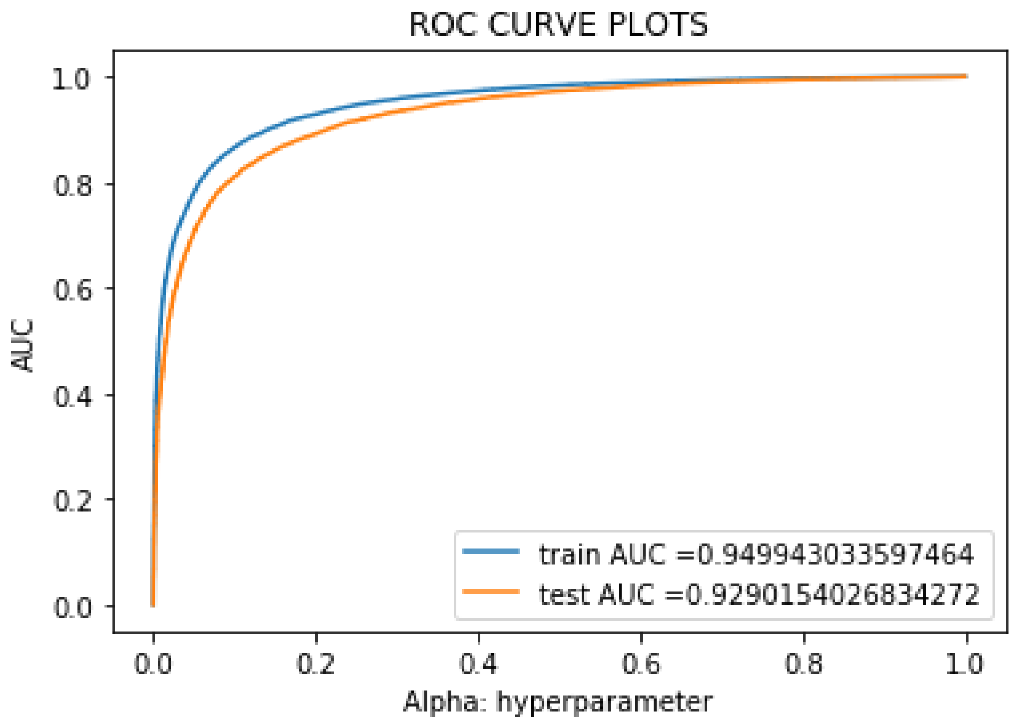


```
In [31]: testconfusionmatrix(model_New_Bow,X_Train_New,Y_train)
```

```
Confusion Matrix for Test set
```




```
In [32]: plot_auc_roc(model_New_Bow,X_Train_New,X_Test_New,Y_train,Y_test)
```




```
In [33]: print("Classification Report: \n")
         y_pred=model_New_Bow.predict(X_Test_New)

         print(classification_report(Y_test, y_pred))
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.88      0.20      0.33      7793
           1       0.87      0.99      0.93     41707

   micro avg       0.87      0.87      0.87     49500
   macro avg       0.88      0.60      0.63     49500
weighted avg       0.87      0.87      0.83     49500
```

```
In [37]: show_30_informative_feature(vectorizer,model_New_Bow)
```

```
S.N                 Positive                                                        Negative
------------------------------------------------------------------------------------------
1.                 -9.346        amazon                                                -9.346        ama
2.                 -10.598        best                                                  -9.371        ba
```

```
3.                 -8.908        buy                                          -9.238        bo
4.                 -8.942        coffe                                        -8.908        bu
5.                 -9.124        dont                                         -8.942        co
6.                 -9.384        eat                                          -9.453        di
7.                 -9.974        find                                         -9.124        do
8.                 -8.649        flavor                                       -9.384        ea
9.                 -9.312        food                                         -9.299        eve
10.                 -9.031        get                                          -8.649        fl
11.                 -8.871        good                                         -9.312        fo
12.                 -9.765        great                                        -9.031        ge
13.                 -8.148        like                                         -8.871        go
14.                 -9.913        littl                                        -8.148        li
15.                 -9.423        love                                         -9.468        lo
16.                 -9.437        make                                         -9.423        lo
17.                 -9.363        much                                         -9.437        ma
18.                 -8.635        one                                          -9.363        mu
19.                 -8.943        order                                        -8.635        or
20.                 -9.581        price                                        -8.943        o
21.                 -8.261        product                                      -9.479        pa
22.                 -9.392        realli                                       -8.261        p
23.                 -9.897        store                                        -9.490        pu
24.                 -8.042        tast                                         -9.392        re
25.                 -9.262        tea                                          -8.042        ta
26.                 -9.397        time                                         -9.262        te
27.                 -8.790        tri                                          -9.397        ti
28.                 -8.961        use                                          -8.790        t
29.                 -9.922        well                                         -8.961        us
30.                 -8.746        would                                        -8.746        wo
```

1.14 TF-IDF

```
In [30]: X_Train_New_Tf=X_Train_Tfidf.todense()
```

```
In [31]: X_Train_New_Tf1=np.append(X_Train_New_Tf,Train_len,axis=1)
```

```
In [32]: from scipy.sparse import csr_matrix
         X_Train_New_Tf1=csr_matrix(X_Train_New_Tf1)
```

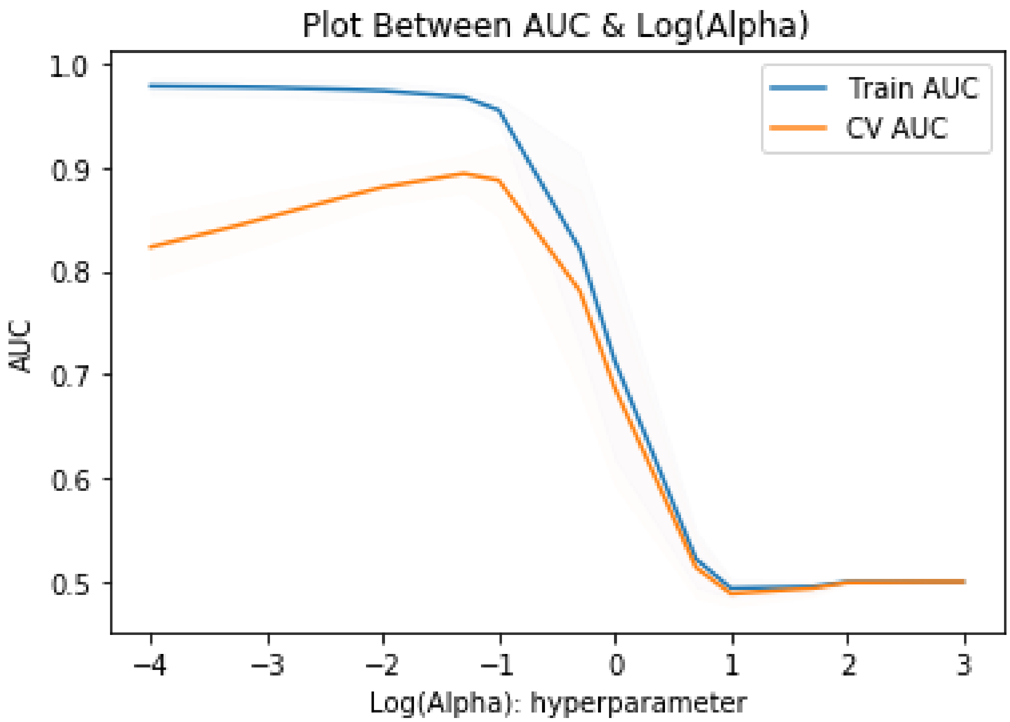
1.14.1 Finding the best value of hyperparameter (Alpha)

```
In [33]: Alpha={'alpha':[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]}
         gsv2=Grid_SearchCV(X_Train_New_Tf1,Y_train,Alpha)

         print("Best HyperParameter: ",gsv2.best_params_)
         print("Best Accuracy: %.2f%%"%(gsv2.best_score_*100))
```

```
Fitting 10 folds for each of 15 candidates, totalling 150 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 150 out of 150 | elapsed:   24.7s finished


Best HyperParameter:  {'alpha': 0.05}
Best Accuracy: 89.40%
```

```
In [34]: plot(Alpha['alpha'],gsv2)
```




```
In [35]: X_Test_New_Tf1=X_Test_Tfidf.todense()

In [36]: X_Test_New_Tf1=np.append(X_Test_New_Tf1,Test_len,axis=1)

In [37]: from scipy.sparse import csr_matrix
         X_Test_New_Tf1=csr_matrix(X_Test_New_Tf1)
```

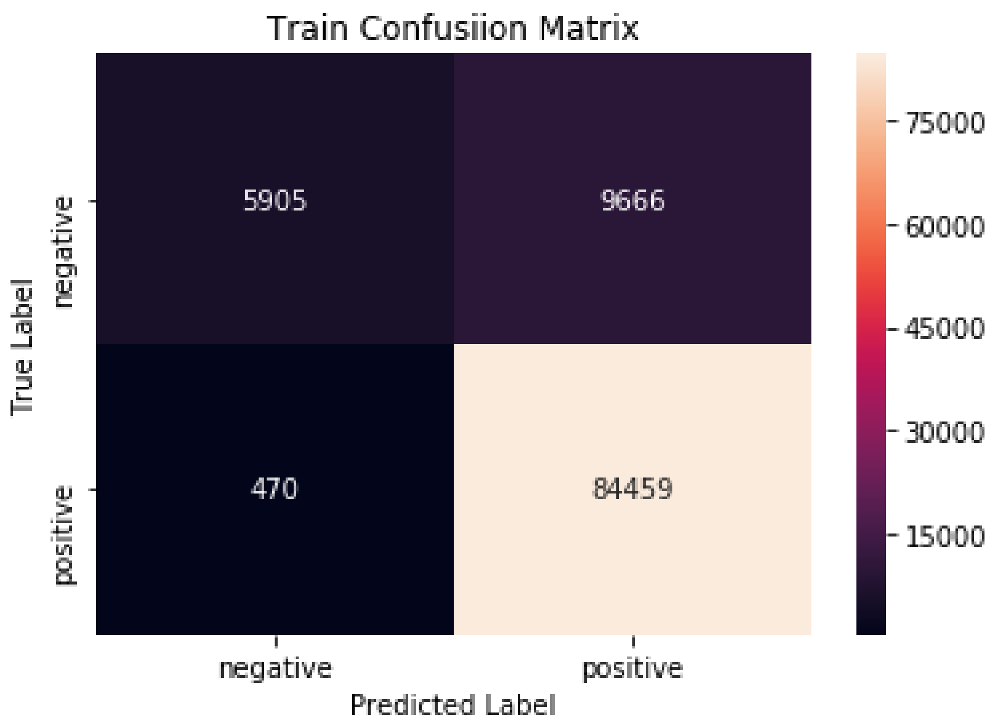
1.14.2 Training the model

```
In [38]: model_New_Tfidf1=MultinomialNB(alpha=gsv2.best_params_['alpha'])
         model_New_Tfidf1.fit(X_Train_New_Tf1,Y_train)

Out[38]: MultinomialNB(alpha=0.05, class_prior=None, fit_prior=True)
```

1.14.3 Evaluating the performance of model

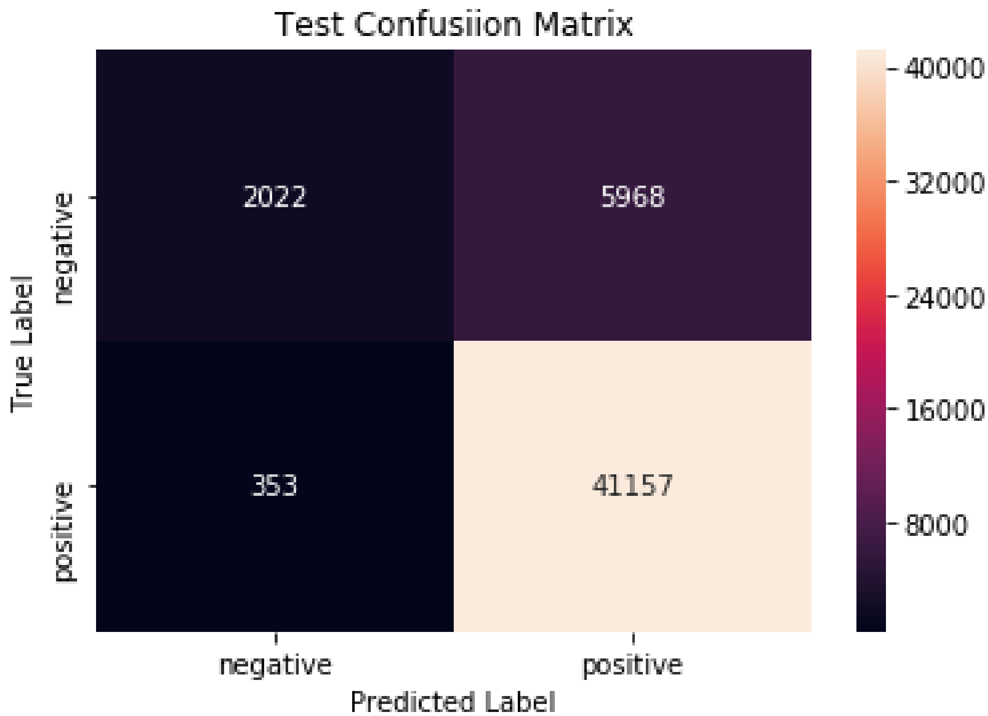
In [39]: trainconfusionmatrix(model_New_Tfidf1,X_Train_New_Tf1,Y_train)

```
Confusion Matrix for Train set
```

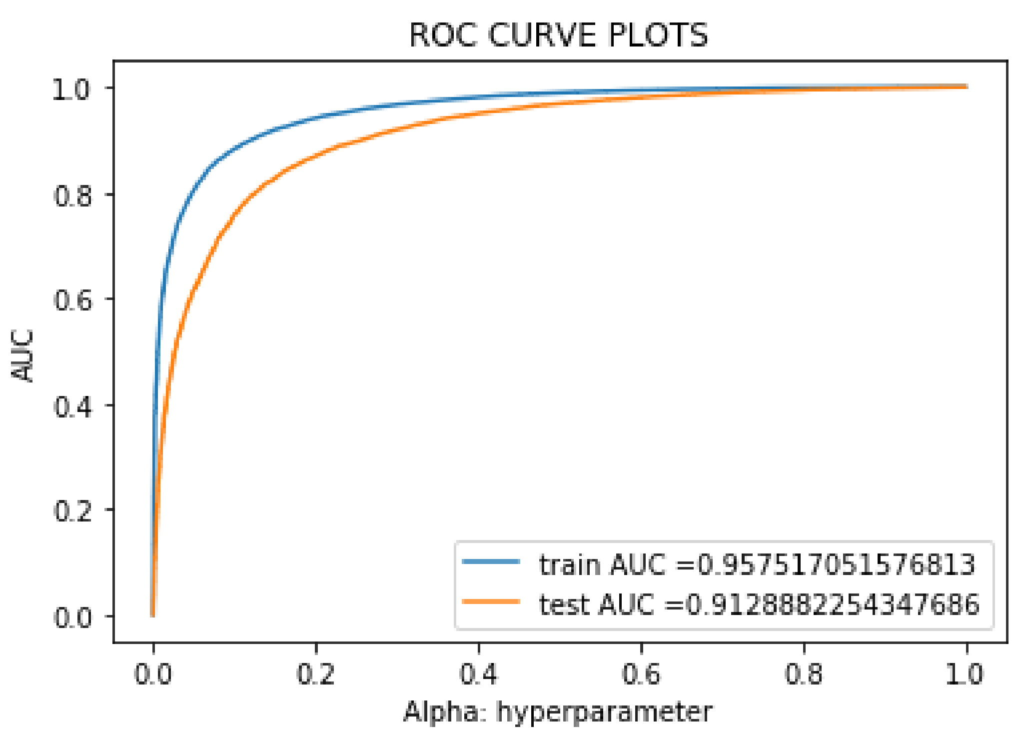


In [40]: testconfusionmatrix(model_New_Tfidf1,X_Test_New_Tf1,Y_test)

```
Confusion Matrix for Test set
```



In [41]: `plot_auc_roc(model_New_Tfidf1,X_Train_New_Tf1,X_Test_New_Tf1,Y_train,Y_test)`



```
In [42]: print("Classification Report: \n")
         y_pred=model_New_Tfidf1.predict(X_Test_New_Tf1)

         print(classification_report(Y_test, y_pred))
```

```
Classification Report:

              precision    recall  f1-score   support

           0       0.85      0.25      0.39      7990
           1       0.87      0.99      0.93     41510

   micro avg       0.87      0.87      0.87     49500
   macro avg       0.86      0.62      0.66     49500
weighted avg       0.87      0.87      0.84     49500
```

1.14.4 Displaying 30 most informative feature

```
In [67]: show_30_informative_feature(vectorizer_tfidf,model_New_Tfidf1)
```

```
S.N                   Positive                                                       Negative

------------------------------------------------------------------------------------------
1.                -9.807        amazon                                             -9.807        ama
2.                -11.064        best                                               -9.755        ba
3.                -9.434        buy                                                -9.760        bag
4.                -9.306        coffe                                              -9.856        bou
5.                -9.845        dog                                                -9.547        bo
6.                -10.127        drink                                              -9.434        bu
7.                -9.870        eat                                                -9.306        co
8.                -10.467        find                                               -9.562        di
9.                -9.313        flavor                                             -9.845        dog
10.                -9.732        food                                               -9.616        do
11.                -9.654        get                                                -9.870        ea
12.                -9.634        good                                               -9.717        ev
13.                -10.489        great                                              -9.313        f
14.                -8.958        like                                               -9.732        fo
15.                -10.394        littl                                              -9.654        g
16.                -10.152        love                                               -9.634        g
17.                -10.050        make                                               -8.958        l
18.                -9.850        much                                               -9.858        lo
19.                -9.333        one                                                -9.872        mo
20.                -9.433        order                                              -9.850        mu
21.                -10.043        price                                              -9.333        o
22.                -8.953        product                                            -9.433        o
```

```
23.                  -9.885        realli                                        -9.819        pa
24.                  -10.292        store                                         -8.953        p
25.                  -8.821        tast                                          -9.843        pu
26.                  -9.633        tea                                           -8.821        ta
27.                  -9.904        time                                          -9.633        te
28.                  -9.433        tri                                           -9.433        tr
29.                  -9.659        use                                           -9.659        us
30.                  -9.295        would                                         -9.295        wo
```

2 Conclusion :

1.Report On Different Vectorizer Method

```
In [1]: from prettytable import PrettyTable

        x = PrettyTable()

        x.field_names = ["Vectorizer", "Hyperparameter(Alpha)", "Train AUC","Test AUC","F1-Sco

        x.add_row(["BOW",0.05,0.95,0.93,0.83])
        x.add_row(["TF-IDF",0.1,0.95,0.91,0.84])

        print(x)
```

```
+------------+-----------------------+-----------+----------+----------+
| Vectorizer | Hyperparameter(Alpha) | Train AUC | Test AUC | F1-Score |
+------------+-----------------------+-----------+----------+----------+
|    BOW     |          0.05         |    0.95   |   0.93   |   0.83   |
|   TF-IDF   |          0.1          |    0.95   |   0.91   |   0.84   |
+------------+-----------------------+-----------+----------+----------+
```

2.Report On Different Vectorizer Method After Addition Of Length as another Column

```
In [2]: from prettytable import PrettyTable

        x = PrettyTable()

        x.field_names = ["Vectorizer", "Hyperparameter(Alpha)", "Train AUC","Test AUC","F1-Sco

        x.add_row(["BOW",0.05,0.94,0.92,0.83])
        x.add_row(["TF-IDF",0.05,0.95,0.91,0.84])

        print(x)
```

```
+------------+-----------------------+-----------+----------+----------+
| Vectorizer | Hyperparameter(Alpha) | Train AUC | Test AUC | F1-Score |
```

BOW	0.05	0.94	0.92	0.83
TF-IDF	0.05	0.95	0.91	0.84

3. I have taken considerable amount of data but it did not take long time in execution .

4. Since data is unbalanced , i did time based splitting and used roc_auc metric as scoring parameter in GridsearchCV .

5. After adding Length as another column , there is no any improvement.