AMAZON NAIVE BAYES ASSIGNMENT

January 22, 2019

1 Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454 Number of users: 256,059 Number of products: 74,258 Timespan: Oct 1999 - Oct 2012 Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
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 between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective: Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative? [Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

1.1 [1]. Reading Data

2 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

```
In [1]: import sqlite3
        import pandas as pd
        import numpy as np
        import nltk
        import string
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.feature_extraction.text import TfidfTransformer
        from sklearn.feature_extraction.text import TfidfVectorizer


        from sklearn.model_selection import train_test_split
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.metrics import accuracy_score
        from sklearn.model_selection import cross_val_score
        from collections import Counter
        from sklearn.preprocessing import StandardScaler

        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.metrics import confusion_matrix
        from sklearn import metrics
        from sklearn.metrics import roc_curve, auc

        import re
        # Tutorial about Python regular expressions: https://pymotw.com/2/re/
        import string
        from nltk.corpus import stopwords
        from nltk.stem import SnowballStemmer
        from nltk.stem.wordnet import WordNetLemmatizer

        import pickle
```

```
In [1]: data = pd.read_csv('Reviews.csv')
        print (data.head(2))
        print(data.shape)
```

```
   Id   ProductId          UserId ProfileName  HelpfulnessNumerator  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW  delmartian                     1
1   2  B00813GRG4  A1D87F6ZCVE5NK      dll pa                     0

   HelpfulnessDenominator  Score        Time               Summary  \
0                       1      5  1303862400  Good Quality Dog Food
1                       0      1  1346976000      Not as Advertised
```

```
                                                Text
0  I have bought several of the Vitality canned d...
1  Product arrived labeled as Jumbo Salted Peanut...
(568454, 10)
```

2.1 [2] Data Cleaning: Deduplication and Nan features

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [2]:
```
#checking for Nan values in data. True indicates Nan values are present along the colu
data.isnull().any()
```

Out[2]:
```
Id                        False
ProductId                 False
UserId                    False
ProfileName                True
HelpfulnessNumerator      False
HelpfulnessDenominator    False
Score                     False
Time                      False
Summary                    True
Text                      False
dtype: bool
```

In [3]:
```
# checking for Nan values along 'profilename' column
#data[data['ProfileName'].isnull()].head(2)
```

In [4]:
```
# checking for Nan values along 'summary' column
#data[data['Summary'].isnull()]
```

In [5]:
```
#Dropping Nan values
data = data.dropna()
```

In [6]:
```
#printing shape of data after dropping Nan values
print (data.shape)
```

```
(568411, 10)
```

In [7]:
```
#Review score should lie between 1 to 5
#Returns True if all the scores lie between 1 to 5(inclusive)
list1 = data['Score'].map(lambda x: True if x in [1,2,3,4,5] else False)
list1.all()
```

Out[7]:
```
True
```

```
In [8]: filtered_data = data.loc[data['Score']!=3].copy()
        print (filtered_data.head(2))
        print (filtered_data.shape)
```

```
   Id   ProductId          UserId ProfileName  HelpfulnessNumerator  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW  delmartian                     1
1   2  B00813GRG4  A1D87F6ZCVE5NK      dll pa                     0

   HelpfulnessDenominator  Score        Time                Summary  \
0                       1      5  1303862400  Good Quality Dog Food
1                       0      1  1346976000      Not as Advertised

                                                Text
0  I have bought several of the Vitality canned d...
1  Product arrived labeled as Jumbo Salted Peanut...
(525773, 10)
```

```
In [9]: #mapping positive(>3) and negative(<3) reviews based on scores of the data.
        import pandas as pd
        pos_negative = filtered_data['Score'].map(lambda x: 1 if int (x)>3  else 0)
        filtered_data['Score'] = pos_negative
        print ('shape of filtered_data')
        print (filtered_data.shape)
        #print (filtered_data.head())
```

```
shape of filtered_data
(525773, 10)
```

```
In [10]: #arranging data with increasing productid
         sorted_data = filtered_data.sort_values('ProductId',axis=0,ascending=True,inplace=Fals
```

```
In [11]: #finding the duplicates in our data
         #If the same person gives for the same product at the same time we call it as suplica
         #sorted_data.loc[sorted_data.duplicated(["UserId","ProfileName","Time","Text"],keep =
```

```
In [12]: #counting number of duplicates present in our data
         sorted_data.duplicated(["UserId","ProfileName","Time","Text"]).sum()
```

```
Out[12]: 161612
```

```
In [13]: #dropping all duplicates keeping the first one
         final = sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},kee
         final.shape
```

```
Out[13]: (364161, 10)
```

```
In [14]: #Checking to see how much % of data still remains
         (final['Id'].size*1.0)/(sorted_data['Id'].size*1.0)*100
```

Out[14]: 69.26201992114468

In [15]:
```
#helpfulness numerator denotes number of people who found the review helpful
#helpfulness denominator denotes number of people who indicated whether or not the re
#so, helpfulness numerator should be less than denominator
final = final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [16]:
```
#final shape of data after preprocessing
final.shape
```

Out[16]: (364159, 10)

In [17]:
```
final['Score'].value_counts()
```

Out[17]:
```
1    307054
0     57105
Name: Score, dtype: int64
```

In [18]:
```
final.shape
```

Out[18]: (364159, 10)

In [19]:
```
#arranging data with increasing time
final_data = final.sort_values('Time',axis=0,ascending=True,inplace=False,kind='quicks
```

3 [3] Preprocessing

4 [3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [120]:
```
#final_data = final_data.iloc[0:10000,:].copy()
```

In [20]:
```
stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer
not_words = re.findall(r'\w*n[\'|o]t',str (stop)) #finding NOT words in stop words
```

```
not_words.append('n\'t')
not_words.append('no')
print (not_words)
stop_words = stop - set (not_words)  #removing  NOT words from stop words
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r'\w*n[\'|o]t', "not", phrase)
    # general
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
def cleanhtmlpunc(sentence): #function to clean the word of any html-tags
    clean = re.compile('<.*?>')
    clean = re.sub(clean, ' ', sentence)
    clean = re.sub(r"(http|www)\S+", "", clean)
    clean = re.sub(r"\S+com", "",clean)
    #clean = re.sub(r"\(\w+\)","",clean)
    clean = re.sub(r"\."," ",clean)
    cleaned = re.sub(r'[?+|!+|\'+|"+|#+|:+]',r' ',clean)
    cleantext = re.sub(r'[\.+|,+|)+|(+|\+|/+]',r' ',cleaned)
    return cleantext
#def cleanpunc(sentence): #function to clean the word of any punctuation or special c
   # return  cleaned
print('***********************************')
print(stop_words)
```

```
["couldn't", "haven't", "shan't", "wouldn't", "mightn't", 'not', "don't", "weren't", "hasn't",
***********************************
{"you're", 'wouldn', 'does', 't', 'i', 'd', 'at', 'them', 'didn', 'a', 'when', 'we', "should've
```

In [21]:
```
def cleanedtext(reviews):
    str1=' '
    final_string=[]
    s=''
    for sent in reviews:
        filtered_sentence=[]
        sent=cleanhtmlpunc(decontracted(sent)) # remove HTMl tags
        for w in sent.split():
            for cleaned_words in w.split():
```

```
                    if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                        if((cleaned_words.lower() not in stop_words)):
                            s=(sno.stem(cleaned_words.lower())).encode('utf8')
                            filtered_sentence.append(s)
                        else:
                            continue
                    else:
                        continue
            str1 = b" ".join(filtered_sentence) #final string of cleaned words for review
            final_string.append(str1)
        return final_string
```

In [22]:
```
final_string = cleanedtext(final_data['Text'].values)
```

In [23]:
```
final_data['CleanedText']=final_string #adding a column of CleanedText which displays
print (final_data.shape)
final_data.head(2) #below the processed review can be seen in the CleanedText Column
```

```
(364159, 11)
```

Out[23]:
```
            Id   ProductId          UserId         ProfileName  \
150523  150524  0006641040   ACITT7DI6IDDL     shari zychinski
150500  150501  0006641040  AJ46FKXOVC7NR  Nicholas A Mesiano

        HelpfulnessNumerator  HelpfulnessDenominator  Score       Time  \
150523                     0                       0      1  939340800
150500                     2                       2      1  940809600

                                                   Summary  \
150523                              EVERY book is educational
150500  This whole series is great way to spend time w...

                                                      Text  \
150523  this witty little book makes my son laugh at l...
150500  I can remember seeing the show when it aired o...

                                               CleanedText
150523  b'witti littl book make son laugh loud recit c...
150500  b'rememb see show air televis year ago child s...
```

In [2]:
```
import pickle
with open("final_data.pkl", "rb") as f:
    final_data = pickle.load(f)
```

5 4) FEATURIZATION

In featurization we use BOW and TF-IDF, as the values in naive bayes need to positive and word2vec may contain negative values.

6 4.1) BAG OF WORDS WITH UNI-GRAM

In [3]:
```
# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(final_data['CleanedText'].values, f
                                                                       test_size=0.30, random_sta
```

In [4]:
```
# intializing for bag of words
model= CountVectorizer(dtype=float)
final_counts= model.fit_transform(X_train)
```

In [7]:
```
#model.vocabulary_
```

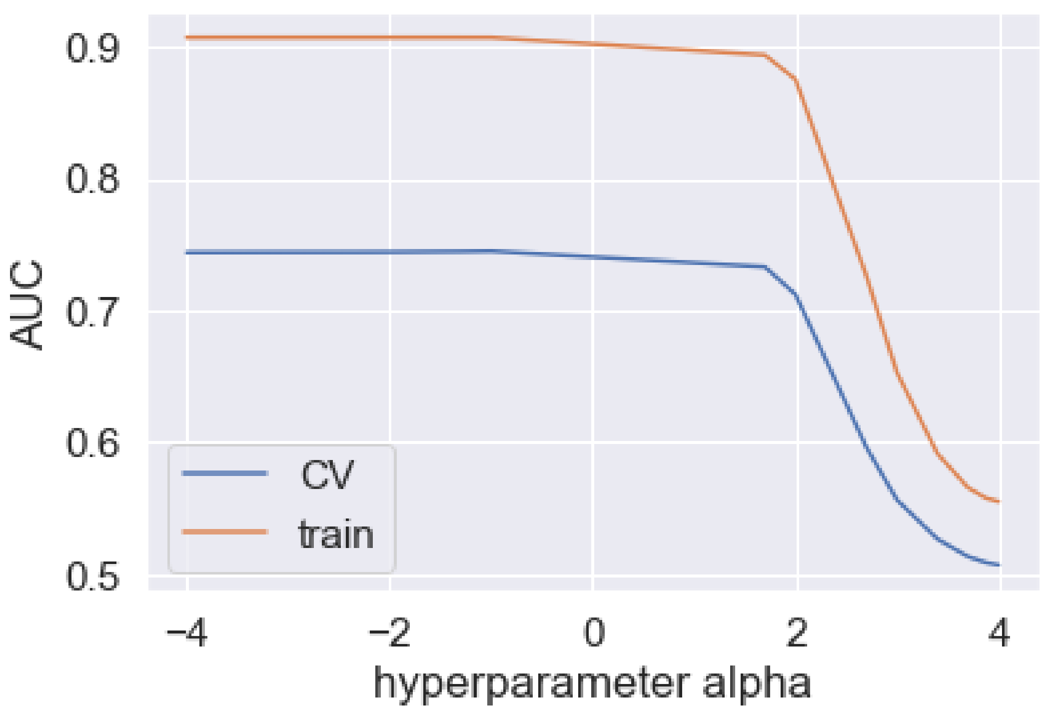
In [5]:
```
#standardizing the bag of words
standardizing  = StandardScaler(with_mean = False)
final_std_data = standardizing.fit_transform(final_counts)
final_std_data.shape
```

Out[5]:
```
(254911, 58339)
```

In [19]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
import math

# creating list for hyperparameter alpha
alpha_values = [0.0001,0.0005,0.001,0.005,0.01,0.05,0.1,50,100,500,1000,2500,5000,7500
# empty list that will hold cv scores

cv_scores = []
train_auc_values = []

# perform 10-fold cross validation
for alpha in alpha_values:
    naive_bayes_model = MultinomialNB(alpha=alpha)
    auc = make_scorer(roc_auc_score)
    auc_scores = cross_val_score(naive_bayes_model, final_std_data, y_train, cv=10, sc
    naive_bayes_model.fit(final_std_data,y_train)
    y_pred_proba = naive_bayes_model.predict_proba(final_std_data)[::,1]
    train_auc = roc_auc_score(y_train, y_pred_proba)
    train_auc_values.append(train_auc)
    cv_scores.append(auc_scores.mean())

print ('train data scores')
print (train_auc_values)
print ('*'*50)
print ('CV scores')
print (cv_scores)
```

```
    # changing to misclassification error

    log = [math.log10(x) for x in alpha_values]
    # plot misclassification error vs alpha
    plt.plot(log, cv_scores,label='CV')
    #plt.label('cv_auc')
    plt.plot(log,train_auc_values,label='train')
    #plt.label('train_auc')
    plt.legend()
    plt.xlabel('hyperparameter alpha')
    plt.ylabel('AUC')
    plt.show()
```

```
train data scores
[0.907019147586722, 0.9070182104209432, 0.9070179750970933, 0.9070162119426658, 0.907013066784
*************************************************
CV scores
[0.7438125188968255, 0.7439476658507528, 0.7440096184118049, 0.744071045557755, 0.744165984047
```



Optimal value of alpha is 0.1

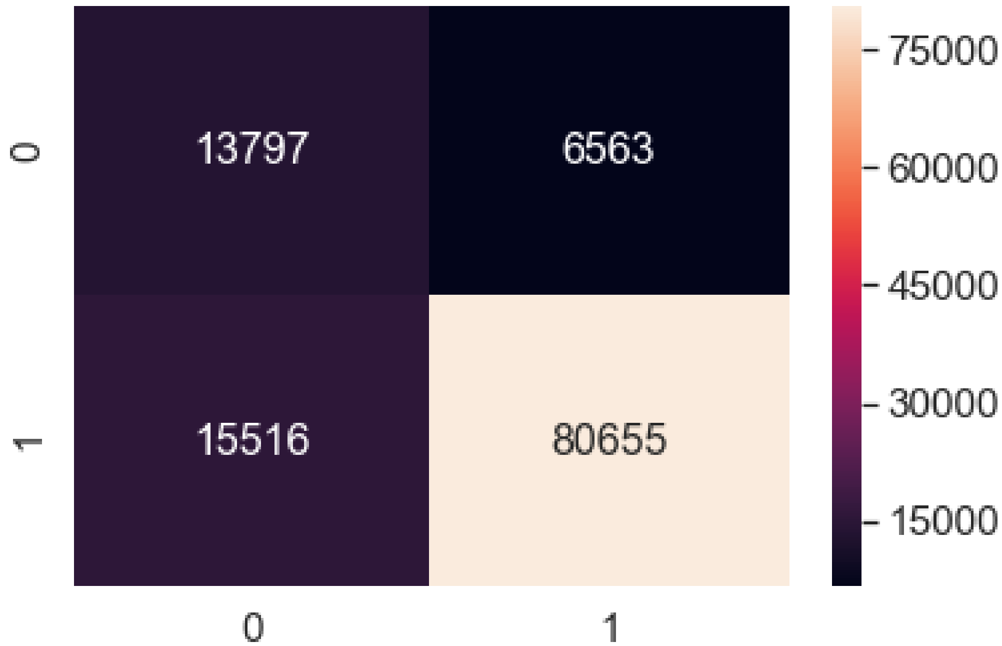
```
In [20]: naive_bayes_model = MultinomialNB(alpha=0.1)
         naive_bayes_model.fit(final_std_data,y_train)
         predictions = naive_bayes_model.predict(standardizing.transform(model.transform(X_tes
```

```
acc = accuracy_score(y_test, predictions) * 100
print ('accuracy_score = {0}'.format(acc))
```

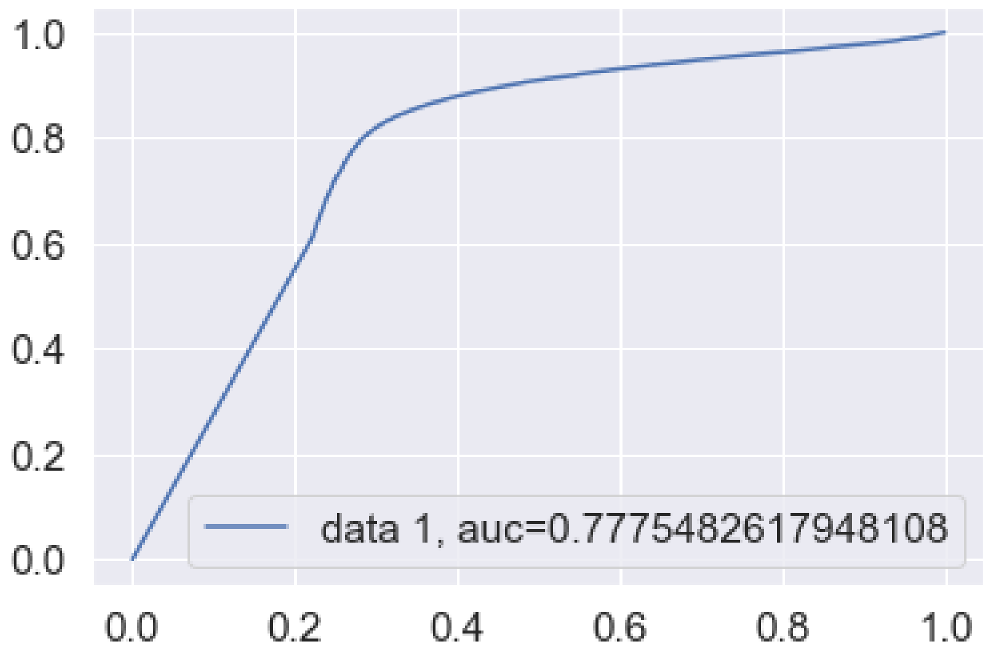
```
accuracy_score = 81.05311033115652
```

In [21]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
result = confusion_matrix(y_test,predictions)
#print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xc5c4a2e860>
```



In [22]:
```
y_pred_proba = naive_bayes_model.predict_proba(standardizing.transform(model.transfor
fpr, tpr, _ = roc_curve(y_test, y_pred_proba,pos_label=1 )
#auc = roc_auc_score(y_test, y_pred_proba)
auc = np.trapz(tpr,fpr)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



7 4.2) TF-IDF WITH UNI-GRAM

```
In [23]: # split the data set into train and test
         tfidf_train, tfidf_test, y_train, y_test = train_test_split(final_data['CleanedText']
                                                                     test_size=0.3, random_sta
```

```
In [24]: vector = TfidfVectorizer(ngram_range = (1,1))
         tf_idf_vector = vector.fit_transform(tfidf_train)
         tf_idf_vector.shape
```

```
Out[24]: (254911, 58339)
```

```
In [25]: standardizing = StandardScaler(with_mean=False)
         tfidf_std_data = standardizing.fit_transform(tf_idf_vector)
         print (tfidf_std_data.shape)
         np.mean(tfidf_std_data)
```

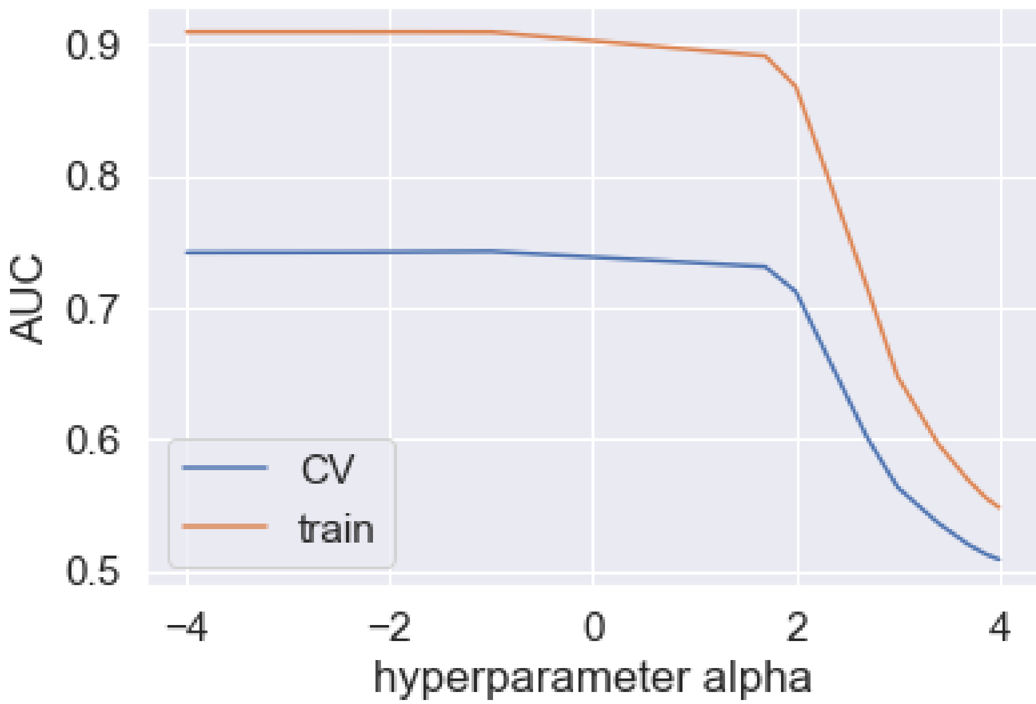
```
(254911, 58339)
```

```
Out[25]: 0.007280159106573833
```

```
In [26]: from sklearn.metrics import f1_score
         from sklearn.metrics import make_scorer
         from sklearn.metrics import roc_curve
```

```python
from sklearn.metrics import roc_auc_score
import math

# creating list for hyperparameter alpha
alpha_values = [0.0001,0.0005,0.001,0.005,0.01,0.05,0.1,50,100,500,1000,2500,5000,7500
# empty list that will hold cv scores

cv_scores = []
train_auc_values = []

# perform 10-fold cross validation
for alpha in alpha_values:
    naive_bayes_model = MultinomialNB(alpha=alpha)
    auc = make_scorer(roc_auc_score)
    auc_scores = cross_val_score(naive_bayes_model, tfidf_std_data, y_train, cv=10, sc
    naive_bayes_model.fit(tfidf_std_data,y_train)
    y_pred_proba = naive_bayes_model.predict_proba(tfidf_std_data)[::,1]
    train_auc = roc_auc_score(y_train, y_pred_proba)
    train_auc_values.append(train_auc)
    cv_scores.append(auc_scores.mean())

print ('train data scores')
print (train_auc_values)
print ('*'*50)
print ('CV scores')
print (cv_scores)

# changing to misclassification error

log = [math.log10(x) for x in alpha_values]
# plot misclassification error vs alpha
plt.plot(log, cv_scores,label='CV')
#plt.label('cv_auc')
plt.plot(log,train_auc_values,label='train')
#plt.label('train_auc')
plt.legend()
plt.xlabel('hyperparameter alpha')
plt.ylabel('AUC')
plt.show()
```

```
train data scores
[0.9091123666803527, 0.9091098834676146, 0.9091080005833618, 0.9090966633286915, 0.90908957723
**************************************************
CV scores
[0.7415988307181505, 0.7417004685958064, 0.7417109426434504, 0.7418387586156369, 0.74200058847
```

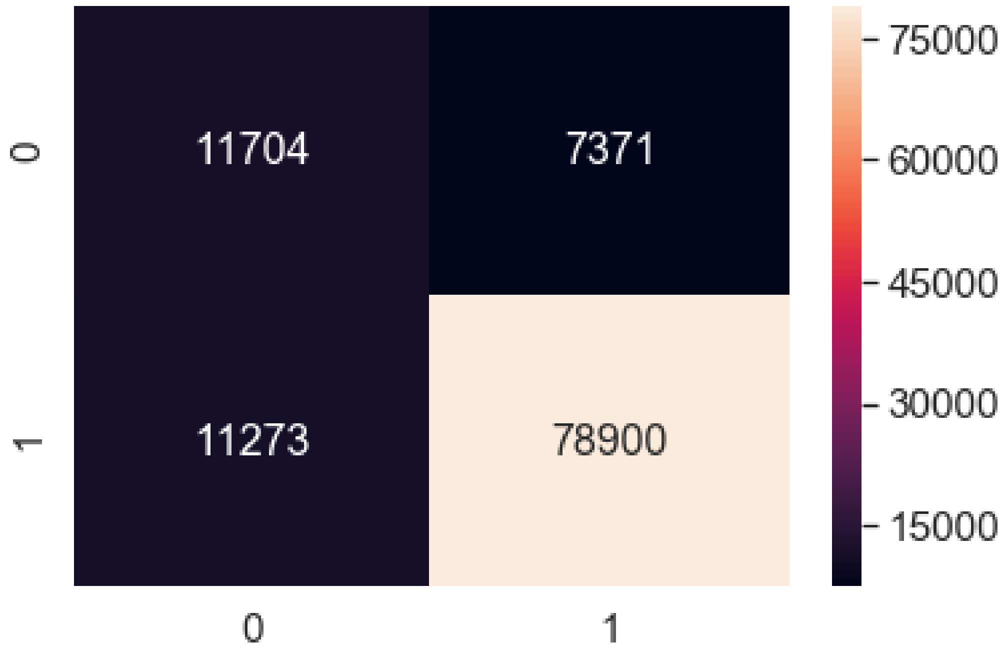



```
In [27]: naive_bayes_model = MultinomialNB(alpha=0.1)
         naive_bayes_model.fit(tfidf_std_data,y_train)
         predictions = naive_bayes_model.predict(standardizing.transform(vector.transform(tfid
         acc = accuracy_score(y_test, predictions) * 100
         print ('accuracy_score = {0}'.format(acc))
```

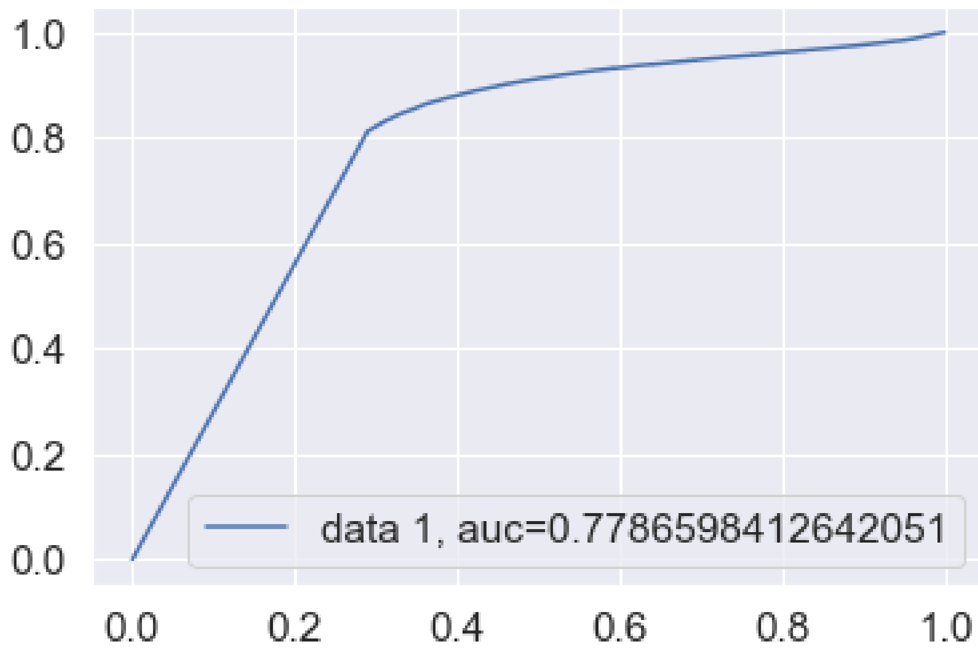
```
accuracy_score = 82.93424135910955
```

```
In [28]: result = confusion_matrix(y_test,predictions)
         #print(result)
         sns.set(font_scale=1.4)#for label size
         sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0xc5912bdcf8>
```




```
In [30]: y_pred_proba = naive_bayes_model.predict_proba(standardizing.transform(vector.transfo
         fpr, tpr, _ = roc_curve(y_test, y_pred_proba,pos_label=1)
         auc = roc_auc_score(y_test, y_pred_proba)
         plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
         plt.legend(loc=4)
         plt.show()
```



8 5) FEATURE ENGINEERING

The AUC value is close to 0.5 which is undesirable,and we can see from confusion matrix that the model is biased towards positive reviews, so we apply feature engineering and add some text from review SUMMARY as well.

9 5.1) BAG OF WORDS WITH BI-GRAM

```
In [46]: #preprocessing text just like reviews
         cleaned_summary = cleanedtext(final_data['Summary'].values)

In [47]: final_data['Cleanedsummary']= cleaned_summary #adding a column of CleanedSummary whic
         print (final_data.shape)
         final_data.head(2) #below the processed review can be seen in the CleanedText Column

(364159, 12)


Out[47]:             Id   ProductId         UserId        ProfileName  \
         150523  150524  0006641040  ACITT7DI6IDDL     shari zychinski
         150500  150501  0006641040  AJ46FKXOVC7NR  Nicholas A Mesiano

                 HelpfulnessNumerator  HelpfulnessDenominator  Score       Time  \
         150523                     0                       0      1  939340800
```

```
150500                          2                          2      1  940809600

                                                   Summary  \
150523                              EVERY book is educational
150500  This whole series is great way to spend time w...

                                                      Text  \
150523  this witty little book makes my son laugh at l...
150500  I can remember seeing the show when it aired o...

                                               CleanedText  \
150523  b'witti littl book make son laugh loud recit c...
150500  b'rememb see show air televis year ago child s...

                                  Cleanedsummary
150523                        b'everi book educ'
150500  b'whole seri great way spend time child'
```

In [48]:
```
#final_data['Cleaned']= final_string + cleaned_summary #adding a column of CleanedTex
final_data['Cleaned']= final_data['CleanedText'].map(str) + " " + final_data['Cleaneds
print (final_data.shape)
final_data.head(2) #below the processed review can be seen in the CleanedText Column
```

```
(364159, 13)
```

Out[48]:
```
            Id   ProductId          UserId        ProfileName  \
150523  150524  0006641040   ACITT7DI6IDDL     shari zychinski
150500  150501  0006641040   AJ46FKXOVC7NR  Nicholas A Mesiano

        HelpfulnessNumerator  HelpfulnessDenominator  Score       Time  \
150523                     0                       0      1  939340800
150500                     2                       2      1  940809600

                                                   Summary  \
150523                              EVERY book is educational
150500  This whole series is great way to spend time w...

                                                      Text  \
150523  this witty little book makes my son laugh at l...
150500  I can remember seeing the show when it aired o...

                                               CleanedText  \
150523  b'witti littl book make son laugh loud recit c...
150500  b'rememb see show air televis year ago child s...

                                  Cleanedsummary  \
150523                        b'everi book educ'
```

```
150500  b'whole seri great way spend time child'

                                                   Cleaned
150523  b'witti littl book make son laugh loud recit c...
150500  b'rememb see show air televis year ago child s...
```

```
In [3]: X_train1, X_test1, y_train, y_test = train_test_split(final_data['Cleaned'].values, fi
                                                                    test_size=0.3, random_sta
```

```
In [4]: # intializing for bag of words with bi gram
        model1 = CountVectorizer(ngram_range = (1,2),dtype=float)
        final_counts1 = model1.fit_transform(X_train1)
```

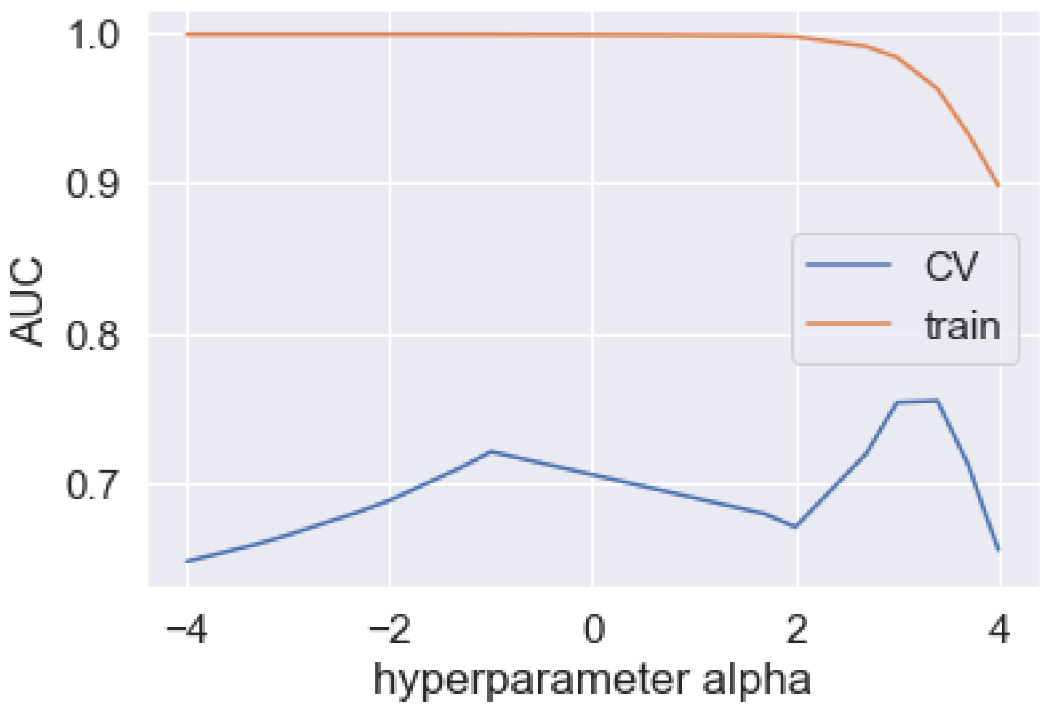
```
In [5]: #standardizing the bag of words
        from sklearn.preprocessing import StandardScaler
        standardizing  = StandardScaler(with_mean = False)
        final_std_data1 = standardizing.fit_transform(final_counts1)
        final_std_data1.shape
```

```
Out[5]: (254911, 2352748)
```

```
In [34]: # creating list for hyperparameter alpha
         alpha_values = [0.0001,0.0005,0.001,0.005,0.01,0.05,0.1,50,100,500,1000,2500,5000,750
         # empty list that will hold cv scores

         cv_scores = []
         train_auc_values = []

         # perform 10-fold cross validation
         for alpha in alpha_values:
             naive_bayes_model = MultinomialNB(alpha=alpha)
             auc = make_scorer(roc_auc_score)
             auc_scores = cross_val_score(naive_bayes_model, final_std_data1, y_train, cv=10, 
             naive_bayes_model.fit(final_std_data1,y_train)
             y_pred_proba = naive_bayes_model.predict_proba(final_std_data1)[::,1]
             train_auc = roc_auc_score(y_train, y_pred_proba)
             train_auc_values.append(train_auc)
             cv_scores.append(auc_scores.mean())

         print ('train data scores')
         print (train_auc_values)
         print ('*'*50)
         print ('CV scores')
         print (cv_scores)

         # changing to misclassification error

         log = [math.log10(x) for x in alpha_values]
         # plot misclassification error vs alpha
```

```
plt.plot(log, cv_scores,label='CV')
#plt.label('cv_auc')
plt.plot(log,train_auc_values,label='train')
#plt.label('train_auc')
plt.legend()
plt.xlabel('hyperparameter alpha')
plt.ylabel('AUC')
plt.show()
```

```
train data scores
[0.9988944212277083, 0.9988943807935874, 0.9988943611524251, 0.9988942948331923, 0.99889418177
**************************************************
CV scores
[0.6486895968463751, 0.6603606611617309, 0.666283840707478, 0.681697301840068, 0.68928790900680
```

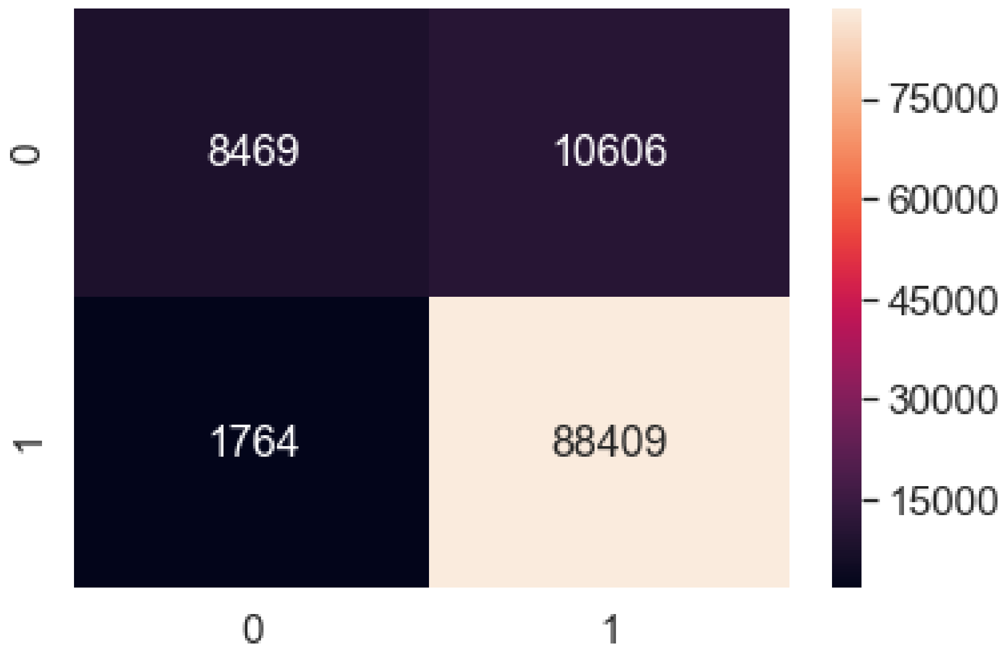



```
In [35]: naive_bayes_model = MultinomialNB(alpha=1000)
         naive_bayes_model.fit(final_std_data1,y_train)
         predictions = naive_bayes_model.predict(standardizing.transform(model1.transform(X_tes
         acc = accuracy_score(y_test, predictions) * 100
         print ('accuracy_score = {0}'.format(acc))
```

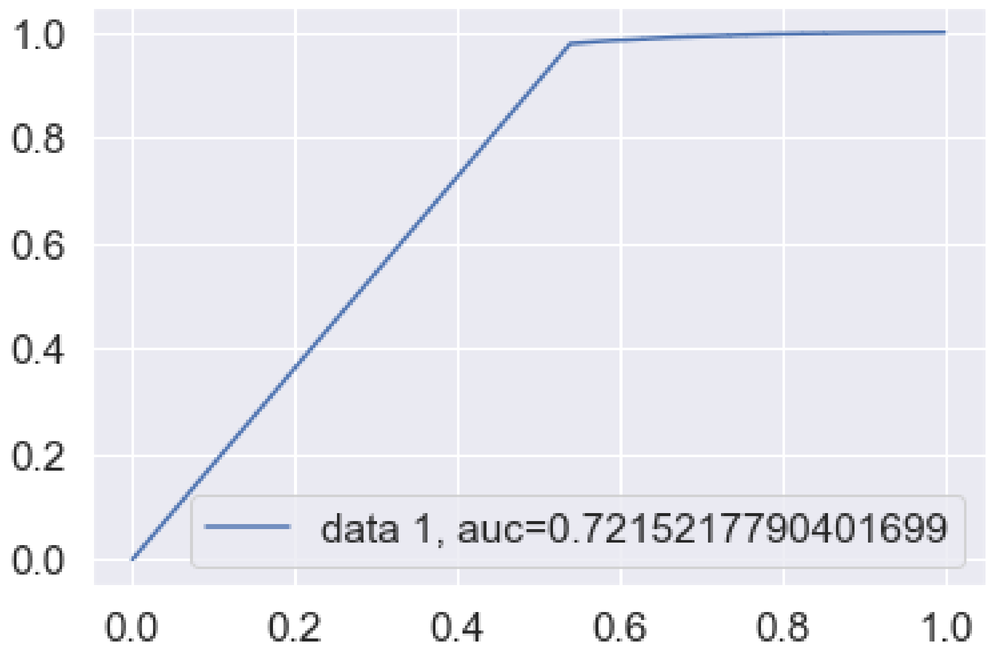
```
accuracy_score = 88.67713825424721
```

In [36]:
```
result = confusion_matrix(y_test,predictions)
#print(result)
sns.set(font_scale=1.4)#for label size
sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xc5f39ba588>
```



In [37]:
```
y_pred_proba = naive_bayes_model.predict_proba(standardizing.transform(model1.transfo
fpr, tpr, _ = roc_curve(y_test, y_pred_proba,pos_label=1)
auc = roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



10 5.2) TF-IDF WITH BI-GRAM

In [8]:
```
# split the data set into train and test
tfidf_train1, tfidf_test1, y_train, y_test = train_test_split(final_data['Cleaned'].val
                                                              test_size=0.3, random_sta
```

In [10]:
```
vector1 = TfidfVectorizer(ngram_range = (1,2))
tf_idf_vector1 = vector1.fit_transform(tfidf_train1)
tf_idf_vector1.shape
```

Out[10]: (254911, 2352748)

In [11]:
```
standardizing = StandardScaler(with_mean=False)
tfidf_std_data1 = standardizing.fit_transform(tf_idf_vector1)
print (tfidf_std_data1.shape)
np.mean(tfidf_std_data1)
```

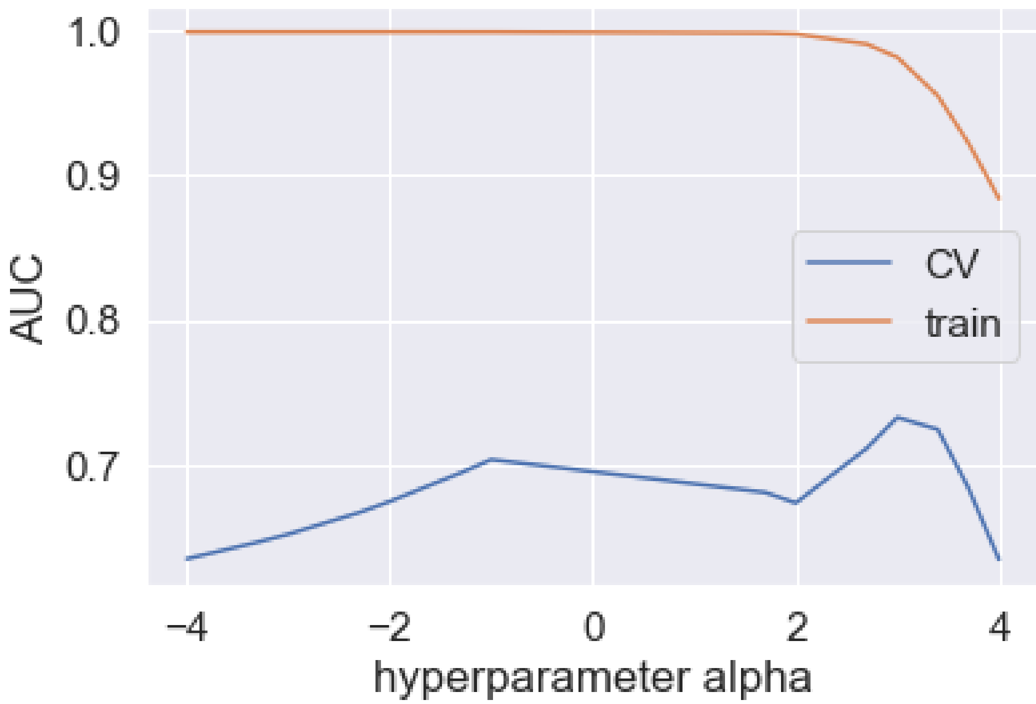
```
(254911, 2352748)
```

Out[11]: 0.0030033202478754743

In [41]:
```
# creating list for hyperparameter alpha
alpha_values = [0.0001,0.0005,0.001,0.005,0.01,0.05,0.1,50,100,500,1000,2500,5000,750
# empty list that will hold cv scores
```

```
cv_scores = []
train_auc_values = []

# perform 10-fold cross validation
for alpha in alpha_values:
    naive_bayes_model = MultinomialNB(alpha=alpha)
    auc = make_scorer(roc_auc_score)
    auc_scores = cross_val_score(naive_bayes_model, tfidf_std_data1, y_train, cv=10, s
    naive_bayes_model.fit(tfidf_std_data1,y_train)
    y_pred_proba = naive_bayes_model.predict_proba(tfidf_std_data1)[::,1]
    train_auc = roc_auc_score(y_train, y_pred_proba)
    train_auc_values.append(train_auc)
    cv_scores.append(auc_scores.mean())

print ('train data scores')
print (train_auc_values)
print ('*'*50)
print ('CV scores')
print (cv_scores)

# changing to misclassification error

log = [math.log10(x) for x in alpha_values]
# plot misclassification error vs alpha
plt.plot(log, cv_scores,label='CV')
#plt.label('cv_auc')
plt.plot(log,train_auc_values,label='train')
#plt.label('train_auc')
plt.legend()
plt.xlabel('hyperparameter alpha')
plt.ylabel('AUC')
plt.show()
```

```
train data scores
[0.9992022099391863, 0.9992022099391863, 0.9992022099391863, 0.9992022099391863, 0.99920219151
**************************************************
CV scores
[0.63465359141029, 0.646233507964389, 0.651712109785119, 0.6667340334441021, 0.67442041199042
```

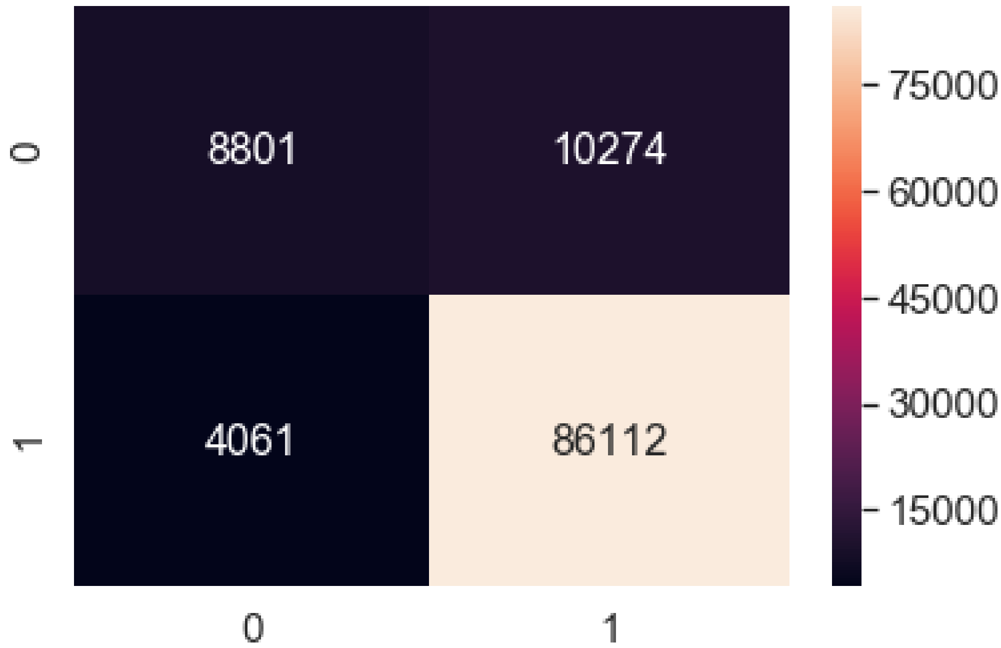



```
In [42]: naive_bayes_model = MultinomialNB(alpha=1000)
         naive_bayes_model.fit(tfidf_std_data1,y_train)
         predictions = naive_bayes_model.predict(standardizing.transform(vector1.transform(tfi
         acc = accuracy_score(y_test, predictions) * 100
         print ('Accuracy = {0}'.format(acc))
```

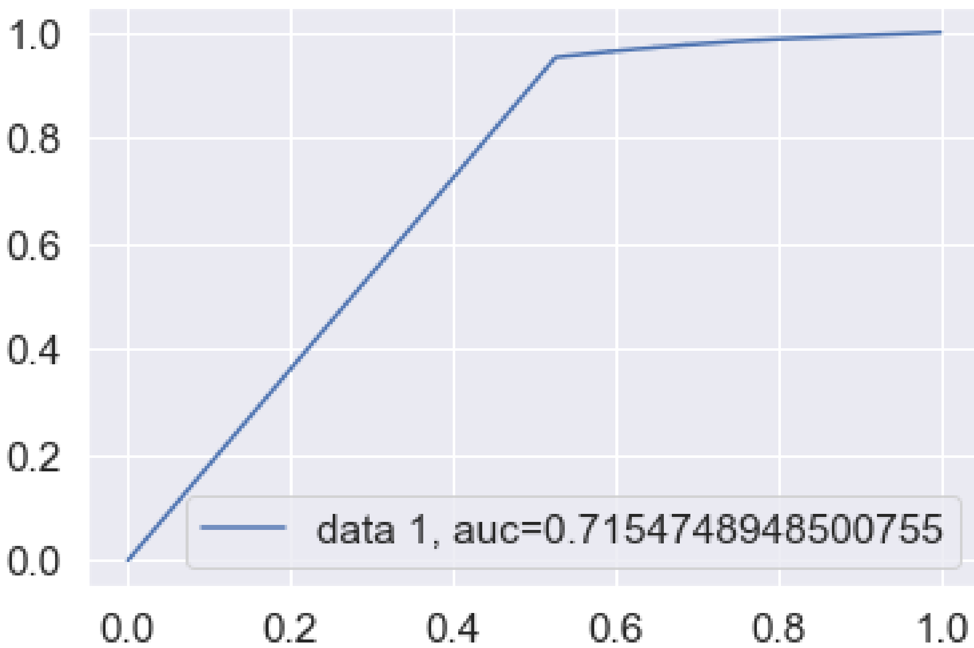
```
Accuracy = 86.87847832454598
```

```
In [43]: result = confusion_matrix(y_test,predictions)
         #print(result)
         sns.set(font_scale=1.4)#for label size
         sns.heatmap(result, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0xc601e41780>
```




```
In [44]: y_pred_proba = naive_bayes_model.predict_proba(standardizing.transform(vector1.transfo
         fpr, tpr, _ = roc_curve(y_test, y_pred_proba,pos_label=1)
         #auc = roc_auc_score(y_test, y_pred_proba)
         auc = np.trapz(tpr,fpr)
         plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
         plt.legend(loc=4)
         plt.show()
```



11 6) FEATURE IMPORTANCE

11.1 6.1) BAG OF WORDS WITH BI-GRAM

```
In [16]: #https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-naive-
         clf = MultinomialNB(alpha=1000)
         clf.fit(final_std_data1,y_train)
         max_ind_pos=np.argsort((clf.feature_log_prob_)[1])[::-1][0:25]
         max_ind_neg=np.argsort((clf.feature_log_prob_)[0])[::-1][0:25]
         print ('positiive class top 25 features')
         print(np.take(model1.get_feature_names(), max_ind_pos))
         print ('*'*100)
         print ('negative class top 25 features')
         print(np.take(model1.get_feature_names(), max_ind_neg))
```

```
positiive class top 25 features
['great' 'not' 'love' 'good' 'like' 'tast' 'one' 'tri' 'best' 'flavor'
 'use' 'make' 'get' 'product' 'find' 'time' 'buy' 'would' 'realli' 'price'
 'also' 'much' 'delici' 'littl' 'store']
****************************************************************************************************
negative class top 25 features
['not' 'tast' 'disappoint' 'like' 'product' 'would' 'bad' 'not buy' 'buy'
 'one' 'money' 'tri' 'wast' 'even' 'wast money' 'order' 'worst' 'aw'
 'horribl' 'return' 'terribl' 'tast like' 'not good' 'thought' 'flavor']
```

11.2 6.2) TF-IDF WITH BI-GRAM

In [17]:
```
#https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-naive-
clf = MultinomialNB(alpha=1000)
clf.fit(tfidf_std_data1,y_train)
max_ind_pos=np.argsort((clf.feature_log_prob_)[1])[::-1][0:25]
max_ind_neg=np.argsort((clf.feature_log_prob_)[0])[::-1][0:25]
print ('positiive class top 25 features')
print(np.take(vector1.get_feature_names(), max_ind_pos))
print ('*'*100)
print ('negative class top 25 features')
print(np.take(vector1.get_feature_names(), max_ind_neg))
```

```
positiive class top 25 features
['not' 'great' 'love' 'good' 'like' 'tast' 'use' 'flavor' 'one' 'tri'
 'make' 'product' 'best' 'get' 'would' 'time' 'find' 'buy' 'also' 'realli'
 'price' 'littl' 'amazon' 'much' 'eat']
****************************************************************************************************
negative class top 25 features
['not' 'tast' 'disappoint' 'like' 'would' 'product' 'bad' 'not buy' 'one'
 'money' 'wast' 'tri' 'buy' 'return' 'horribl' 'even' 'worst' 'wast money'
 'aw' 'terribl' 'flavor' 'not good' 'tast like' 'review' 'thought']
```

12 7) CONCLUSION

In [104]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ['Featurization','aptimal_alpha','CV_accuracy','test_accuracy','AUC']
x.add_row(['BOW with uni-gram','0.1','74.5','81.05','0.7775'])
x.add_row(['TFIDF with uni-gram','50','74.5','82.93','0.778'])
x.add_row(['BOW with Bi-gram ','1000','75.43','88.67','0.721'])
x.add_row(['and review summary','','','',''])
x.add_row(['TF-IDF with Bi-gram ','1000','77.25','82.6','0.715'])
x.add_row(['and review summary','','','',''])
print (x)
```

```
+----------------------+---------------+-------------+---------------+--------+
|    Featurization     | aptimal_alpha | CV_accuracy | test_accuracy |  AUC   |
+----------------------+---------------+-------------+---------------+--------+
|  BOW with uni-gram   |      0.1      |     74.5    |     81.05     | 0.7775 |
| TFIDF with uni-gram  |       50      |     74.5    |     82.93     | 0.778  |
|  BOW with Bi-gram    |      1000     |    75.43    |     88.67     | 0.721  |
|  and review summary  |               |             |               |        |
| TF-IDF with Bi-gram  |      1000     |    77.25    |      82.6     | 0.715  |
|  and review summary  |               |             |               |        |
```

```
+----------------------+---------------+-------------+---------------+--------+
```

BOW with Bi-gram and using review summary increases the accuracy, but the AUC is less compared to uni-fram, if we observe confusion matrix we can see that TNR is high and FNR is less compared to oter methods