04 Amazon Fine Food Reviews Analysis_NaiveBayes

February 19, 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.

2 [1]. Reading Data

2.1 [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 [218]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


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.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

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

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from sklearn.model_selection import train_test_split
from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
```

In [219]:
```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')
```

```
# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data poi
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3""", con

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negat
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (525814, 10)
```

```
Out[219]:    Id   ProductId          UserId                      ProfileName  \
          0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
          1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
          2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"

             HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
          0                     1                       1      1  1303862400
          1                     0                       0      0  1346976000
          2                     1                       1      1  1219017600

                           Summary                                               Text
          0  Good Quality Dog Food  I have bought several of the Vitality canned d...
          1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
          2  "Delight" says it all  This is a confection that has been around a fe...
```

```
In [220]: display = pd.read_sql_query("""
          SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
          FROM Reviews
          GROUP BY UserId
          HAVING COUNT(*)>1
          """, con)
```

```
In [221]: print(display.shape)
          display.head()
```

```
(80668, 7)
```

```
Out[221]:                UserId   ProductId             ProfileName        Time  Score  \
          0  #oc-R115TNMSPFT9I7  B005ZBZLT4                 Breyton  1331510400      2
          1  #oc-R11D9D7SHXIJB9  B005HG9ESG  Louis E. Emory "hoppy"  1342396800      5
          2  #oc-R11DNU2NBKQ23Z  B005ZBZLT4        Kim Cieszykowski  1348531200      1
          3  #oc-R11O5J5ZVQE25C  B005HG9ESG            Penguin Chick  1346889600      5
          4  #oc-R12KPBODL2B5ZD  B007OSBEV0   Christopher P. Presta  1348617600      1

                                                          Text  COUNT(*)
          0  Overall its just OK when considering the price...         2
          1  My wife has recurring extreme muscle spasms, u...         3
          2  This coffee is horrible and unfortunately not ...         2
          3  This will be the bottle that you grab from the...         3
          4  I didnt like this coffee. Instead of telling y...         2
```

```
In [222]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[222]:               UserId   ProductId                       ProfileName        Time  \
          80638  AZY10LLTJ71NX  B001ATMQK2  undertheshrine "undertheshrine"  1296691200

                 Score                                               Text  COUNT(*)
          80638      5  I bought this 6 pack because for the price tha...         5
```

```
In [223]: display['COUNT(*)'].sum()
```

```
Out[223]: 393063
```

3 [2] Exploratory Data Analysis

3.1 [2.1] Data Cleaning: Deduplication

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 [224]: display= pd.read_sql_query("""
          SELECT *
          FROM Reviews
          WHERE Score != 3 AND UserId="AR5J8UI46CURR"
          ORDER BY ProductID
          """, con)
          display.head()
```

```
Out[224]:        Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
          0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
          1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
          2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
          3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
          4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2

             HelpfulnessDenominator  Score        Time  \
          0                       2      5  1199577600
          1                       2      5  1199577600
          2                       2      5  1199577600
          3                       2      5  1199577600
          4                       2      5  1199577600

                                       Summary  \
          0  LOACKER QUADRATINI VANILLA WAFERS
          1  LOACKER QUADRATINI VANILLA WAFERS
          2  LOACKER QUADRATINI VANILLA WAFERS
          3  LOACKER QUADRATINI VANILLA WAFERS
          4  LOACKER QUADRATINI VANILLA WAFERS

                                                          Text
          0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
          1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
          2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
          3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
          4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

```
In [225]: #Sorting data according to ProductId in ascending order
          sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fa

In [226]: #Deduplication of entries
          final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, kee
          final.shape
```

```
Out[226]: (364173, 10)
```

```
In [227]: #Checking to see how much % of data still remains
          (final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
Out[227]: 69.25890143662969
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

```
In [228]: display= pd.read_sql_query("""
          SELECT *
          FROM Reviews
          WHERE Score != 3 AND Id=44737 OR Id=64422
          ORDER BY ProductID
          """, con)

          display.head()
```

```
Out[228]:       Id   ProductId          UserId              ProfileName  \
          0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
          1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

             HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
          0                     3                       1      5  1224892800
          1                     3                       2      4  1212883200

                                               Summary  \
          0            Bought This for My Son at College
          1  Pure cocoa taste with crunchy almonds inside

                                                       Text
          0  My son loves spaghetti so I didn't hesitate or...
          1  It was almost a 'love at first bite' - the per...
```

```
In [229]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [230]: #Before starting the next phase of preprocessing lets see the number of entries left
          print(final.shape)

          #How many positive and negative reviews are present in our dataset?
          final['Score'].value_counts()
```

```
(364171, 10)
```

```
Out[230]: 1    307061
          0     57110
          Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [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 [231]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alo
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I pre
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only t
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of thi
==================================================
```

In [232]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
```

```
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alon
```

```
In [233]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-al
          from bs4 import BeautifulSoup

          soup = BeautifulSoup(sent_0, 'lxml')
          text = soup.get_text()
          print(text)
          print("="*50)

          soup = BeautifulSoup(sent_1000, 'lxml')
          text = soup.get_text()
          print(text)
          print("="*50)

          soup = BeautifulSoup(sent_1500, 'lxml')
          text = soup.get_text()
          print(text)
          print("="*50)

          soup = BeautifulSoup(sent_4900, 'lxml')
          text = soup.get_text()
          print(text)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alon
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I pre
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only t
==================================================
Can't do sugar.  Have tried scores of SF Syrups.  NONE of them can touch the excellence of thi
```

```
In [234]: # https://stackoverflow.com/a/47091490/4084039
          import re

          def decontracted(phrase):
              # specific
              phrase = re.sub(r"won't", "will not", phrase)
              phrase = re.sub(r"can\'t", "can not", phrase)

              # general
              phrase = re.sub(r"n\'t", " not", phrase)
```

```
            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
```

In [235]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Great ingredients although, chicken should have been 1st rather than chicken broth, the only th
==================================================
```

In [236]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving alon
```

In [237]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Great ingredients although chicken should have been 1st rather than chicken broth the only thin
```

In [238]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselv
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', '
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've",
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
```

```
                    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
                    'won', "won't", 'wouldn', "wouldn't"])
```

In [239]:
```
# Sampling the data
final = final.sample(n=100000)
```

In [240]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopw
    preprocessed_reviews.append(sentance.strip())
```

```
100%|| 100000/100000 [00:56<00:00, 1771.53it/s]
```

In [241]:
```
preprocessed_reviews[1500]
```

Out[241]:
```
'love tea drink time taste nice not even sweeten use lemon almost sweet taste take al
```

[3.2] Preprocessing Review Summary

In [242]:
```
## Preprocessing for review summary.
```

In [243]:
```
## Similarly you can do preprocessing for review summary also.

# Combining all the above stundents
from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for summary in tqdm(final['Summary'].values):
    summary = re.sub(r"http\S+", "", summary)
    summary = BeautifulSoup(summary, 'lxml').get_text()
    summary = decontracted(summary)
    summary = re.sub("\S*\d\S*", "", summary).strip()
    summary = re.sub('[^A-Za-z]+', ' ', summary)
    # https://gist.github.com/sebleier/554280
    summary = ' '.join(e.lower() for e in summary.split() if e.lower() not in stopwo

    preprocessed_summary.append(summary.strip())
```

```
100%|| 100000/100000 [00:36<00:00, 2728.83it/s]
```

```
In [244]: final['CleanedText'] = preprocessed_reviews #adding a column of CleanedText which di
          final['CleanedText'] = final['CleanedText'].astype('str')

          final['CleanedSummary'] = preprocessed_summary #adding a column of CleanedSummary wh
          final['CleanedSummary'] = final['CleanedSummary'].astype('str')

          final['Text_Summary'] = final['CleanedSummary'] + final['CleanedText']

          # # store final table into an SQlLite table for future.
          # conn = sqlite3.connect('final.sqlite')
          # c=conn.cursor()
          # conn.text_factory = str
          # final.to_sql('Reviews', conn,  schema=None, if_exists='replace', \
          #               index=True, index_label=None, chunksize=None, dtype=None)
          # conn.close()
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [245]: # #BoW
          # count_vect = CountVectorizer() #in scikit-learn
          # count_vect.fit(preprocessed_reviews)
          # print("some feature names ", count_vect.get_feature_names()[:10])
          # print('='*50)

          # final_counts = count_vect.transform(preprocessed_reviews)
          # print("the type of count vectorizer ",type(final_counts))
          # print("the shape of out text BOW vectorizer ",final_counts.get_shape())
          # print("the number of unique words ", final_counts.get_shape()[1])
```

5.2 [4.2] Bi-Grams and n-Grams.

```
In [246]: # #bi-gram, tri-gram and n-gram

          # #removing stop words like "not" should be avoided before building n-grams
          # # count_vect = CountVectorizer(ngram_range=(1,2))
          # # please do read the CountVectorizer documentation http://scikit-learn.org/stable/

          # # you can choose these numebrs min_df=10, max_features=5000, of your choice
          # count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
          # final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
          # print("the type of count vectorizer ",type(final_bigram_counts))
          # print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
          # print("the number of unique words including both unigrams and bigrams ", final_big
```

5.3 [4.3] TF-IDF

In [247]:
```
# tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
# tf_idf_vect.fit(preprocessed_reviews)
# print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_na
# print('='*50)

# final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
# print("the type of count vectorizer ",type(final_tf_idf))
# print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
# print("the number of unique words including both unigrams and bigrams ", final_tf_
```

5.4 [4.4] Word2Vec

In [248]:
```
# # Train your own Word2Vec model using your own text corpus
# i=0
# list_of_sentance=[]
# for sentance in preprocessed_reviews:
#     list_of_sentance.append(sentance.split())
```

In [249]:
```
# # Using Google News Word2Vectors

# # in this project we are using a pretrained model by google
# # its 3.3G file, once you load this into your memory
# # it occupies ~9Gb, so please do this step only if you have >12G of ram
# # we will provide a pickle file wich contains a dict ,
# # and it contains all our courpus words as keys and  model[word] as values
# # To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# # from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# # it's 1.9GB in size.


# # http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# # you can comment this whole cell
# # or change these varible according to your need

# is_your_ram_gt_16g=False
# want_to_use_google_w2v = False
# want_to_train_w2v = True

# if want_to_train_w2v:
#     # min_count = 5 considers only words that occured atleast 5 times
#     w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
#     print(w2v_model.wv.most_similar('great'))
#     print('='*50)
#     print(w2v_model.wv.most_similar('worst'))

# elif want_to_use_google_w2v and is_your_ram_gt_16g:
#     if os.path.isfile('GoogleNews-vectors-negative300.bin'):
```

```
#         w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative30
#         print(w2v_model.wv.most_similar('great'))
#         print(w2v_model.wv.most_similar('worst'))
#     else:
#         print("you don't have gogole's word2vec file, keep want_to_train_w2v = Tru
```

In [250]:
```
# w2v_words = list(w2v_model.wv.vocab)
# print("number of words that occured minimum 5 times ",len(w2v_words))
# print("sample words ", w2v_words[0:50])
```

5.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [251]:
```
# # average Word2Vec
# # compute average word2vec for each review.
# sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
# for sent in tqdm(list_of_sentance): # for each review/sentence
#     sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might nee
#     cnt_words =0; # num of words with a valid vector in the sentence/review
#     for word in sent: # for each word in a review/sentence
#         if word in w2v_words:
#             vec = w2v_model.wv[word]
#             sent_vec += vec
#             cnt_words += 1
#     if cnt_words != 0:
#         sent_vec /= cnt_words
#     sent_vectors.append(sent_vec)
# print(len(sent_vectors))
# print(len(sent_vectors[0]))
```

[4.4.1.2] TFIDF weighted W2v

In [252]:
```
# # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
# model = TfidfVectorizer()
# tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# # we are converting a dictionary with word as a key, and the idf as a value
# dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [253]:
```
# # TF-IDF weighted Word2Vec
# tfidf_feat = model.get_feature_names() # tfidf words/col-names
# # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tf

# tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in thi
# row=0;
# for sent in tqdm(list_of_sentance): # for each review/sentence
#     sent_vec = np.zeros(50) # as word vectors are of zero length
#     weight_sum =0; # num of words with a valid vector in the sentence/review
#     for word in sent: # for each word in a review/sentence
```

```
#         if word in w2v_words and word in tfidf_feat:
#             vec = w2v_model.wv[word]
# #             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
#             # to reduce the computation we are
#             # dictionary[word] = idf value of word in whole courpus
#             # sent.count(word) = tf valeus of word in this review
#             tf_idf = dictionary[word]*(sent.count(word)/len(sent))
#             sent_vec += (vec * tf_idf)
#             weight_sum += tf_idf
#     if weight_sum != 0:
#         sent_vec /= weight_sum
#     tfidf_sent_vectors.append(sent_vec)
#     row += 1
```

6 [5] Assignment 4: Apply Naive Bayes

Apply Multinomial NaiveBayes on these feature sets
SET 1:Review text, preprocessed one converted into vectors using (BOW)
SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
The hyper paramter tuning(find best Alpha)
Find the best hyper parameter which will give the maximum AUC value
Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
Find the best hyper paramter using k-fold cross validation or simple cross validation data
Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

```
</ul>
</li>
<br>
<li><strong>Feature importance</strong>
    <ul>
<li>Find the top 10 features of positive class and top 10 features of negative class for both f
    </ul>
</li>
<br>
<li><strong>Feature engineering</strong>
    <ul>
<li>To increase the performance of your model, you can also experiment with with feature engine
        <ul>
        <li>Taking length of reviews as another feature.</li>
        <li>Considering some features from review summary as well.</li>
    </ul>
    </ul>
</li>
<br>
<li><strong>Representation of results</strong>
    <ul>
```

```
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

7 Applying Multinomial Naive Bayes

7.1 [5.1] Applying Naive Bayes on BOW, SET 1

In [254]:
```
# Please write all the code with proper documentation
```

In [255]:
```
# Source: https://docs.python.org/3/library/pickle.html

# Saving data to pickle file
def topicklefile(obj, file_name):
    pickle.dump(obj,open(file_name+'.pkl', 'wb'))
```

In [256]:
```
# Data from pickle file
def frompicklefile(file_name):
    data = pickle.load(open(file_name+'.pkl', 'rb'))
    return data
```

In [257]:
```
# Sort 'Time' column
final = final.sort_values(by='Time', ascending=True)
```

In [258]:
```
# Source: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.

# Train Test split for train and test data
X = np.array(final['Text_Summary'])
```

```
y = np.array(final['Score'])

# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state
```

In [259]:
```
topicklefile(X_train, 'X_train')
topicklefile(X_test, 'X_test')
topicklefile(y_train, 'y_train')
topicklefile(y_test, 'y_test')
```

In [260]:
```
# Applying BOW on train and test data and creating the
from sklearn.preprocessing import StandardScaler
from scipy.sparse import hstack

#Standardize 'bow_train' data features by removing the mean and scaling to unit vari
std_scalar = StandardScaler(copy=True, with_mean=False, with_std=True)

def apply_vectorizers_train_test(scenario, model_name, train_data, test_data):

    if model_name == 'BOW':
        #Applying BoW on Train data
        count_vect = CountVectorizer()

    elif model_name == 'TF-IDF':
        #Applying TF-IDF on Train data
        count_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)

    else:
        #Error Message
        print('Model specified is not valid! Please check.')

    #Applying BoW on Test data
    train_vect = count_vect.fit_transform(train_data)

    #Applying BoW on Test data similar to the bow_train data
    test_vect = count_vect.transform(test_data)

    # Standardise train data
    train_vect = std_scalar.fit_transform(train_vect)
    # Standardize the unseen bow_test data
    test_vect = std_scalar.transform(test_vect)


    if scenario == 'With Feature Engg.':
        train_data_list = train_data.tolist()
        df_train_data = pd.DataFrame({'train_data':train_data_list})
        sent_len_train = df_train_data['train_data'].str.split().apply(len)
```

```
        # Source: https://stackoverflow.com/questions/41927781/adding-pandas-columns
        train_vect = hstack((train_vect,np.array(sent_len_train)[:,None]))

        test_data_list = test_data.tolist()
        df_test_data = pd.DataFrame({'test_data':test_data_list})
        sent_len_test = df_test_data['test_data'].str.split().apply(len)

        test_vect = hstack((test_vect,np.array(sent_len_test)[:,None]))

    elif scenario == 'Without Feature Engg.':
        train_vect = train_vect
        test_vect = test_vect
    else:
        print('Invalid scenario specified')

    # Standardise train data
    train_vect = std_scalar.fit_transform(train_vect)
    # Standardize the unseen bow_test data
    test_vect = std_scalar.transform(test_vect)


    topicklefile(train_vect, 'train_vect')
    topicklefile(test_vect, 'test_vect')

    print("'train_vect' and 'test_vect' are the pickle files.")

    return count_vect
```

In [261]:
```
def applying_naive_bayes(alphas, train_data, y_train):

    parameters = {'alpha':alphas}
    nb_clf = MultinomialNB(fit_prior=True, class_prior=None)
    clf = GridSearchCV(nb_clf, parameters, cv=10, scoring= 'roc_auc', n_jobs=4, retu
    clf.fit(train_data, y_train)

    alpha_optimal = clf.best_params_.get('alpha')

    train_auc= clf.cv_results_['mean_train_score']
    train_auc_std= clf.cv_results_['std_train_score']
    cv_auc = clf.cv_results_['mean_test_score']
    cv_auc_std= clf.cv_results_['std_test_score']

    return clf, alpha_optimal, train_auc, train_auc_std, cv_auc, cv_auc_std
```

In [262]:
```
def train_cv_error_plot(train_auc, train_auc_std, cv_auc, cv_auc_std):
    plt.plot(alpha_values, train_auc, label='Train AUC')

    # Source: https://stackoverflow.com/a/48803361/4084039
```

```
        plt.gca().fill_between(alpha_values,train_auc - train_auc_std,train_auc + train_a

        plt.plot(alpha_values, cv_auc, label='CV AUC')
        # Source: https://stackoverflow.com/a/48803361/4084039
        plt.gca().fill_between(alpha_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha
        plt.legend()
        plt.xlabel("Alpha: hyperparameter")
        plt.ylabel("AUC")
        plt.title("ERROR PLOTS")
        plt.show()
```

```
In [263]: # instantiate learning model K = optimal_k
          def naive_bayes_optimal(optimal_alpha):
              nb_optimal = MultinomialNB(alpha=optimal_alpha)
              return nb_optimal
```

```
In [264]: def retrain_naive_bayes(nb_optimal, train_vec, y_train, test_vec, y_test):

              # fitting the model with optimal K for training data
              nb_optimal.fit(train_vec, y_train)

              # predict the response for the unseen bow_test data
              y_pred = nb_optimal.predict(test_vec)
```

```
In [265]: # Confusion Matrix
          def cm_fig(nb_optimal, y_test, test_vec):
              cm = pd.DataFrame(confusion_matrix(y_test, nb_optimal.predict(test_vec)))
              # print(confusion_matrix(y_test, y_pred))

              plt.figure(1, figsize=(18,5))
              plt.subplot(121)
              plt.title("Confusion Matrix")
              sns.set(font_scale=1.4)
              sns.heatmap(cm, cmap= 'gist_earth', annot=True, annot_kws={'size':15}, fmt='g')
```

```
In [266]: #Reference: https://stackoverflow.com/questions/52910061/implementing-roc-curves-for
          def error_plot(nb_optimal, train_vec, y_train, test_vec, y_test):
              train_fpr, train_tpr, thresholds = roc_curve(y_train, nb_optimal.predict_proba(tr
              test_fpr, test_tpr, thresholds = roc_curve(y_test, nb_optimal.predict_proba(test_

              plt.plot(train_fpr, train_tpr, label="train AUC = %0.3f" %auc(train_fpr, train_tp
              plt.plot(test_fpr, test_tpr, label="train AUC = %0.3f" %auc(test_fpr, test_tpr))
              plt.plot([0.0, 1.0], [0.0, 1.0],'k--')
              plt.legend()
              plt.xlabel("Alpha: hyperparameter")
              plt.ylabel("AUC")
              plt.title("ERROR PLOTS")
              plt.show()
```

```
            return auc(test_fpr, test_tpr)
```

```
In [267]: def get_features_top(count_vect, nb_optimal):
              '''# creating a dataframe for with the features and their probability scores
              df_feature_proba = pd.DataFrame(nb_optimal.feature_log_prob_,columns=count_vect.g
              df_feature_proba = df_feature_proba.T.reset_index()
              print(df_feature_proba.columns)
              df_feature_proba.rename(columns={'index':'features', '0':'Negative', '1':'Positi
              df_sort_0 = df_feature_proba.sort_values(by=[0])
              df_sort_1 = df_feature_proba.sort_values(by=[1])
              df_sort_1[['features',1]].head(10), df_sort_0[['features',0]].head(10)'''

              feat=count_vect.get_feature_names()
              neg_prob=nb_optimal.feature_log_prob_[0,:]
              pos_prob=nb_optimal.feature_log_prob_[1,:]
              sorted_neg_prob_feat=sorted(zip(neg_prob,feat),reverse=True)
              sorted_pos_prob_feat=sorted(zip(pos_prob,feat),reverse=True)
          #     print(sorted_neg_prob_feat[:10])
          #     print(sorted_pos_prob_feat[:10])

              df_feature_proba = pd.DataFrame({'0':sorted_neg_prob_feat[:10], '1':sorted_pos_pr
          #     df_feature_proba[['prob_score_pos', 'Feature']] = pd.DataFrame(df_feature_proba
          #     df_feature_proba[['prob_score_neg', 'Feature']] = pd.DataFrame(df_feature_proba
              return df_feature_proba['0'], df_feature_proba['1']
```

```
In [268]: X_train = frompicklefile('X_train')
          X_test = frompicklefile('X_test')
          count_vect = apply_vectorizers_train_test('Without Feature Engg.','BOW', X_train, X_t
```

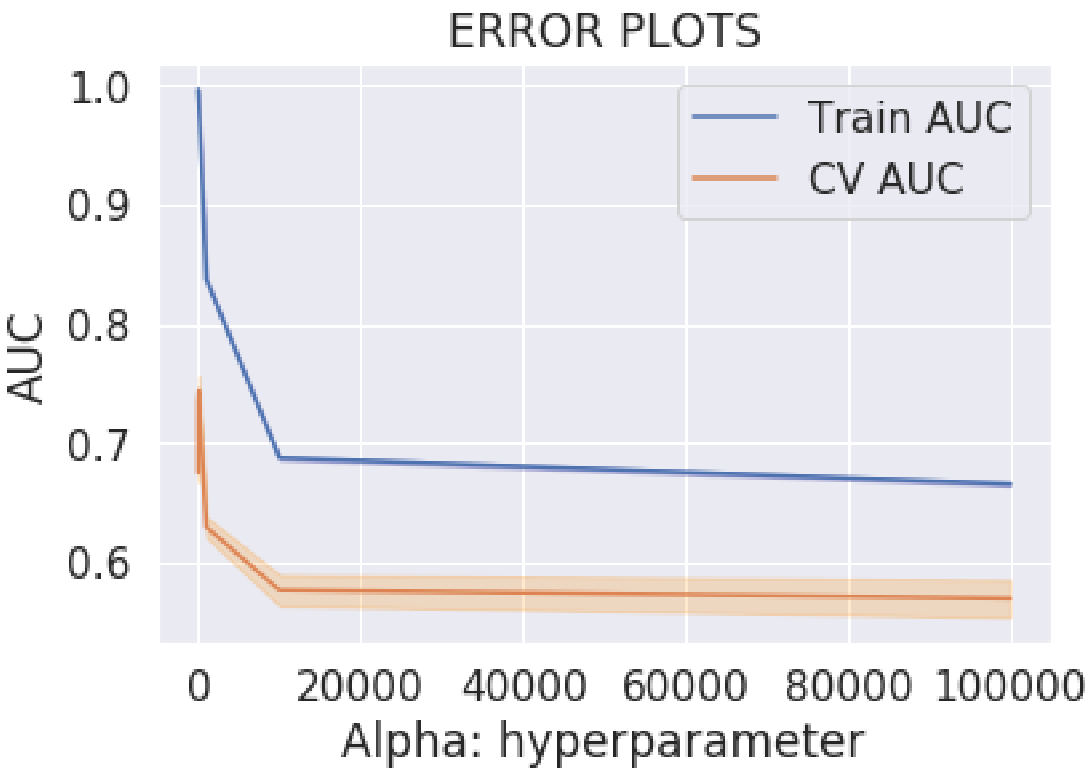
```
'train_vect' and 'test_vect' are the pickle files.
```

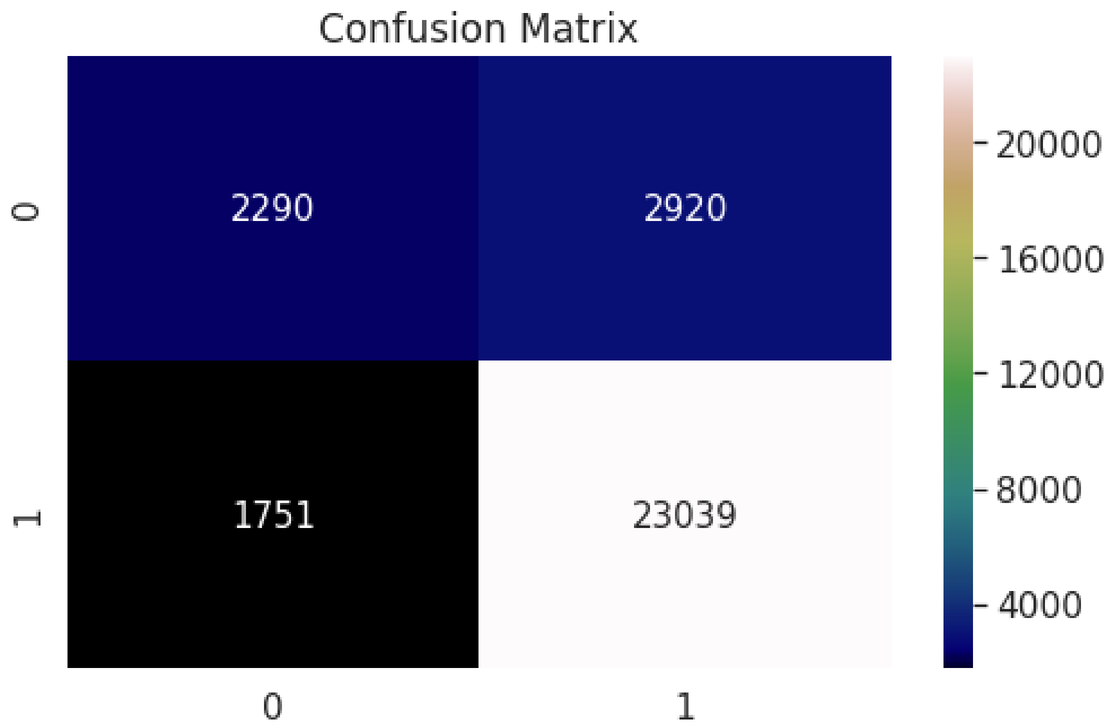
```
In [269]: train_vect = frompicklefile('train_vect')
          test_vect = frompicklefile('test_vect')
          y_train = frompicklefile('y_train')
          y_test = frompicklefile('y_test')
```

```
In [270]: alpha_values = [10**-5,10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4
          clf, optimal_alpha, train_auc, train_auc_std, cv_auc, cv_auc_std = applying_naive_bay

          optimal_alpha_bow1 = optimal_alpha

          print('The optimal alpha is {}' .format(optimal_alpha))

          train_cv_error_plot(train_auc, train_auc_std, cv_auc, cv_auc_std)

          nb_optimal = naive_bayes_optimal(optimal_alpha)
```

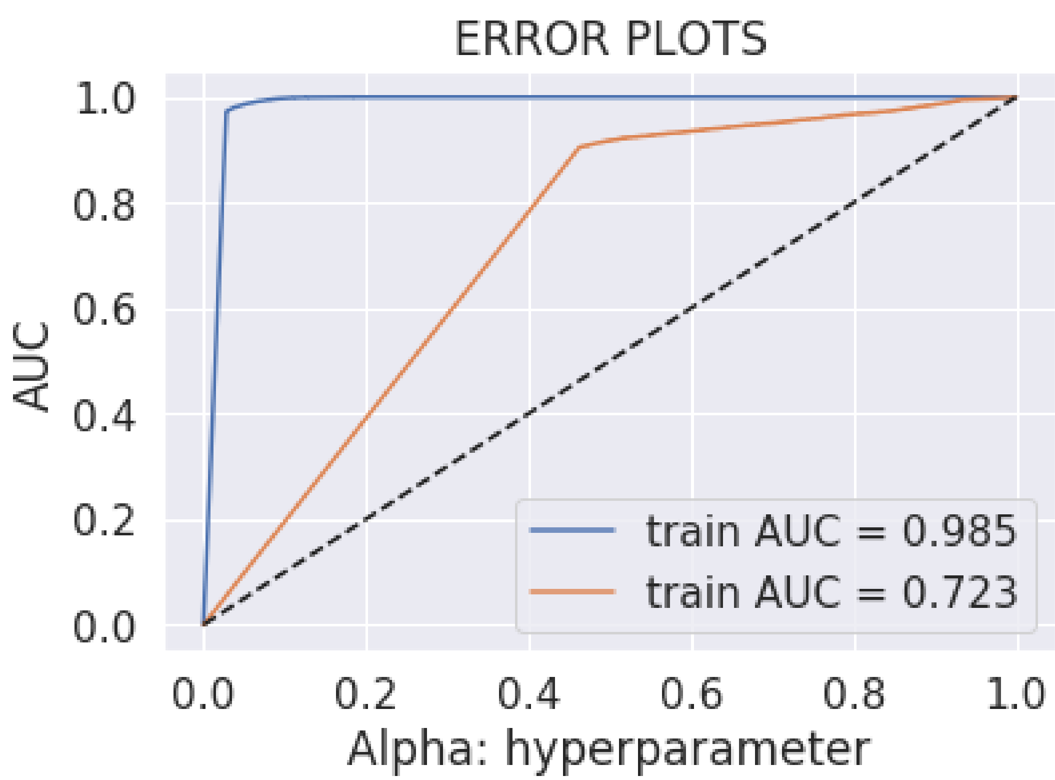
```
retrain_naive_bayes(nb_optimal, train_vect, y_train, test_vect, y_test)

cm_fig(nb_optimal, y_test, test_vect)
```

```
The optimal alpha is 100
```






```
In [271]: bow_auc1 = error_plot(nb_optimal, train_vect, y_train, test_vect, y_test)
```



7.1.1 [5.1.1] Top 10 important features of positive class from SET 1

In [272]:
```
top_10_features_pos, top_10_features_neg = get_features_top(count_vect, nb_optimal)
top_10_features_pos
```

Out[272]:
```
0             (-7.324159324306041, not)
1            (-7.912946955752755, like)
2           (-7.915352271515625, would)
3         (-7.972582844816168, product)
4           (-7.978847821763232, taste)
5             (-8.052958806104584, bad)
6              (-8.160177266855117, no)
7             (-8.176071251058007, one)
8     (-8.213281851437957, disappointed)
9           (-8.213949419331499, money)
Name: 0, dtype: object
```

7.1.2 [5.1.2] Top 10 important features of negative class from SET 1

In [275]:
```
top_10_features_neg
```

Out[275]:
```
0      (-7.276607192892094, great)
1        (-7.285067259197474, not)
2      (-7.4017581044218055, good)
3       (-7.554091584669358, like)
4        (-7.658646999371134, one)
5       (-7.664574487707354, love)
6       (-7.676982929030105, best)
7      (-7.688982779578561, taste)
8     (-7.745656772105399, flavor)
9         (-7.80131673374326, get)
Name: 1, dtype: object
```

7.2 [5.1.2] Feature Engineering- Taking length of reviews as a feature for BOW

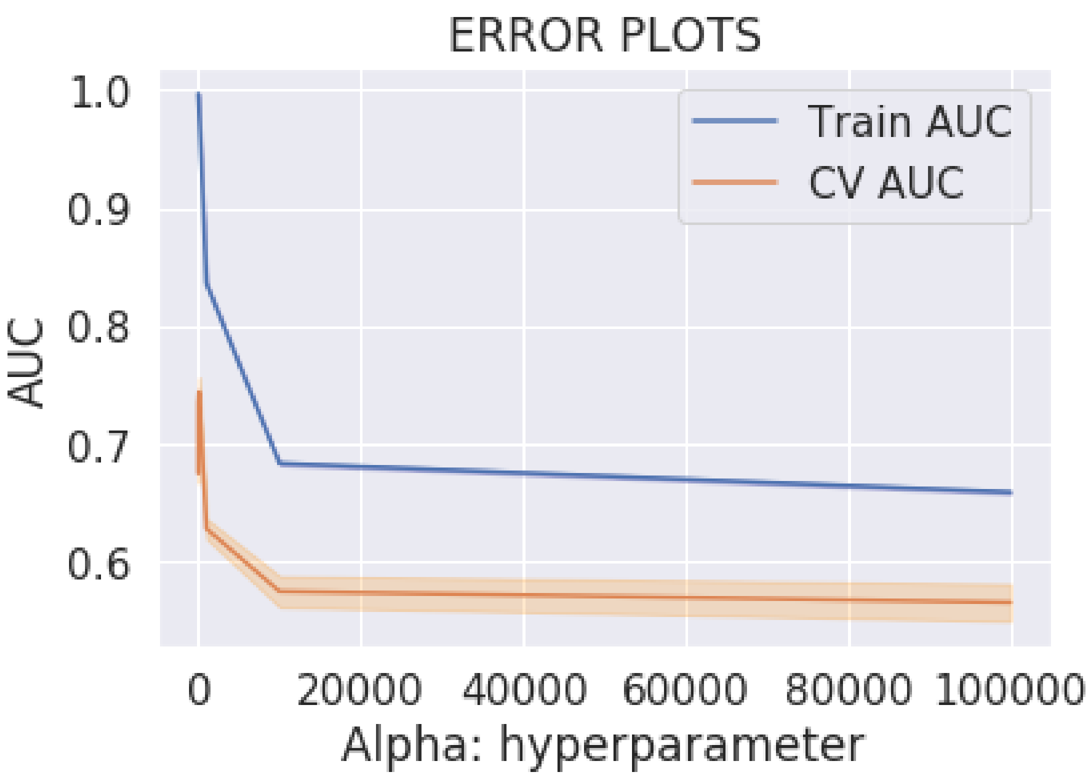
In [276]:
```
X_train = frompicklefile('X_train')
X_test = frompicklefile('X_test')
count_vect = apply_vectorizers_train_test('With Feature Engg.','BOW', X_train, X_tes
```

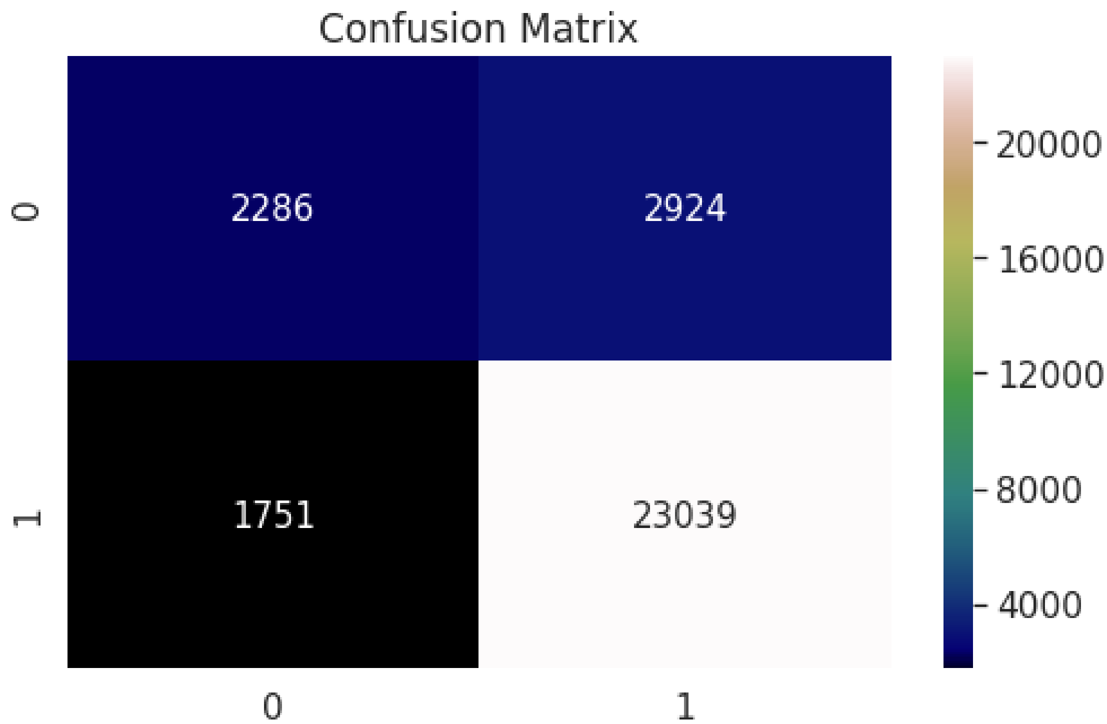
```
'train_vect' and 'test_vect' are the pickle files.
```

In [277]:
```
train_vect = frompicklefile('train_vect')
test_vect = frompicklefile('test_vect')
y_train = frompicklefile('y_train')
y_test = frompicklefile('y_test')
```

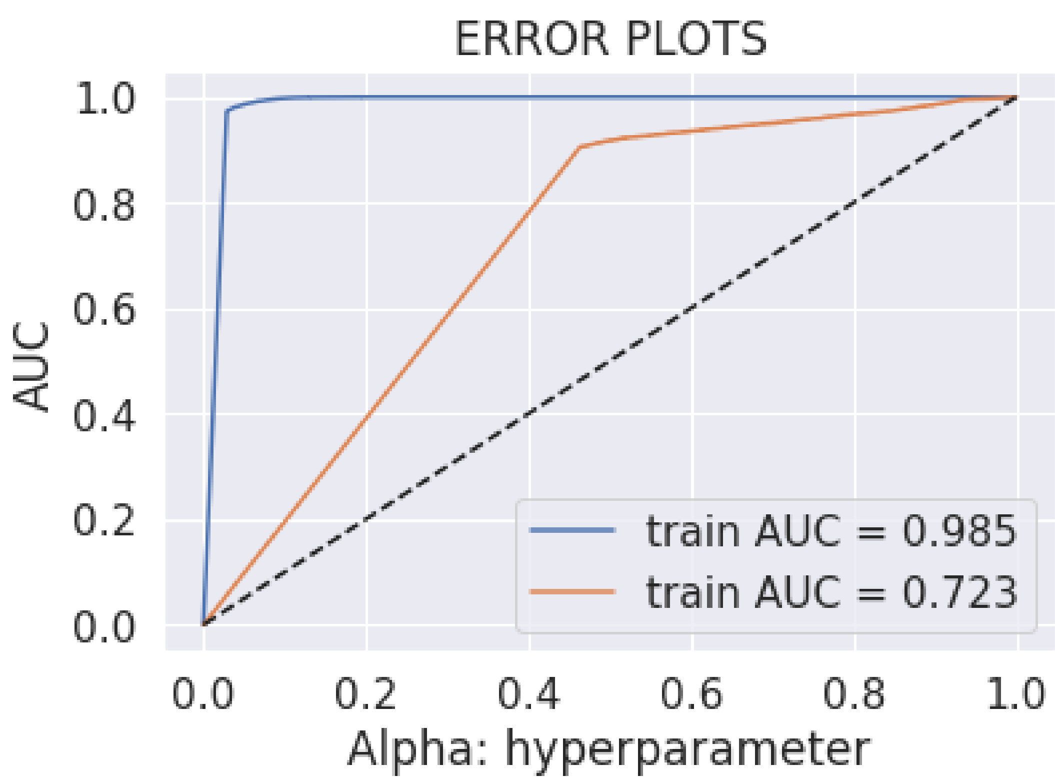
In [278]:
```
alpha_values = [10**-5,10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4
clf, optimal_alpha, train_auc, train_auc_std, cv_auc, cv_auc_std = applying_naive_ba

optimal_alpha_bow2 = optimal_alpha

print('The optimal alpha is {}' .format(optimal_alpha))

train_cv_error_plot(train_auc, train_auc_std, cv_auc, cv_auc_std)

nb_optimal = naive_bayes_optimal(optimal_alpha)

retrain_naive_bayes(nb_optimal, train_vect, y_train, test_vect, y_test)

cm_fig(nb_optimal, y_test, test_vect)
```

```
The optimal alpha is 100
```






```
In [279]: bow_auc2 = error_plot(nb_optimal, train_vect, y_train, test_vect, y_test)
```



7.3 [5.2] Applying Naive Bayes on TFIDF, SET 2

```
In [280]: # Please write all the code with proper documentation
```

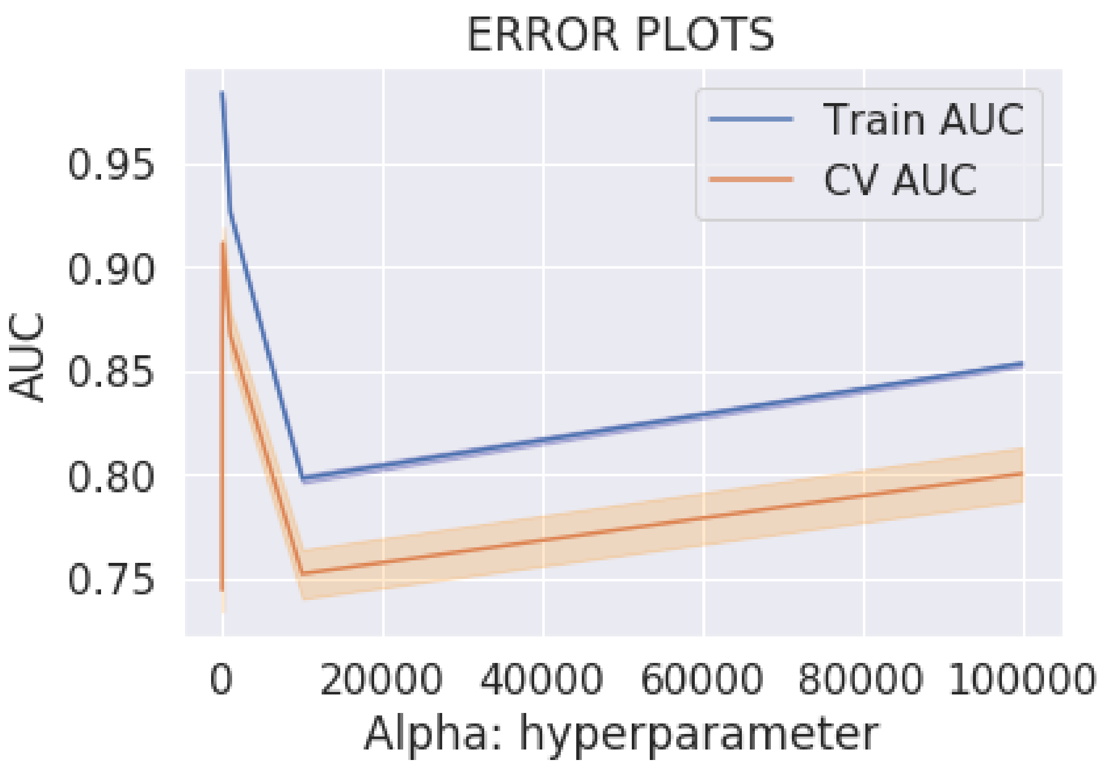
```
In [281]: X_train = frompicklefile('X_train')
          X_test = frompicklefile('X_test')
          count_vect = apply_vectorizers_train_test('Without Feature Engg.','TF-IDF', X_train,
```

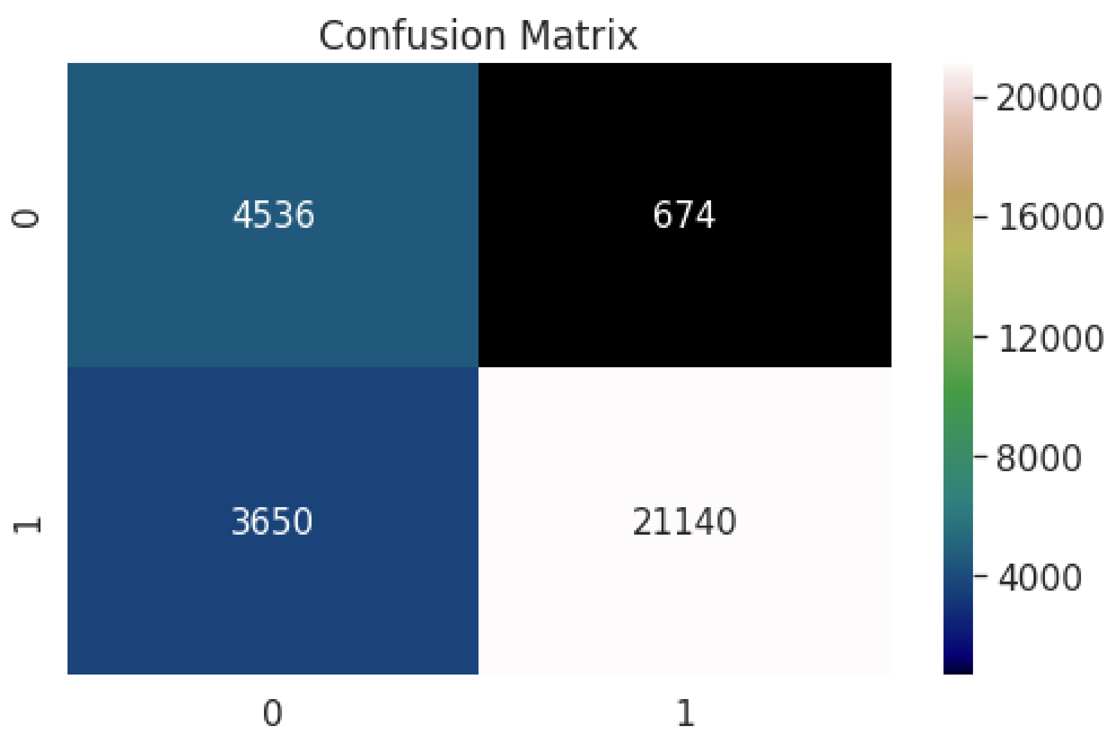
```
'train_vect' and 'test_vect' are the pickle files.
```

```
In [282]: train_vect = frompicklefile('train_vect')
          test_vect = frompicklefile('test_vect')
          y_train = frompicklefile('y_train')
          y_test = frompicklefile('y_test')
```

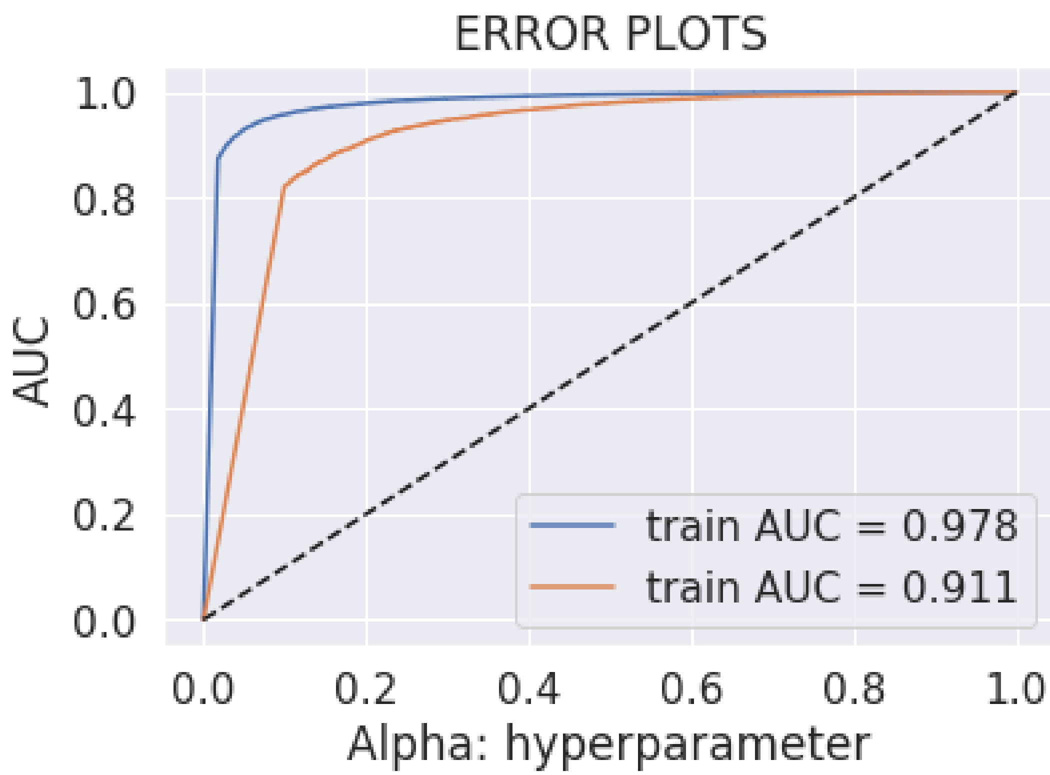
```
In [283]: alpha_values = [10**-5, 10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4
          clf, optimal_alpha, train_auc, train_auc_std, cv_auc, cv_auc_std = applying_naive_bay

          optimal_alpha_tfidf1 = optimal_alpha

          print('The optimal alpha is {}' .format(optimal_alpha))

          train_cv_error_plot(train_auc, train_auc_std, cv_auc, cv_auc_std)

          nb_optimal = naive_bayes_optimal(optimal_alpha)

          retrain_naive_bayes(nb_optimal, train_vect, y_train, test_vect, y_test)

          cm_fig(nb_optimal, y_test, test_vect)
```

```
The optimal alpha is 100
```


ERROR PLOTS
Train AUC
CV AUC
AUC
0.95
0.90
0.85
0.80
0.75
0
20000
40000
60000
80000
100000
Alpha: hyperparameter


Confusion Matrix
0
1
4536
674
3650
21140
0
1
20000
16000
12000
8000
4000

In [284]: tfidf_auc1 = error_plot(nb_optimal, train_vect, y_train, test_vect, y_test)



7.3.1 [5.2.1] Top 10 important features of positive class from SET 2

```
In [285]: top_10_features_pos, top_10_features_neg = get_features_top(count_vect, nb_optimal)
          top_10_features_pos
```

```
Out[285]: 0           (-6.767984364695124, not)
          1            (-7.5050816264151, like)
          2          (-7.517667149516836, would)
          3         (-7.6029793381607735, taste)
          4        (-7.707923178113731, product)
          5           (-7.7157410859468225, bad)
          6            (-7.810562384259493, no)
          7            (-7.82001830749023, one)
          8     (-7.882861845603019, would not)
          9         (-7.8967955366042055, money)
          Name: 0, dtype: object
```

7.3.2 [5.2.2] Top 10 important features of negative class from SET 2

```
In [286]: top_10_features_neg
```

```
Out[286]: 0        (-7.141960546073776, not)
          1      (-7.3295539087541695, great)
          2        (-7.421438354146275, good)
          3        (-7.525176857210102, like)
          4         (-7.639776867517439, one)
          5       (-7.672688934688452, taste)
          6      (-7.711646226886442, flavor)
          7        (-7.715743563728433, love)
          8        (-7.74196916224692З, best)
          9     (-7.786181522438268, product)
          Name: 1, dtype: object
```

7.4 [5.2.3] Feature Engineering- Taking length of reviews as a feature for TF-IDF

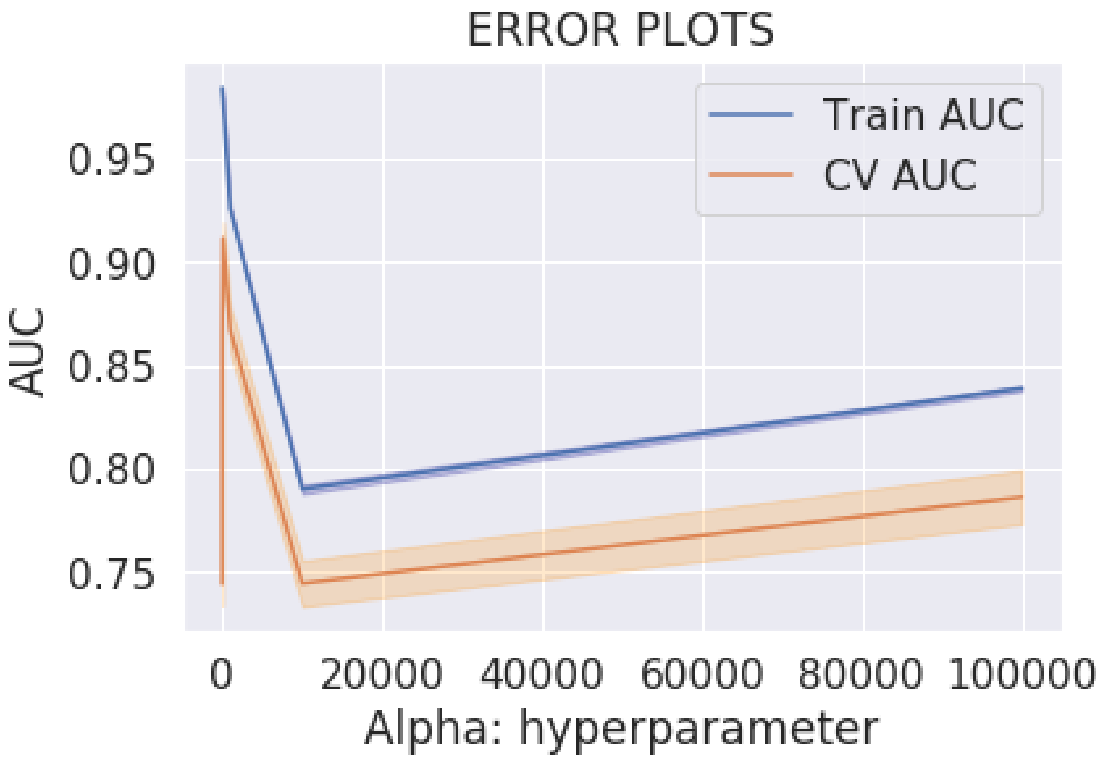
```
In [287]: X_train = frompicklefile('X_train')
          X_test = frompicklefile('X_test')
          count_vect = apply_vectorizers_train_test('With Feature Engg.','TF-IDF', X_train, X_
```

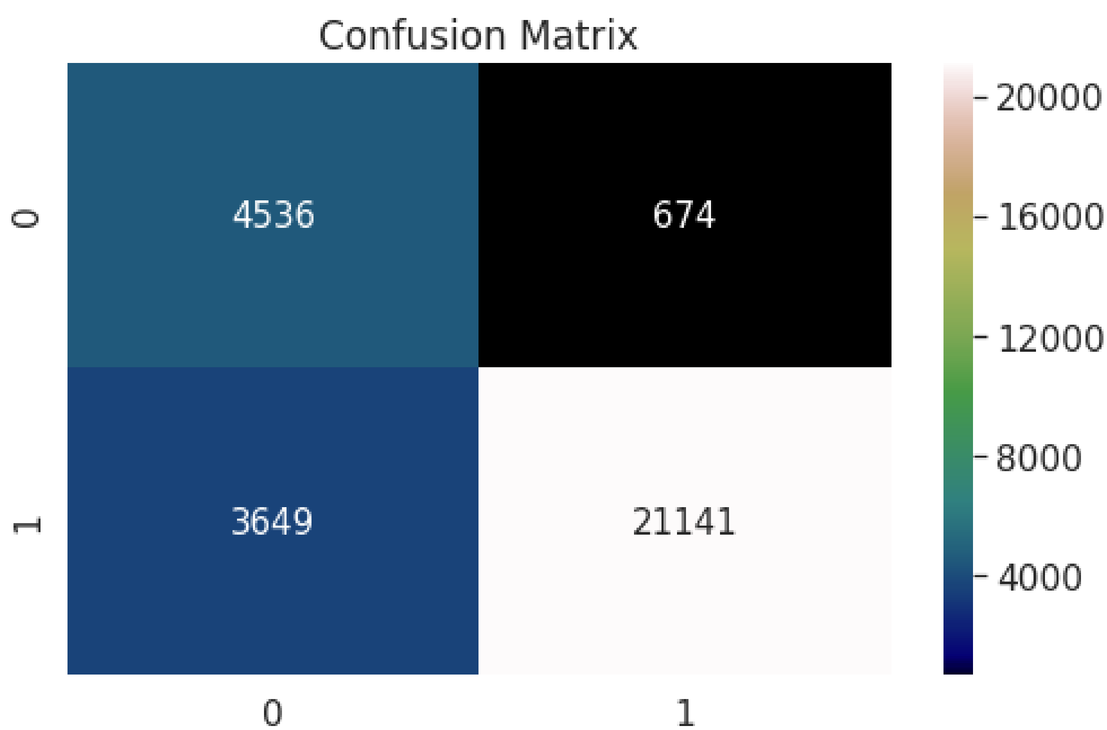
```
'train_vect' and 'test_vect' are the pickle files.
```

```
In [288]: train_vect = frompicklefile('train_vect')
          test_vect = frompicklefile('test_vect')
          y_train = frompicklefile('y_train')
          y_test = frompicklefile('y_test')
```

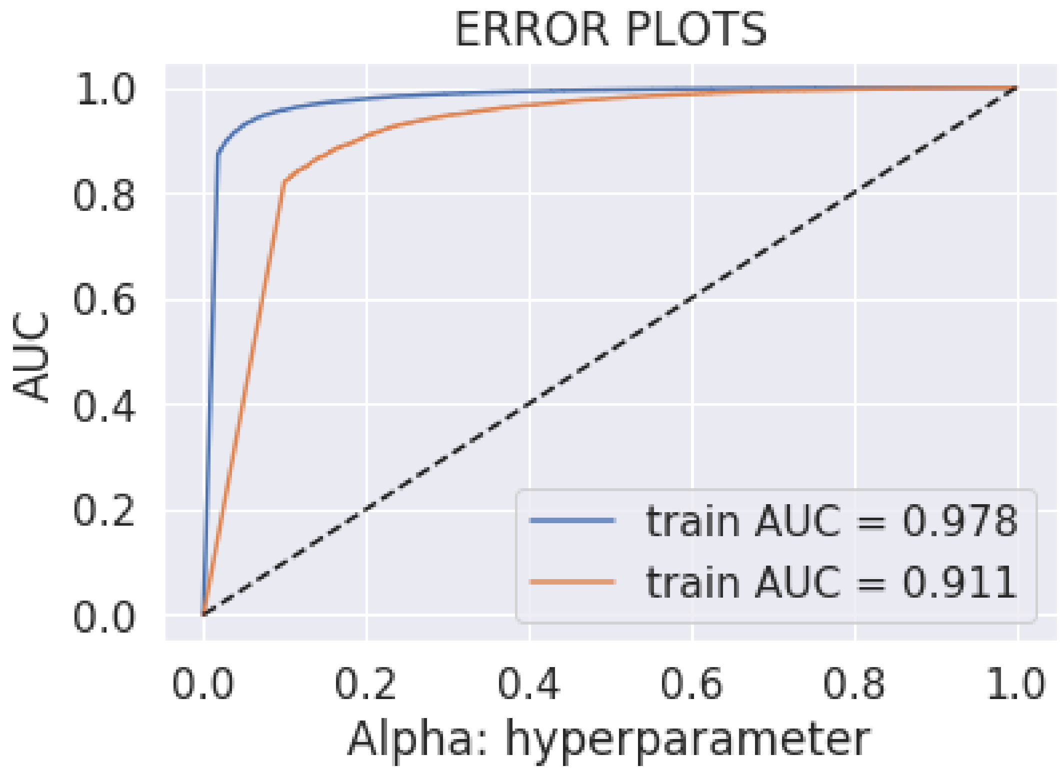
```
In [289]: alpha_values = [10**-5,10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4
          clf, optimal_alpha, train_auc, train_auc_std, cv_auc, cv_auc_std = applying_naive_ba

          optimal_alpha_tfidf2 = optimal_alpha

          print('The optimal alpha is {}' .format(optimal_alpha))

          train_cv_error_plot(train_auc, train_auc_std, cv_auc, cv_auc_std)

          nb_optimal = naive_bayes_optimal(optimal_alpha)

          retrain_naive_bayes(nb_optimal, train_vect, y_train, test_vect, y_test)

          cm_fig(nb_optimal, y_test, test_vect)
```

```
The optimal alpha is 100
```


ERROR PLOTS
Train AUC
CV AUC
0.95
0.90
0.85
0.80
0.75
AUC
0
20000
40000
60000
80000
100000
Alpha: hyperparameter


Confusion Matrix
4536
674
3649
21141
0
1
0
1
20000
16000
12000
8000
4000

In [290]:
```python
tfidf_auc2 = error_plot(nb_optimal, train_vect, y_train, test_vect, y_test)
```



8 [6] Conclusions

In [291]:
```python
#Source: http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

model_metric = PrettyTable()
model_metric = PrettyTable(["Model Name", "Scenario", 'Hyperparameter', 'AUC'])

model_metric.add_row(["Bag of Words","Without Feature Engg.", optimal_alpha_bow1, bo
model_metric.add_row(["TF-IDF","Without Feature Engg.", optimal_alpha_tfidf1, bow_au
model_metric.add_row(["Bag of Words","With Feature Engg.", optimal_alpha_bow2, tfidf
model_metric.add_row(["TF-IDF","With Feature Engg.", optimal_alpha_tfidf2, tfidf_auc

print(model_metric.get_string(start=0, end=5))
```

```
+--------------+-----------------------+----------------+--------------------+
|  Model Name  |        Scenario       | Hyperparameter |        AUC         |
+--------------+-----------------------+----------------+--------------------+
| Bag of Words | Without Feature Engg. |      100       | 0.723260954396973  |
|    TF-IDF    | Without Feature Engg. |      100       | 0.7228750912656718 |
```

Bag of Words	With Feature Engg.	100	0.9112632485236835
TF-IDF	With Feature Engg.	100	0.9112646228317869

8.1 [6.1] Observations:

1. Multinomial Naive Bayes is used for this assignment because the BOW and TF-IDF vectorizers produce counts and fractional counts for the features respectively and it can be used in this case.

2. The time complexity is incredibly low for Naive Bayes.

3. There is a very significantly less or no change in the AUC score of the models with or without the length of sentance as feature as part of Feature Enginerring. The performance of the models remain unchanged.