Part 4 - Naive Bayes

By Aziz Presswala

In [1]:
```
#importing libraries
import math
import numpy as np
import pandas as pd
import seaborn as sn
import sqlite3
import matplotlib.pyplot as plt
from prettytable import PrettyTable

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score, auc, roc_curve
from sklearn import model_selection
```

In [2]:
```
# Using the CleanedText column saved in final.sqlite db
con = sqlite3.connect('final.sqlite')
filtered_data = pd.read_sql_query("SELECT * FROM Reviews", con)
filtered_data.shape
```

Out[2]:
```
(364171, 12)
```

In [3]:
```
# replacing all the 'positive' values of the Score attribute with 1
filtered_data['Score']=filtered_data['Score'].replace('positive',1)
```

```
In [4]: # replacing all the 'neagtive' values of the Score attribute with 0
        filtered_data['Score']=filtered_data['Score'].replace('negative',0)
```

```
In [5]: #randomly selecting 100k points from the dataset
        df=filtered_data.sample(100000)
```

```
In [6]: #sort the dataset by timestamp
        df = df.sort_values('Time')
        #splitting the dataset into train(70%) & test(30%)
        train_data = df[0:70000]
        test_data = df[70000:100000]
```

Featurization

BoW

```
In [7]: #applying fit transform on train datasset
        count_vect = CountVectorizer(min_df=10)
        x_train_bow = count_vect.fit_transform(train_data['CleanedText'].values
        )
        x_train_bow.shape
```

```
Out[7]: (70000, 7180)
```

```
In [8]: #applying transform on test dataset
        x_test_bow = count_vect.transform(test_data['CleanedText'].values)
        x_test_bow.shape
```

```
Out[8]: (30000, 7180)
```

```
In [9]: y_train_bow = train_data['Score']
        y_test_bow = test_data['Score']
```

TF-IDF

In [10]:
```
#applying fit transform on train datasset
tf_idf_vect = TfidfVectorizer(min_df=10)
x_train_tfidf = tf_idf_vect.fit_transform(train_data['CleanedText'].val
ues)
x_train_tfidf.shape
```

Out[10]:
```
(70000, 7180)
```

In [11]:
```
#applying transform on test dataset
x_test_tfidf = tf_idf_vect.transform(test_data['CleanedText'].values)
x_test_tfidf.shape
```

Out[11]:
```
(30000, 7180)
```

In [12]:
```
y_train_tfidf = train_data['Score']
y_test_tfidf = test_data['Score']
```

Applying Multinomial Naive Bayes

Applying Naive Bayes on BOW

GridSearchCV

In [13]:
```
# initializing Mutinomial Naive Bayes model
nb = MultinomialNB()

# alpha values we need to try on classifier
alpha_values = [10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10
**3,10**4]
param_grid = {'alpha':[10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,1
0**2,10**3,10**4]}
```

```
# using GridSearchCV to find the optimal value of alpha
# using roc_auc as the scoring parameter & applying 10 fold CV
gscv = GridSearchCV(nb,param_grid,scoring='roc_auc',cv=10,return_train_
score=True)

gscv.fit(x_train_bow,y_train_bow)

print("Best Alpha Value: ",gscv.best_params_)
print("Best ROC AUC Score: %.5f"%(gscv.best_score_))
```

```
Best Alpha Value:  {'alpha': 1}
Best ROC AUC Score: 0.91149
```

In [14]:

```
# determining optimal alpha
optimal_alpha = gscv.best_params_['alpha']

#training the model using the optimal alpha
nbf = MultinomialNB(alpha=optimal_alpha)
nbf.fit(x_train_bow,y_train_bow)

#predicting the class label using test data
y_pred = nbf.predict_proba(x_test_bow)[:,1]

#determining the Test roc_auc_score for optimal alpha
auc_score = roc_auc_score(y_test_bow, y_pred)
print('\n**** Test roc_auc_score for alpha = %f is %f ****' % (optimal_
alpha,auc_score))
```

```
**** Test roc_auc_score for alpha = 1.000000 is 0.912246 ****
```

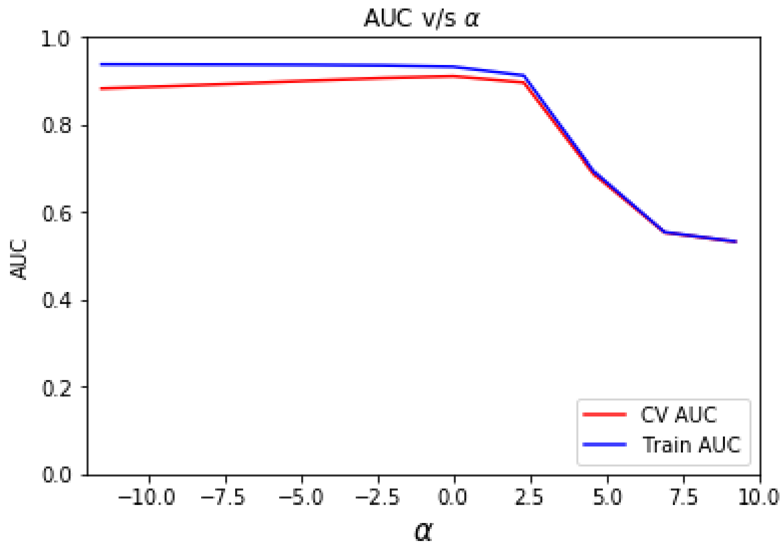
AUC vs alpha plot

In [15]:

```
# plotting AUC vs alpha on Train & Validation dataset
log_alpha=[math.log(x) for x in alpha_values]
print(log_alpha)
plt.xlim(-12,10)
plt.ylim(0.0,1.0)
plt.xlabel(r"$\alpha$",fontsize=15)
```

```
plt.ylabel('AUC')
plt.title(r'AUC v/s $\alpha$')
plt.plot(log_alpha, gscv.cv_results_['mean_test_score'], 'r', label='CV
 AUC')
plt.plot(log_alpha, gscv.cv_results_['mean_train_score'], 'b', label='T
rain AUC')
plt.legend(loc='lower right')
plt.show()
```

```
[-11.512925464970229, -9.210340371976182, -6.907755278982137, -4.605170
185988091, -2.3025850929940455, 0.0, 2.302585092994046, 4.6051701859880
92, 6.907755278982137, 9.210340371976184]
```



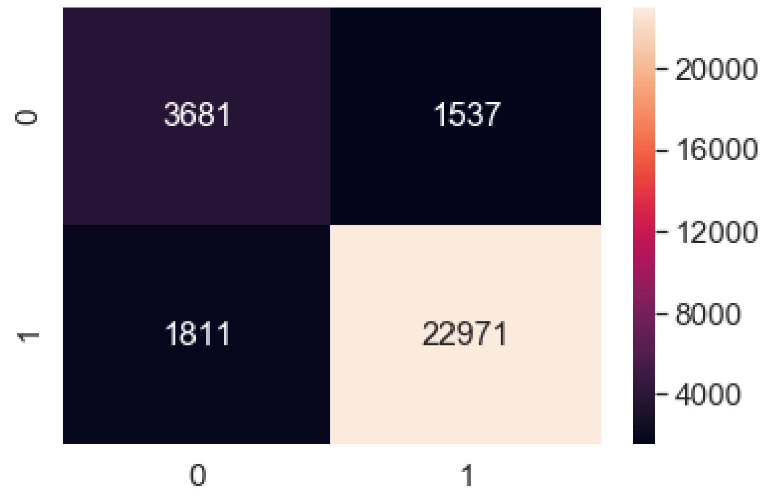
Confusion Matrix

In [16]:
```
#plotting confusion matrix as heatmap
pred = nbf.predict(x_test_bow)
cm = confusion_matrix(y_test_bow, pred)
print(cm)
df_cm = pd.DataFrame(cm, range(2),range(2))
```

```
sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
[[ 3681  1537]
 [ 1811 22971]]
```

Out[16]: `<matplotlib.axes._subplots.AxesSubplot at 0x70ca13fb38>`



Top 10 important features of positive & negative class

In [21]:
```
feature_names = count_vect.get_feature_names()
max_ind_pos=np.argsort((nbf.feature_log_prob_)[1])[::-1][0:10]
max_ind_neg=np.argsort((nbf.feature_log_prob_)[0])[::-1][0:10]
print(max_ind_pos,max_ind_neg)
coefs_with_fns1 = sorted(zip(max_ind_pos, feature_names))
coefs_with_fns2 = sorted(zip(max_ind_neg, feature_names))
top = zip(coefs_with_fns1, coefs_with_fns2)
print("\t\tPositive\t\t\t\tNegative")
print("________________________________________________________________________________________________________________")
for (coef_1, fn_1), (coef_2, fn_2) in top:
```

```
    print("\t%.4f\t%-15s\t\t\t\t%.4f\t%-15s" % (coef_1, fn_1, coef_2, f
n_2))
```

```
[3600 6272 2712 2389 3688 2774 6748 4342 6288 4872] [6272 3600 4872 434
2 2389 7071 6550 6748 2712 1224]
                Positive                                        Negative
_______________________________________________________________________

___________________________
        2389.0000       abil                                    1224.00
00      absolutley
        2712.0000       abdomin                                 2389.00
00      abl
        2774.0000       abroad                                  2712.00
00      absolut
        3600.0000       aback                                   3600.00
00      abandon
        3688.0000       abl                                     4342.00
00      abil
        4342.0000       absent                                  4872.00
00      abdomin
        4872.0000       absolutley                              6272.00
00      aback
        6272.0000       abandon                                 6550.00
00      absenc
        6288.0000       absolut                                 6748.00
00      absent
        6748.0000       absenc                                  7071.00
00      abroad
```

In [51]:
```
feature_names = count_vect.get_feature_names()
coefs_with_fns = sorted(zip(nbf.coef_[0], feature_names),reverse=True)
top = zip(coefs_with_fns[:10], coefs_with_fns[:-(10 + 1):-1])
print("\t\tPositive\t\t\t\tNegative")
print("_______________________________________________________________
___________________________")
for (coef_1, fn_1), (coef_2, fn_2) in top:
    print("\t%.4f\t%-15s\t\t\t\t%.4f\t%-15s" % (coef_1, fn_1, coef_2, f
n_2))
```

```
                Positive                                        Negative
```

```
____________________________________________________________________________
_____________________________
          -4.4269 like                                              -13.9164
misrepresent
          -4.4929 tast                                              -13.5109
fraud
          -4.6308 good                                              -13.2232
emptor
          -4.6471 flavor                                            -13.2232
letdown
          -4.6778 love                                              -13.2232
pawn
          -4.7085 great                                             -13.2232
rubbish
          -4.7177 use                                               -13.2232
unidentifi
          -4.7642 one                                               -13.2232
unsaf
          -4.8520 tea                                               -13.0001
acesulfam
          -4.8555 product                                           -13.0001
heed
```

ROC Curve

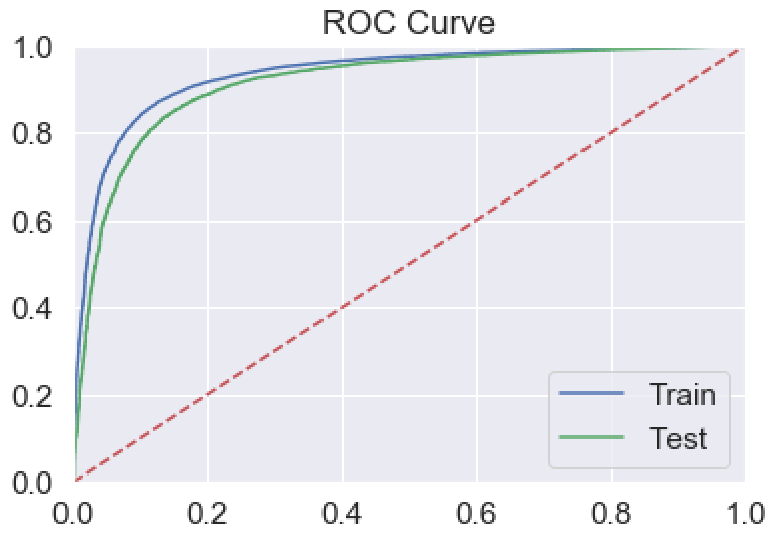
In [52]:

```
# Plotting roc curve on Train Data
prob_train = nbf.predict_proba(x_train_bow)[:,1]
fpr, tpr, threshold = roc_curve(y_train_bow, prob_train)
plt.plot(fpr, tpr, 'b', label='Train')

# Plotting roc curve on Test Data
prob_test = nbf.predict_proba(x_test_bow)[:,1]
fpr, tpr, threshold = roc_curve(y_test_bow, prob_test)
plt.plot(fpr, tpr, 'g', label='Test')

plt.title('ROC Curve')
plt.plot([0, 1], [0, 1],'r--')
```

```
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```



Applying Naive Bayes on TFIDF

GridSearchCV

In [53]:
```
# initializing Mutinomial Naive Bayes model
nb = MultinomialNB()

# alpha values we need to try on classifier
alpha_values = [10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]
param_grid = {'alpha':[10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]}

# using GridSearchCV to find the optimal value of alpha
# using roc_auc as the scoring parameter & applying 10 fold CV
```

```
gst = GridSearchCV(nb,param_grid,scoring='roc_auc',cv=10,return_train_s
core=True)

gst.fit(x_train_tfidf,y_train_tfidf)

print("Best Alpha Value: ",gst.best_params_)
print("Best ROC AUC Score: %.5f"%(gst.best_score_))
```

```
Best Alpha Value:  {'alpha': 0.1}
Best ROC AUC Score: 0.92205
```

In [54]:

```
# determining optimal alpha
optimal_alpha = gst.best_params_['alpha']

#training the model using the optimal alpha
nbf = MultinomialNB(alpha=optimal_alpha)
nbf.fit(x_train_tfidf,y_train_tfidf)

#predicting the class label using test data
y_pred = nbf.predict_proba(x_test_tfidf)[:,1]

#determining the Test roc_auc_score for optimal alpha
auc_score = roc_auc_score(y_test_tfidf, y_pred)
print('\n**** Test roc_auc_score for alpha = %f is %f ****' % (optimal_
alpha,auc_score))
```

```
**** Test roc_auc_score for alpha = 0.100000 is 0.919208 ****
```

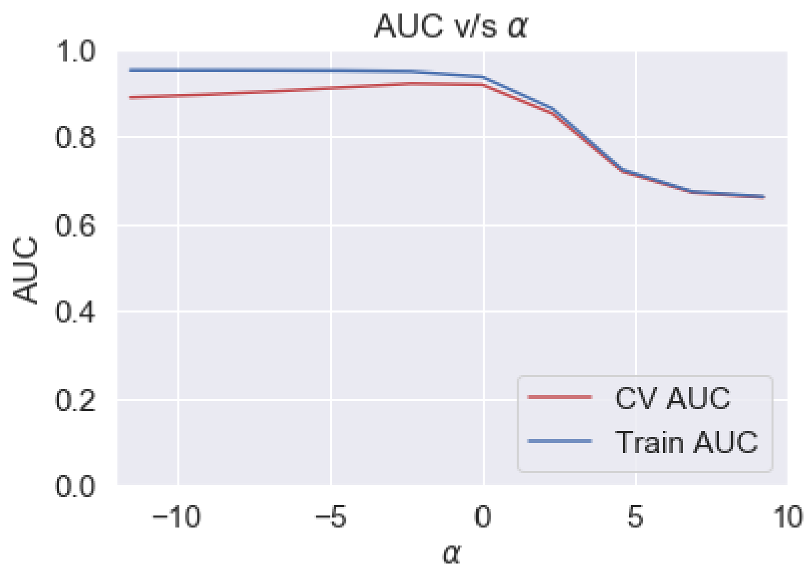
AUC vs alpha plot

In [55]:

```
# plotting AUC vs alpha on Train & Validation dataset
log_alpha=[math.log(x) for x in alpha_values]
print(log_alpha)
plt.xlim(-12,10)
plt.ylim(0.0,1.0)
plt.xlabel(r"$\alpha$",fontsize=15)
plt.ylabel('AUC')
plt.title(r'AUC v/s $\alpha$')
```

```
plt.plot(log_alpha, gst.cv_results_['mean_test_score'], 'r', label='CV
 AUC')
plt.plot(log_alpha, gst.cv_results_['mean_train_score'], 'b', label='Tr
ain AUC')
plt.legend(loc='lower right')
plt.show()
```

```
[-11.512925464970229, -9.210340371976182, -6.907755278982137, -4.605170
185988091, -2.3025850929940455, 0.0, 2.302585092994046, 4.6051701859880
92, 6.907755278982137, 9.210340371976184]
```



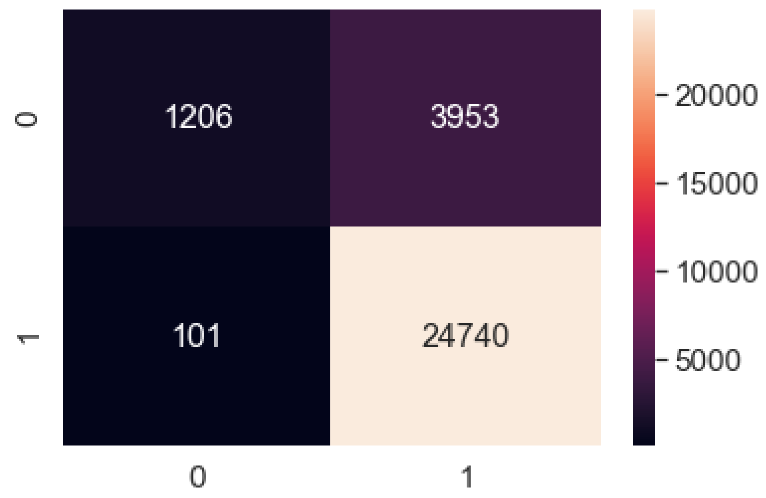
Confusion Matrix

In [56]:
```
#plotting confusion matrix as heatmap
pred = nbf.predict(x_test_tfidf)
cm = confusion_matrix(y_test_tfidf, pred)
print(cm)
df_cm = pd.DataFrame(cm, range(2),range(2))
sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
[[ 1206  3953]
 [  101 24740]]
```

Out[56]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x51267d7898>
```



Top 10 important features of positive & negative class

In [57]:

```python
# Please write all the code with proper documentation
feature_names = tf_idf_vect.get_feature_names()
coefs_with_fns = sorted(zip(nbf.coef_[0], feature_names),reverse=True)
top = zip(coefs_with_fns[:10], coefs_with_fns[:-(10 + 1):-1])
print("\t\tPositive\t\t\t\tNegative")
print("________________________________________________________________________________________________________")
for (coef_1, fn_1), (coef_2, fn_2) in top:
    print("\t%.4f\t%-15s\t\t\t\t%.4f\t%-15s" % (coef_1, fn_1, coef_2, f
n_2))
```

```
                Positive                                        Negative
________________________________________________________________________
___________________________
        -5.0301 great                                           -14.1393
misrepresent
```

```
misrepresent
         -5.0355 love                                             -13.4209
fraud
         -5.1026 good                                             -13.2250
emptor
         -5.1037 tast                                             -13.1347
acesulfam
         -5.1274 like                                             -13.0502
improp
         -5.1321 tea                                              -12.9497
unsaf
         -5.1764 flavor                                           -12.9250
unwil
         -5.2095 coffe                                            -12.8997
monsanto
         -5.2718 product                                          -12.8976
rude
         -5.2896 use                                              -12.8954
letdown
```

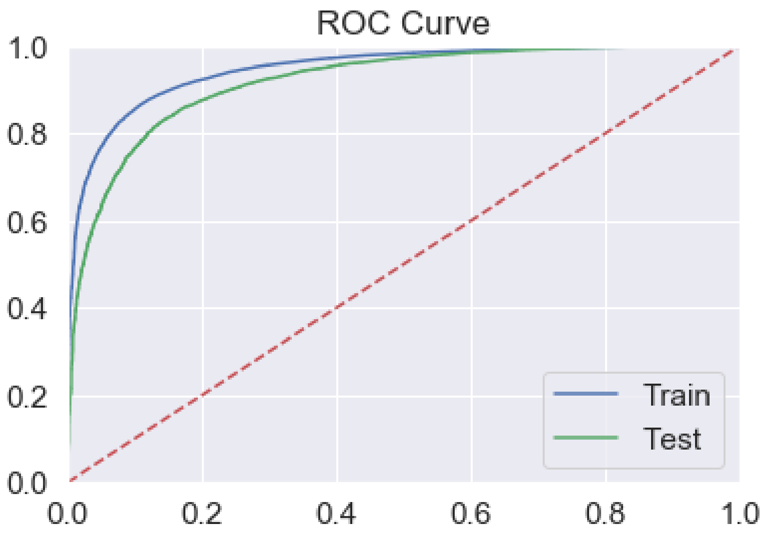
ROC Curve

In [45]:

```
# Plotting roc curve on Train Data
prob_train = nbf.predict_proba(x_train_tfidf)[:,1]
fpr, tpr, threshold = roc_curve(y_train_tfidf, prob_train)
plt.plot(fpr, tpr, 'b', label='Train')

# Plotting roc curve on Test Data
prob_test = nbf.predict_proba(x_test_tfidf)[:,1]
fpr, tpr, threshold = roc_curve(y_test_tfidf, prob_test)
plt.plot(fpr, tpr, 'g', label='Test')

plt.title('ROC Curve')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```



Conclusion

In [46]:
```
# Summarizing the observations
x=PrettyTable()
x.field_names = ['Vectorizer','Alpha','AUC']
x.add_row(['BOW','1.0','0.914005'])
x.add_row(['Tfidf','0.1','0.919208'])
print(x)
```

```
+------------+-------+----------+
| Vectorizer | Alpha |   AUC    |
+------------+-------+----------+
|    BOW     |  1.0  | 0.914005 |
|   Tfidf    |  0.1  | 0.919208 |
+------------+-------+----------+
```

Conclusions:-

1. Training Time of Naive Bayes is significantly less as compared to other classification algorithms (such as KNN).

2. Performance of Naive Bayes is almost similar when trained using Bag of Words & TF-IDF (based on AUC scores).