

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 - unique 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 considered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered neutral 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 will 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 column
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=False)
```

```
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"},keep =
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='quicksort')
```

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 observed 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,
                                                            test_size=0.30, random_state=42)

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, scoring=auc)
            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
```

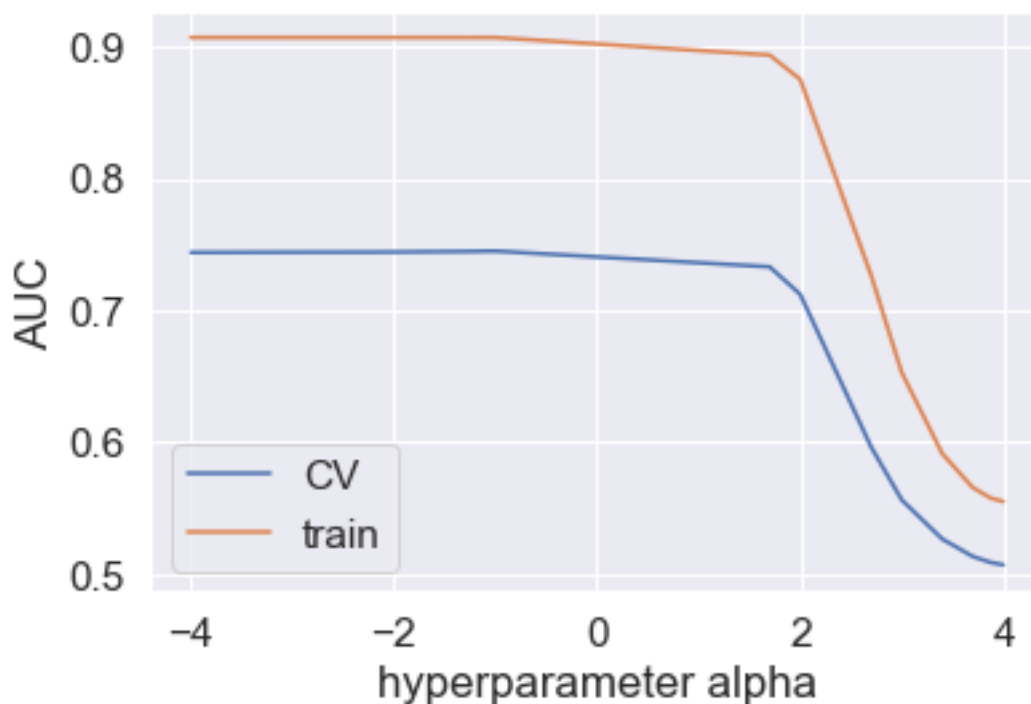
```
log = [math.log10(x) for x in alpha_values]
# plot misclassification error vs alpha
plt.plot(log, cv_scores, label='CV')
#plt.label('cv_auc')
plt.plot(log, train_auc_values, label='train')
#plt.label('train_auc')
plt.legend()
plt.xlabel('hyperparameter alpha')
plt.ylabel('AUC')
plt.show()
```

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

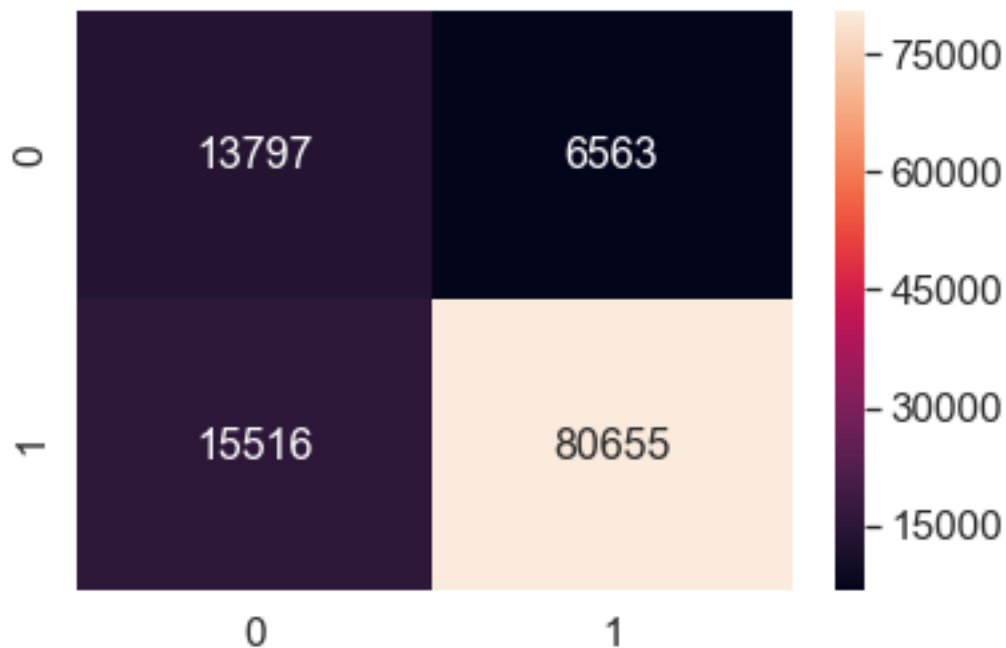
```
In [20]: naive_bayes_model = MultinomialNB(alpha=0.1)
naive_bayes_model.fit(final_std_data,y_train)
predictions = naive_bayes_model.predict(standardizing.transform(model.transform(X_test)))
```

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

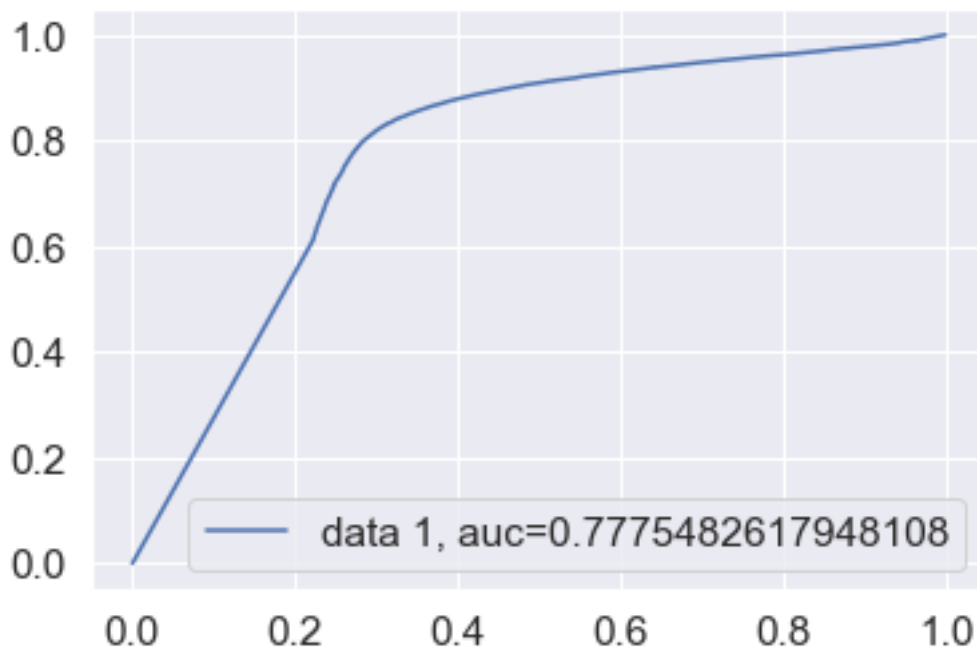
accuracy_score = 81.05311033115652

```
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.transform(
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_state=42)
```

```
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
```

```

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, scoring=auc)
    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)

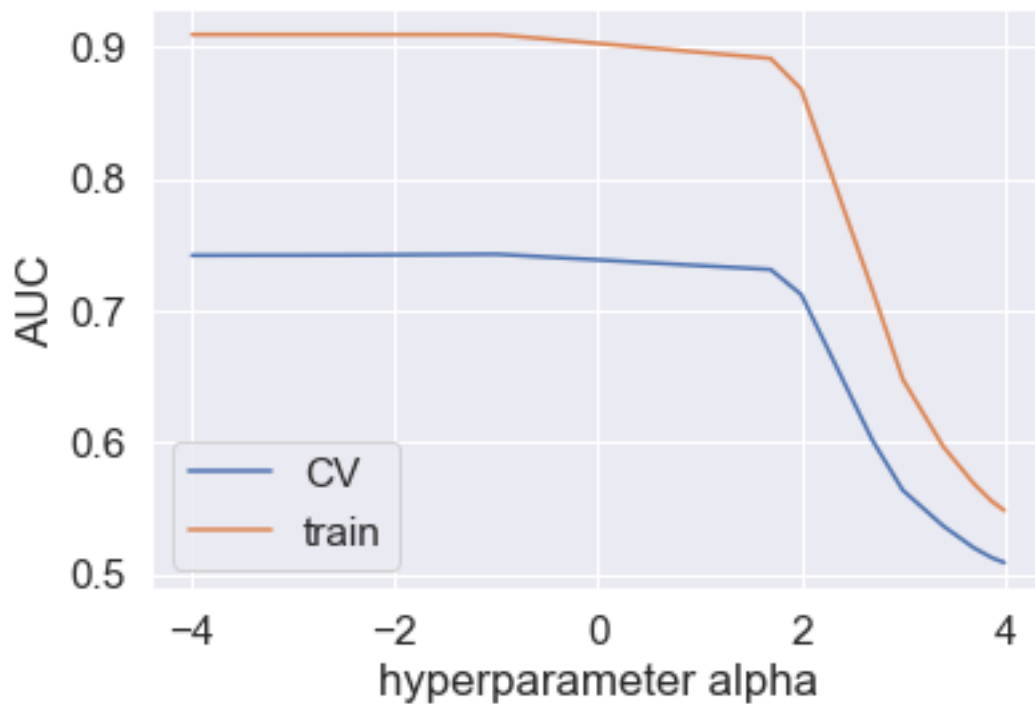
# changing to misclassification error

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, 0.9090895772301115, 0.9090824881111111, 0.9090753990000001, 0.9090683098888889, 0.9090612197777778, 0.9090541296666667, 0.9090470395555556, 0.9090399494444444, 0.9090328593333333, 0.9090257692222222, 0.9090186791111111, 0.9090115890000001, 0.9090044988888889, 0.9089974087777778, 0.9089903186666667, 0.9089832285555556, 0.9089761384444444, 0.9089690483333333, 0.9089619582222222, 0.9089548681111111, 0.9089477780000001, 0.9089406878888889, 0.9089335977777778, 0.9089265076666667, 0.9089194175555556, 0.9089123274444444, 0.9089052373333333, 0.9088981472222222, 0.9088910571111111, 0.9088839670000001, 0.9088768768888889, 0.9088697867777778, 0.9088626966666667, 0.9088556065555556, 0.9088485164444444, 0.9088414263333333, 0.9088343362222222, 0.9088272461111111, 0.9088201560000001, 0.9088130658888889, 0.9088059757777778, 0.9087988856666667, 0.9087917955555556, 0.9087847054444444, 0.9087776153333333, 0.9087705252222222, 0.9087634351111111, 0.9087563450000001, 0.9087492548888889, 0.9087421647777778, 0.9087350746666667, 0.9087279845555556, 0.9087208944444444, 0.9087138043333333, 0.9087067142222222, 0.9086996241111111, 0.9086925340000001, 0.9086854438888889, 0.9086783537777778, 0.9086712636666667, 0.9086641735555556, 0.9086570834444444, 0.9086500000000001, 0.9086429166666667, 0.9086358333333333, 0.9086287500000001, 0.9086216666666667, 0.9086145833333333, 0.9086075000000001, 0.9086004166666667, 0.9085933333333333, 0.9085862500000001, 0.9085791666666667, 0.9085720833333333, 0.9085650000000001, 0.9085579166666667, 0.9085508333333333, 0.9085437500000001, 0.9085366666666667, 0.9085295833333333, 0.9085225000000001, 0.9085154166666667, 0.9085083333333333, 0.9085012500000001, 0.9084941666666667, 0.9084870833333333, 0.9084800000000001, 0.9084729166666667, 0.9084658333333333, 0.9084587500000001, 0.9084516666666667, 0.9084445833333333, 0.9084375000000001, 0.9084304166666667, 0.9084233333333333, 0.9084162500000001, 0.9084091666666667, 0.9084020833333333, 0.9083950000000001, 0.9083879166666667, 0.9083808333333333, 0.9083737500000001, 0.9083666666666667, 0.9083595833333333, 0.9083525000000001, 0.9083454166666667, 0.9083383333333333, 0.9083312500000001, 0.9083241666666667, 0.9083170833333333, 0.9083100000000001, 0.9083029166666667, 0.9082958333333333, 0.9082887500000001, 0.9082816666666667, 0.9082745833333333, 0.9082675000000001, 0.9082604166666667, 0.9082533333333333, 0.9082462500000001, 0.9082391666666667, 0.9082320833333333, 0.9082250000000001, 0.9082179166666667, 0.9082108333333333, 0.9082037500000001, 0.9081966666666667, 0.9081895833333333, 0.9081825000000001, 0.9081754166666667, 0.9081683333333333, 0.9081612500000001, 0.9081541666666667, 0.9081470833333333, 0.9081400000000001, 0.9081329166666667, 0.9081258333333333, 0.9081187500000001, 0.9081116666666667, 0.9081045833333333, 0.9080975000000001, 0.9080904166666667, 0.9080833333333333, 0.9080762500000001, 0.9080691666666667, 0.9080620833333333, 0.9080550000000001, 0.9080479166666667, 0.9080408333333333, 0.9080337500000001, 0.9080266666666667, 0.9080195833333333, 0.9080125000000001, 0.9080054166666667, 0.9079983333333333, 0.9079912500000001, 0.9079841666666667, 0.9079770833333333, 0.9079700000000001, 0.9079629166666667, 0.9079558333333333, 0.9079487500000001, 0.9079416666666667, 0.9079345833333333, 0.9079275000000001, 0.9079204166666667, 0.9079133333333333, 0.9079062500000001, 0.9078991666666667, 0.9078920833333333, 0.9078850000000001, 0.9078779166666667, 0.9078708333333333, 0.9078637500000001, 0.9078566666666667, 0.9078495833333333, 0.9078425000000001, 0.9078354166666667, 0.9078283333333333, 0.9078212500000001, 0.9078141666666667, 0.9078070833333333, 0.9078000000000001, 0.9077929166666667, 0.9077858333333333, 0.9077787500000001, 0.9077716666666667, 0.9077645833333333, 0.9077575000000001, 0.9077504166666667, 0.9077433333333333, 0.9077362500000001, 0.9077291666666667, 0.9077220833333333, 0.9077150000000001, 0.9077079166666667, 0.9077008333333333, 0.9076937500000001, 0.9076866666666667, 0.9076795833333333, 0.9076725000000001, 0.9076654166666667, 0.9076583333333333, 0.9076512500000001, 0.9076441666666667, 0.9076370833333333, 0.9076300000000001, 0.9076229166666667, 0.9076158333333333, 0.9076087500000001, 0.9076016666666667, 0.9075945833333333, 0.9075875000000001, 0.9075804166666667, 0.9075733333333333, 0.9075662500000001, 0.9075591666666667, 0.9075520833333333, 0.9075450000000001, 0.9075379166666667, 0.9075308333333333, 0.9075237500000001, 0.9075166666666667, 0.9075095833333333, 0.9075025000000001, 0.9074954166666667, 0.9074883333333333, 0.9074812500000001, 0.9074741666666667, 0.9074670833333333, 0.9074600000000001, 0.9074529166666667, 0.9074458333333333, 0.9074387500000001, 0.9074316666666667, 0.9074245833333333, 0.9074175000000001, 0.9074104166666667, 0.9074033333333333, 0.9073962500000001, 0.9073891666666667, 0.9073820833333333, 0.9073750000000001, 0.9073679166666667, 0.9073608333333333, 0.9073537500000001, 0.9073466666666667, 0.9073395833333333, 0.9073325000000001, 0.9073254166666667, 0.9073183333333333, 0.9073112500000001, 0.9073041666666667, 0.9072970833333333, 0.9072900000000001, 0.9072829166666667, 0.9072758333333333, 0.9072687500000001, 0.9072616666666667, 0.9072545833333333, 0.9072475000000001, 0.9072404166666667, 0.9072333333333333, 0.9072262500000001, 0.9072191666666667, 0.9072120833333333, 0.9072050000000001, 0.9071979166666667, 0.9071908333333333, 0.9071837500000001, 0.9071766666666667, 0.9071695833333333, 0.9071625000000001, 0.9071554166666667, 0.9071483333333333, 0.9071412500000001, 0.9071341666666667, 0.9071270833333333, 0.9071200000000001, 0.9071129166666667, 0.9071058333333333, 0.9070987500000001, 0.9070916666666667, 0.9070845833333333, 0.9070775000000001, 0.9070704166666667, 0.9070633333333333, 0.9070562500000001, 0.9070491666666667, 0.9070420833333333, 0.9070350000000001, 0.9070279166666667, 0.9070208333333333, 0.9070137500000001, 0.9070066666666667, 0.9069995833333333, 0.9069925000000001, 0.9069854166666667, 0.9069783333333333, 0.9069712500000001, 0.9069641666666667, 0.9069570833333333, 0.9069500000000001, 0.9069429166666667, 0.9069358333333333, 0.9069287500000001, 0.9069216666666667, 0.9069145833333333, 0.9069075000000001, 0.9069004166666667, 0.9068933333333333, 0.9068862500000001, 0.9068791666666667, 0.9068720833333333, 0.9068650000000001, 0.9068579166666667, 0.9068508333333333, 0.9068437500000001, 0.9068366666666667, 0.9068295833333333, 0.9068225000000001, 0.9068154166666667, 0.9068083333333333, 0.9068012500000001, 0.9067941666666667, 0.9067870833333333, 0.9067800000000001, 0.9067729166666667, 0.9067658333333333, 0.9067587500000001, 0.9067516666666667, 0.9067445833333333, 0.9067375000000001, 0.9067304166666667, 0.9067233333333333, 0.9067162500000001, 0.9067091666666667, 0.9067020833333333, 0.9066950000000001, 0.9066879166666667, 0.9066808333333333, 0.9066737500000001, 0.9066666666666667, 0.9066595833333333, 0.9066525000000001, 0.9066454166666667, 0.9066383333333333, 0.9066312500000001, 0.9066241666666667, 0.9066170833333333, 0.9066100000000001, 0.9066029166666667, 0.9065958333333333, 0.9065887500000001, 0.9065816666666667, 0.9065745833333333, 0.9065675000000001, 0.9065604166666667, 0.9065533333333333, 0.9065462500000001, 0.9065391666666667, 0.9065320833333333, 0.9065250000000001, 0.9065179166666667, 0.9065108333333333, 0.9065037500000001, 0.9064966666666667, 0.9064895833333333, 0.9064825000000001, 0.9064754166666667, 0.9064683333333333, 0.9064612500000001, 0.9064541666666667, 0.9064470833333333, 0.9064400000000001, 0.9064329166666667, 0.9064258333333333, 0.9064187500000001, 0.9064116666666667, 0.9064045833333333, 0.9063975000000001, 0.9063904166666667, 0.9063833333333333, 0.9063762500000001, 0.9063691666666667, 0.9063620833333333, 0.9063550000000001, 0.9063479166666667, 0.9063408333333333, 0.9063337500000001, 0.9063266666666667, 0.9063195833333333, 0.9063125000000001, 0.9063054166666667, 0.9062983333333333, 0.9062912500000001, 0.9062841666666667, 0.9062770833333333, 0.9062700000000001, 0.9062629166666667, 0.9062558333333333, 0.9062487500000001, 0.9062416666666667, 0.9062345833333333, 0.9062275000000001, 0.9062204166666667, 0.9062133333333333, 0.9062062500000001, 0.9061991666666667, 0.9061920833333333, 0.9061850000000001, 0.9061779166666667, 0.9061708333333333, 0.9061637500000001, 0.9061566666666667, 0.9061495833333333, 0.9061425000000001, 0.9061354166666667, 0.9061283333333333, 0.9061212500000001, 0.9061141666666667, 0.9061070833333333, 0.9061000000000001, 0.9060929166666667, 0.9060858333333333, 0.9060787500000001, 0.9060716666666667, 0.9060645833333333, 0.9060575000000001, 0.9060504166666667, 0.9060433333333333, 0.9060362500000001, 0.9060291666666667, 0.9060220833333333, 0.9060150000000001, 0.9060079166666667, 0.9059995833333333, 0.9059925000000001, 0.9059854166666667, 0.9059783333333333, 0.9059712500000001, 0.9059641666666667, 0.9059570833333333, 0.9059500000000001, 0.9059429166666667, 0.9059358333333333, 0.9059287500000001, 0.9059216666666667, 0.9059145833333333, 0.9059075000000001, 0.9059004166666667, 0.9058933333333333, 0.9058862500000001, 0.9058791666666667, 0.9058720833333333, 0.9058650000000001, 0.9058579166666667, 0.9058508333333333, 0.9058437500000001, 0.9058366666666667, 0.9058295833333333, 0.9058225000000001, 0.9058154166666667, 0.9058083333333333, 0.9058012500000001, 0.9057941666666667, 0.9057870833333333, 0.9057800000000001, 0.9057729166666667, 0.9057658333333333, 0.9057587500000001, 0.9057516666666667, 0.9057445833333333, 0.9057375000000001, 0.9057304166666667, 0.9057233333333333, 0.9057162500000001, 0.9057091666666667, 0.9057020833333333, 0.9056950000000001, 0.9056879166666667, 0.9056808333333333, 0.9056737500000001, 0.9056666666666667, 0.9056595833333333, 0.9056525000000001, 0.9056454166666667, 0.9056383333333333, 0.9056312500000001, 0.9056241666666667, 0.9056170833333333, 0.9056100000000001, 0.9056029166666667, 0.9055958333333333, 0.9055887500000001, 0.9055816666666667, 0.9055745833333333, 0.9055675000000001, 0.9055604166666667, 0.9055533333333333, 0.9055462500000001, 0.9055391666666667, 0.9055320833333333, 0.9055250000000001, 0.9055179166666667, 0.9055108333333333, 0.9055037500000001, 0.9054966666666667, 0.9054895833333333, 0.9054825000000001, 0.9054754166666667, 0.9054683333333333, 0.9054612500000001, 0.9054541666666667, 0.9054470833333333, 0.9054400000000001, 0.9054329166666667, 0.9054258333333333, 0.9054187500000001, 0.9054116666666667, 0.9054045833333333, 0.9053975000000001, 0.9053904166666667, 0.9053833333333333, 0.9053762500000001, 0.9053691666666667, 0.9053620833333333, 0.9053550000000001, 0.9053479166666667, 0.9053408333333333, 0.9053337500000001, 0.9053266666666667, 0.9053195833333333, 0.9053125000000001, 0.9053054166666667, 0.9052983333333333, 0.9052912500000001, 0.9052841666666667, 0.9052770833333333, 0.9052700000000001, 0.9052629166666667, 0.9052558333333333, 0.9052487500000001, 0.9052416666666667, 0.9052345833333333, 0.9052275000000001, 0.9052204166666667, 0.9052133333333333, 0.9052062500000001, 0.9051991666666667, 0.9051920833333333, 0.9051850000000001, 0.9051779166666667, 0.9051708333333333, 0.9051637500000001, 0.9051566666666667, 0.9051495833333333, 0.9051425000000001, 0.9051354166666667, 0.9051283333333333, 0.9051212500000001, 0.9051141666666667, 0.9051070833333333, 0.9051000000000001, 0.9050929166666667, 0.9050858333333333, 0.9050787500000001, 0.9050716666666667, 0.9050645833333333, 0.9050575000000001, 0.9050504166666667, 0.9050433333333333, 0.9050362500000001, 0.9050291666666667, 0.9050220833333333, 0.9050150000000001, 0.9050079166666667, 0.9050008333333333, 0.9049937500000001, 0.9049866666666667, 0.9049795833333333, 0.9049725000000001, 0.9049654166666667, 0.9049583333333333, 0.9049512500000001, 0.9049441666666667, 0.9049370833333333, 0.9049300000000001, 0.9049229166666667, 0.9049158333333333, 0.904

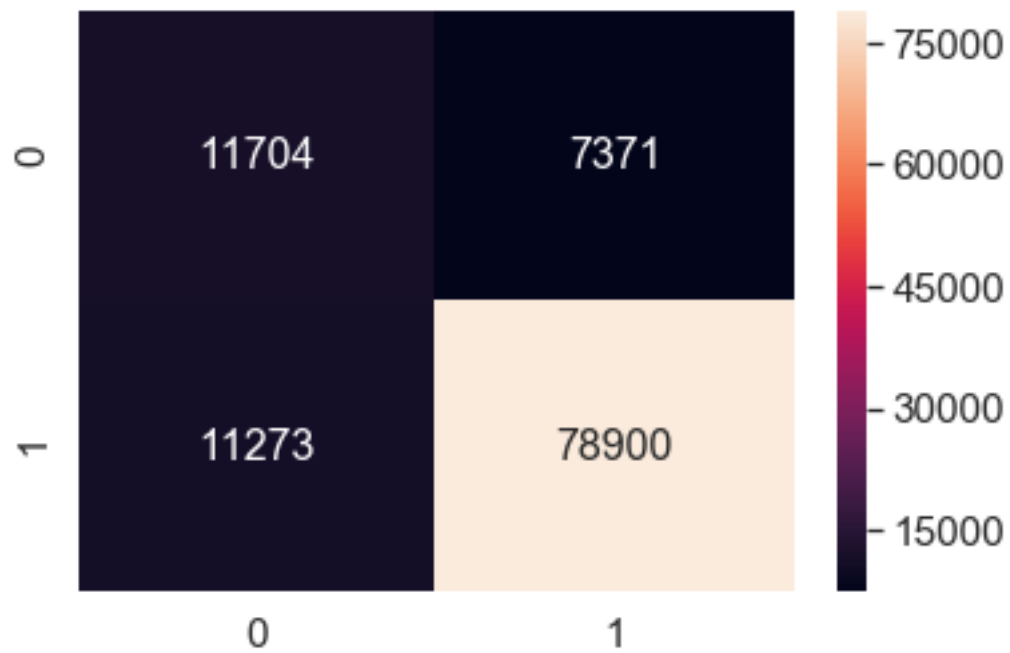


```
In [27]: naive_bayes_model = MultinomialNB(alpha=0.1)
naive_bayes_model.fit(tfidf_std_data,y_train)
predictions = naive_bayes_model.predict(standardizing.transform(vector.transform(tfidf_std_data)))
acc = accuracy_score(y_test, predictions) * 100
print ('accuracy_score = {0}'.format(acc))
```

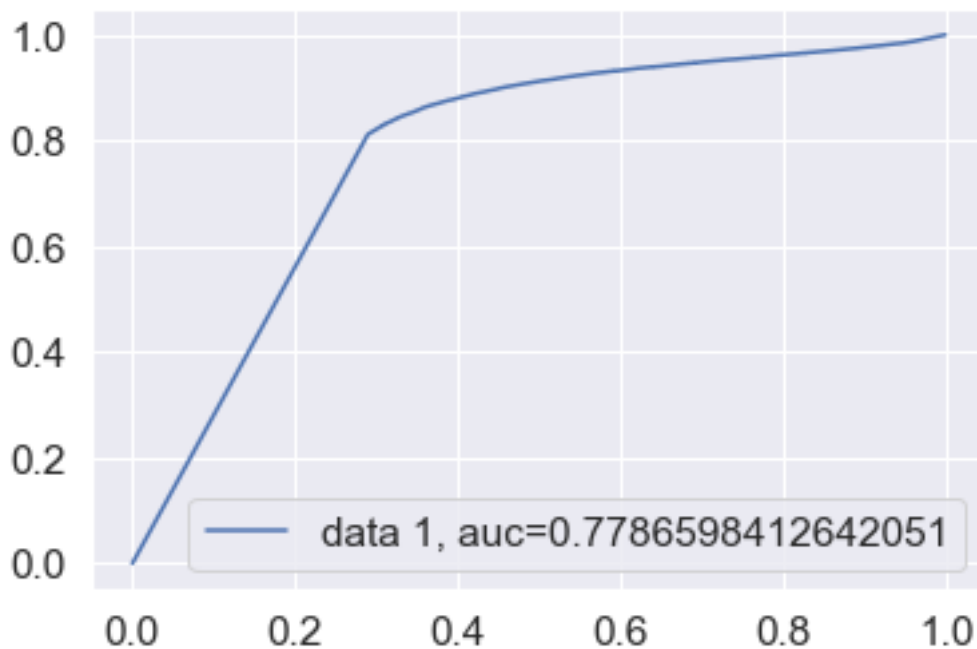
```
accuracy_score = 82.93424135910955
```

```
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.transform(y_test)))
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['CleanedsuSummary'] = cleaned_summary #adding a column of CleanedSummary which
         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 CleanedText
final_data['Cleaned']= final_data['CleanedText'].map(str) + " " + final_data['CleanedSummary']
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, final_data['y'],
                                                             test_size=0.3, random_state=42)

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 = [0.0001,0.0005,0.001,0.005,0.01,0.05,0.1,50,100,500,1000,2500,5000,7500]
# 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_data1, y_train, cv=10, scoring=auc)
    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()

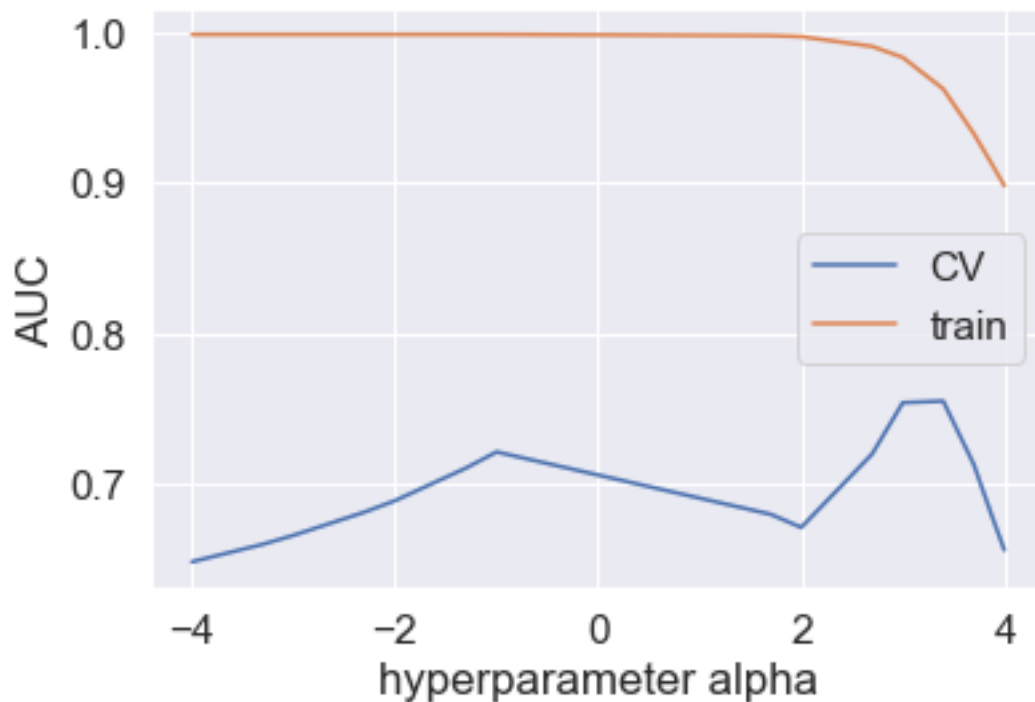
```

train data scores

```
[0.9988944212277083, 0.9988943807935874, 0.9988943611524251, 0.9988942948331923, 0.998894181770
```

CV scores

```
[0.6486895968463751, 0.6603606611617309, 0.666283840707478, 0.681697301840068, 0.68928790900680
```



```

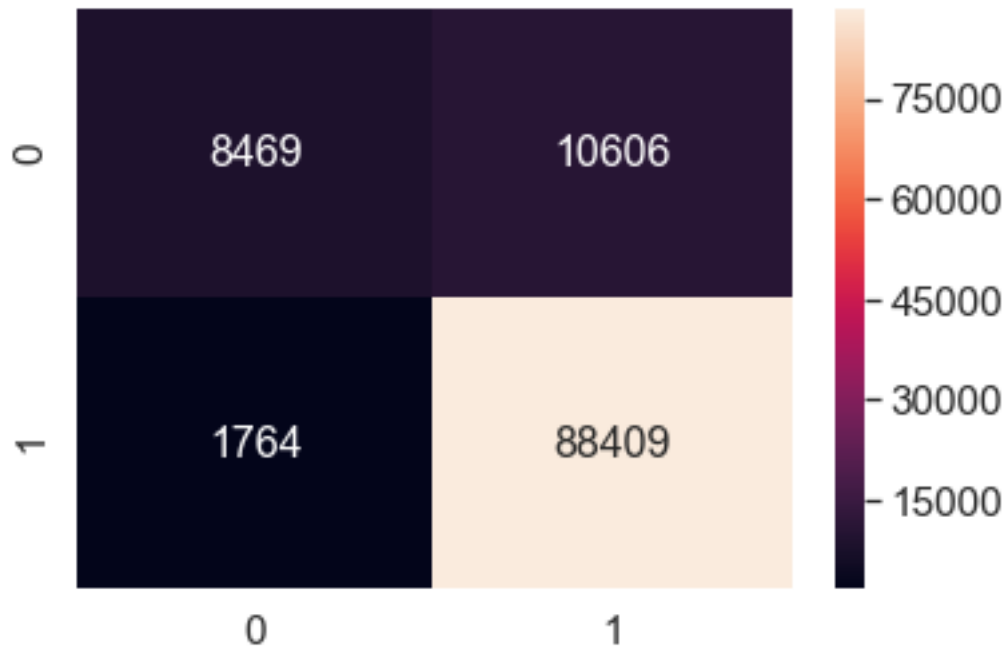
In [35]: naive_bayes_model = MultinomialNB(alpha=1000)
naive_bayes_model.fit(final_std_data1,y_train)
predictions = naive_bayes_model.predict(standardizing.transform(model1.transform(X_test)))
acc = accuracy_score(y_test, predictions) * 100
print ('accuracy_score = {0}'.format(acc))

```

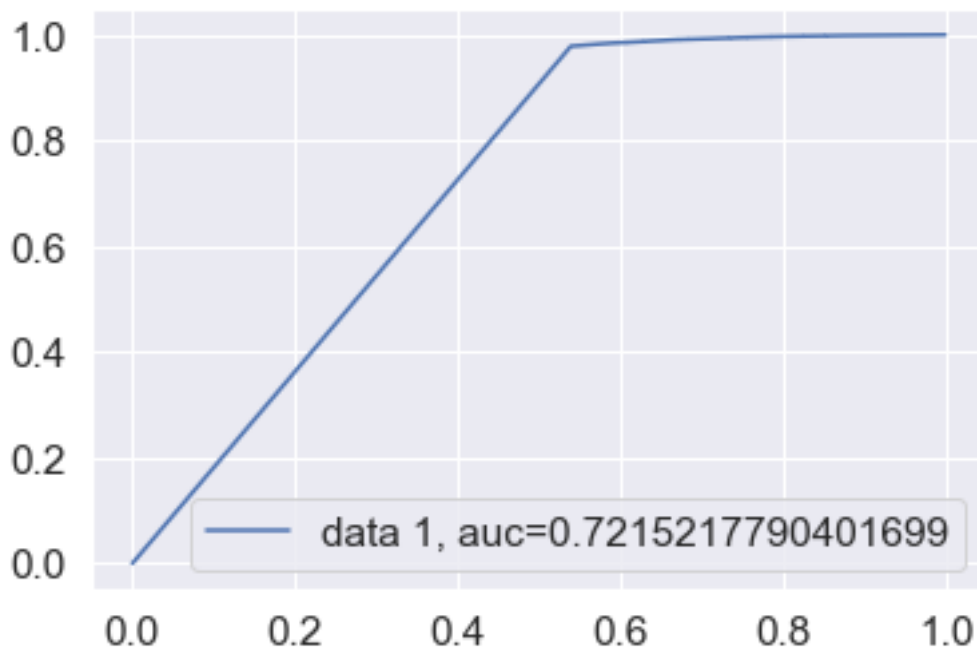
accuracy_score = 88.67713825424721

```
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.transform(
         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'].values,
                                                                    test_size=0.3, random_state=42)
```

```
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 = [0.0001,0.0005,0.001,0.005,0.01,0.05,0.1,50,100,500,1000,2500,5000,7500]
         # 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_data1, y_train, cv=10, scoring=auc)
    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()

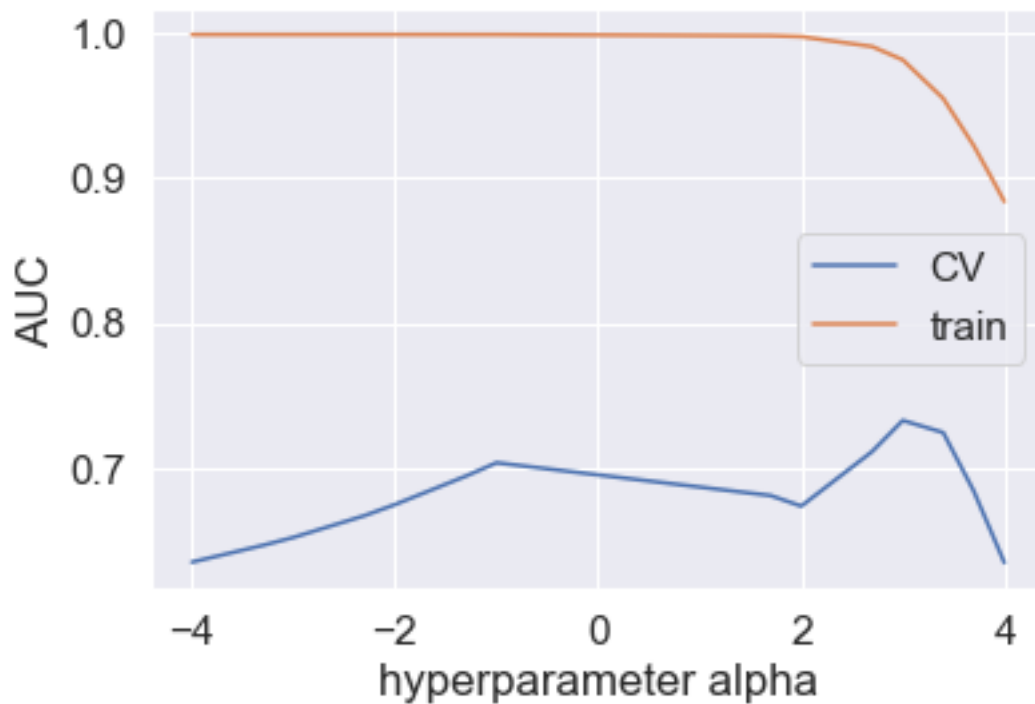
```

train data scores

[0.9992022099391863, 0.9992022099391863, 0.9992022099391863, 0.9992022099391863, 0.9992021915111111]

CV scores

[0.63465359141029, 0.646233507964389, 0.651712109785119, 0.6667340334441021, 0.674420411990424]

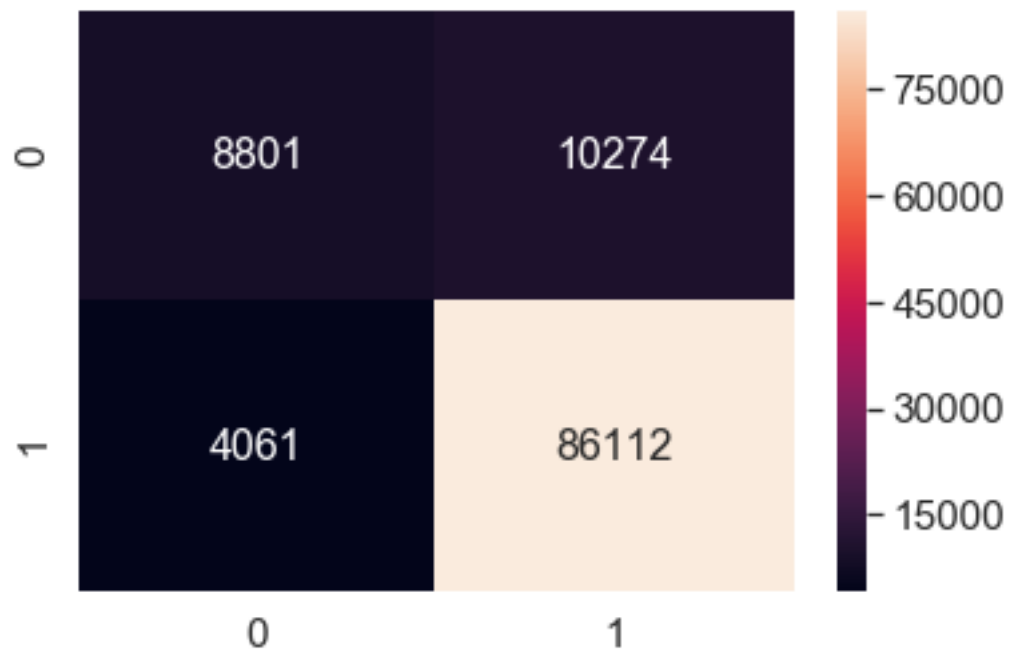


```
In [42]: naive_bayes_model = MultinomialNB(alpha=1000)
naive_bayes_model.fit(tfidf_std_data1,y_train)
predictions = naive_bayes_model.predict(standardizing.transform(vector1.transform(tfi
acc = accuracy_score(y_test, predictions) * 100
print ('Accuracy = {0}'.format(acc))
```

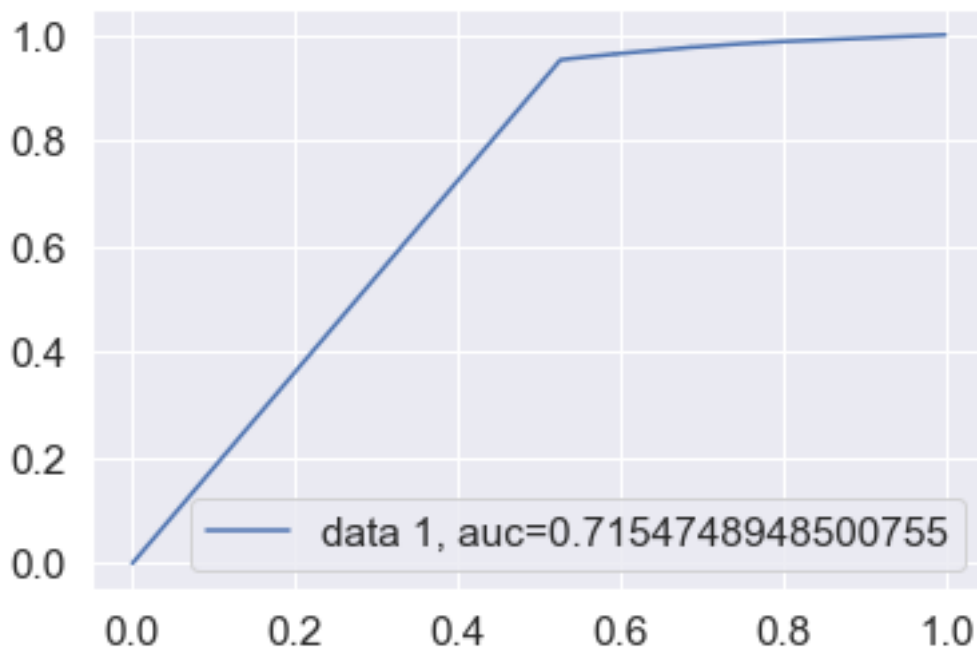
Accuracy = 86.87847832454598

```
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.transform(y_test)))
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-bayes](https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-naive-bayes)

```
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 ('positive 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))
```

positive 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-bayes>

```
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-gram, if we observe confusion matrix we can see that TNR is high and FNR is less compared to other methods