04 Amazon Fine Food Reviews Analysis_NaiveBayes

May 1, 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 [1]:
```
%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
```

In [3]:
```
# using SQLite Table to read data.
con = sqlite3.connect(r'D:\Sayantan\Personal\MLnAI\Assignments\NaiveBayes\database.sqli

# 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 points
# 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 50
```

```python
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 1000

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negativ
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 (100000, 10)
```

```
Out[3]:    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...
```

```python
In [4]: display = pd.read_sql_query("""
        SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
        FROM Reviews
        GROUP BY UserId
        HAVING COUNT(*)>1
        """, con)
```

```python
In [5]: print(display.shape)
        display.head()
```

```
(80668, 7)
```

```
Out[5]:               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 [6]:
```python
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[6]:
```
              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 [7]:
```python
display['COUNT(*)'].sum()
```

Out[7]:
```
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 [8]:
```python
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[8]:
```
       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 [9]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

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

Out[226]:
```
(87898, 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]:
```
87.898
```

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()
```

```
(87896, 10)
```

Out[230]:
```
1    73686
0    14210
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 [15]: # 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid
==================================================
```

```
In [16]: # 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

```
In [17]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-
         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)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraic
```

In [18]:
```
# 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 [19]:
```
sent_1500 = decontracted(sent_1500)
```

```
    print(sent_1500)
    print("="*50)
```

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

```
In [20]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
         sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
         print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

```
In [21]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
         sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
         print(sent_1500)
```

```
was way to hot for my blood took a bite and did a jig lol
```

```
In [22]: # 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', 'ourselve
                     "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 [231]: # 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%|| 87896/87896 [01:13<00:00, 1196.70it/s]
```

In [232]:
```
preprocessed_reviews[1500]
```

Out[232]:
```
'way hot blood took bite jig lol'
```

[3.2] Preprocessing Review Summary

In [225]:
```
## Similartly you can do preprocessing for review summary also.
#Deduplication of entries
import warnings
warnings.filterwarnings("ignore")
final1=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text","Summ
print(final1.shape)

#Checking to see how much % of data still remains
print((final1['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100)

#value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not p
#hence these  rows too are removed from calcualtions
final1=final1[final1.HelpfulnessNumerator<=final1.HelpfulnessDenominator]

#Before starting the next phase of preprocessing lets see the number of entries left
print(final1.shape)

#How many positive and negative reviews are present in our dataset?
print(final1['Score'].value_counts())

from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final1['Summary'].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_summary.append(sentance.strip())
```

```
(87898, 10)
87.898
(87896, 10)
1    73686
0    14210
Name: Score, dtype: int64


100%|| 87896/87896 [01:03<00:00, 1389.82it/s]
```

In [28]:
```python
preprocessed_summary[32710]
```

Out[28]:
```
'great toddler'
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

In [0]:
```python
#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])
```

```
some feature names  ['aa', 'aahhhs', 'aback', 'abandon', 'abates', 'abbott', 'abby', 'abdomina
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 12997)
the number of unique words  12997
```

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

In [0]:
```python
#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/modu

# 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_bigram_c
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

5.3 [4.3] TF-IDF

```
In [0]: 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_names
        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_idf.g
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able find', 'able get',
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (4986, 3144)
the number of unique words including both unigrams and bigrams  3144
```

5.4 [4.4] Word2Vec

```
In [0]: # 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 [0]: # 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-negative300.bi
        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 = True, t
```

```
[('snack', 0.9951335191726685), ('calorie', 0.9946465492248535), ('wonderful', 0.99460321664810
==================================================
[('varieties', 0.9994194507598877), ('become', 0.9992934465408325), ('popcorn', 0.999275088310
```

In [0]:
```
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])
```

```
number of words that occured minimum 5 times  3817
sample words  ['product', 'available', 'course', 'total', 'pretty', 'stinky', 'right', 'nearby
```

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

[4.4.1.1] Avg W2v

In [0]:
```
# 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 need to
    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]))
```

```
100%|| 4986/4986 [00:03<00:00, 1330.47it/s]


4986
50
```

[4.4.1.2] TFIDF weighted W2v

```
In [0]: # 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 [0]: # 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 = tfidf

        tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this li
        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
```

```
100%|| 4986/4986 [00:20<00:00, 245.63it/s]
```

6 [5] Assignment 4: Apply Naive Bayes

```
<li><strong>Apply Multinomial NaiveBayes on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vector
    </ul>
</li>
<br>
<li><strong>The hyper paramter tuning(find best Alpha)</strong>
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001</l
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this t
    </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
    </ul>
</li>
<br>
<li><strong>Feature engineering</strong>
    <ul>
<li>To increase the performance of your model, you can also experiment with with feature engin
        <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 f
<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.
<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

```
In [233]: #common data for following calculations
          #total data length taken 100K, as instructed in the assigment and then it got prepro
          print("Data length after preprocessing: ",len(preprocessed_reviews))

          Y = final['Score'].values
          X = np.asarray(preprocessed_reviews)

          print(type(X))
          print(type(Y))

          #Splitting data in train, cv and test
          #error calc code taken from https://stackabuse.com/k-nearest-neighbors-algorithm-in-
          from sklearn.model_selection import train_test_split
          from sklearn.metrics import roc_auc_score

          X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20,shuffle=Fal
          X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.20,shu
          print(X_train.shape, y_train.shape)
          print(X_cv.shape, y_cv.shape)
          print(X_test.shape, y_test.shape)
```

```
Data length after preprocessing:  87896
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
(56252,) (56252,)
(14064,) (14064,)
(17580,) (17580,)
```

```
In [234]: #Defination of the functions used for analysis in the following sections
          from sklearn.naive_bayes import MultinomialNB
          alpha_val = [0.00001, 0.0001, 0.001, 0.01, 0.1, 0, 1, 10, 100, 1000, 10000, 100000]
          alpha_idx = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

```
def getOptimalAlpha(X_train, y_train, X_cv, y_cv):
    train_auc = []
    cv_auc = []
    for i in alpha_val:
        multiNomialNB = MultinomialNB(alpha=i)
        multiNomialNB.fit(X_train, y_train)
        y_train_pred =  multiNomialNB.predict_proba(X_train)[:,1]
        y_cv_pred =  multiNomialNB.predict_proba(X_cv)[:,1]

        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

    plt.plot(alpha_idx, train_auc, label='Train AUC')
    plt.scatter(alpha_idx, train_auc, label='Train AUC')
    plt.xticks(alpha_idx,alpha_val, rotation=45)
    plt.plot(alpha_idx, cv_auc, label='CV AUC')
    plt.scatter(alpha_idx, cv_auc, label='CV AUC')
    plt.legend()
    plt.xlabel("alpha: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()

    return multiNomialNB
```

In [235]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

def getNBAnalysis(alphaVal, X_train1, y_train1, X_test1, y_test1):
    multiNomialNB = MultinomialNB(alpha=alphaVal)
    multiNomialNB.fit(X_train1, y_train1)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimat
    # not the predicted outputs

    train_fpr, train_tpr, thresholds = roc_curve(y_train1, multiNomialNB.predict_prol
    test_fpr, test_tpr, thresholds = roc_curve(y_test1, multiNomialNB.predict_proba(

    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.xlabel("K: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()

    train_conf_matix = confusion_matrix(y_train1, multiNomialNB.predict(X_train1))
    test_conf_matrix = confusion_matrix(y_test1, multiNomialNB.predict(X_test1))
```

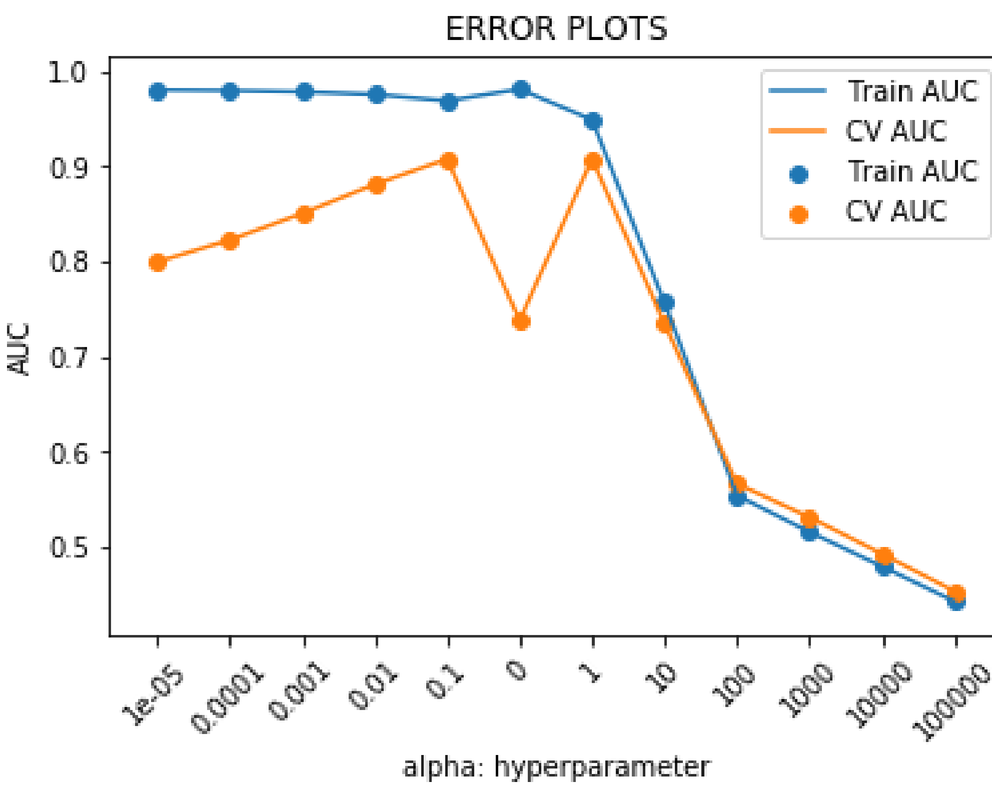
```
        return(train_conf_matix, test_conf_matrix)
```

In [236]:
```
def showConfusionMatrix(confMatrix, titleText):
    print(confMatrix)
    df_train = pd.DataFrame(confMatrix, index=["-ve", "+ve"],columns=["-ve", "+ve"])
    sns.heatmap(df_train, annot=True, fmt='d')
    plt.title(titleText)
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()
```

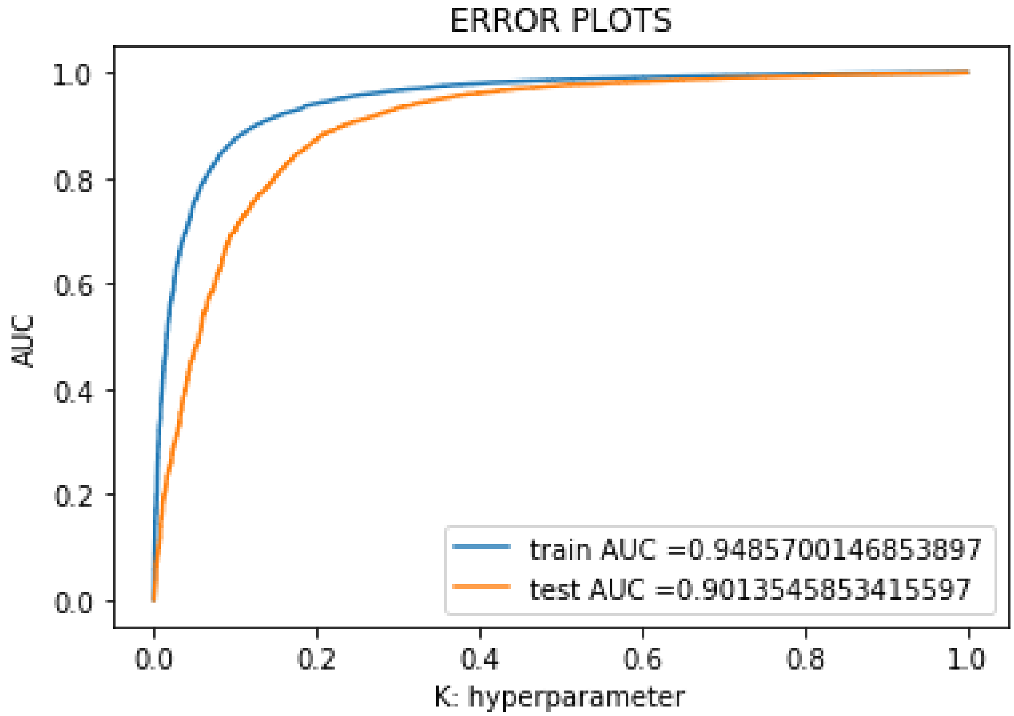
7.1 [5.1] Applying Naive Bayes on BOW, SET 1

In [237]:
```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
#applying the method fit_transform() on you train data, and apply the method transfo
X_train_bow = vectorizer.fit_transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)
```

In [238]:
```
modelNB = getOptimalAlpha(X_train_bow, y_train, X_cv_bow, y_cv)
```



```
In [239]: #Max AUC of for Alpha value 1
          train_confusion_matrix, test_confusion_matrix = getNBAnalysis(1, X_train_bow, y_trai
```

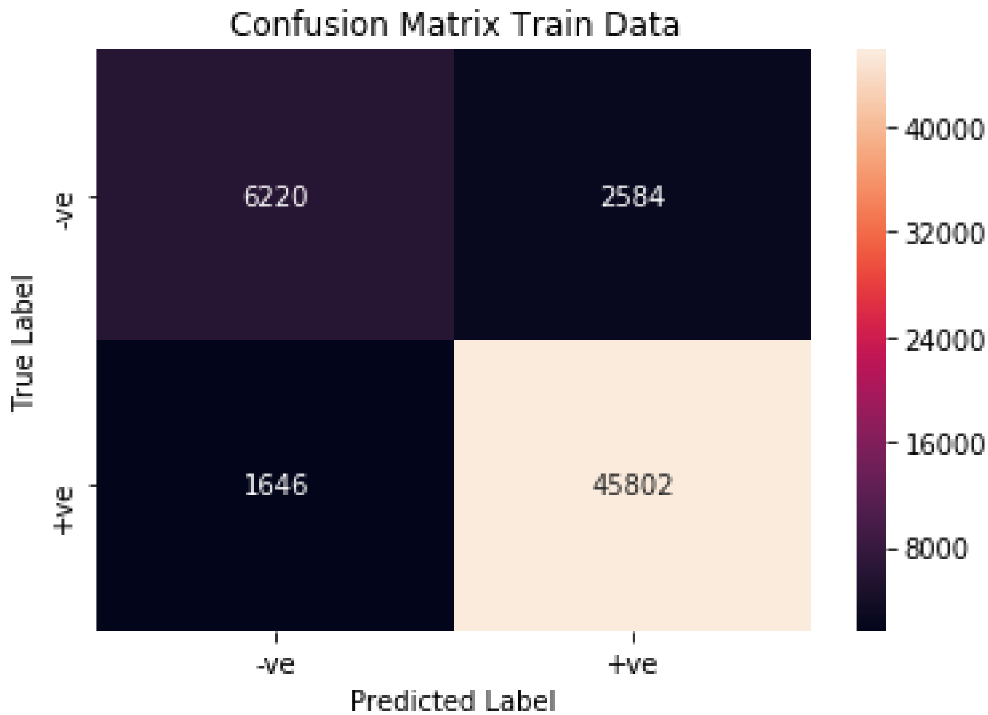



```
In [240]: print(train_confusion_matrix)
          print(type(train_confusion_matrix))

          print(test_confusion_matrix)
          print(type(test_confusion_matrix))
```

```
[[ 6220  2584]
 [ 1646 45802]]
<class 'numpy.ndarray'>
[[ 1781  1246]
 [  537 14016]]
<class 'numpy.ndarray'>
```

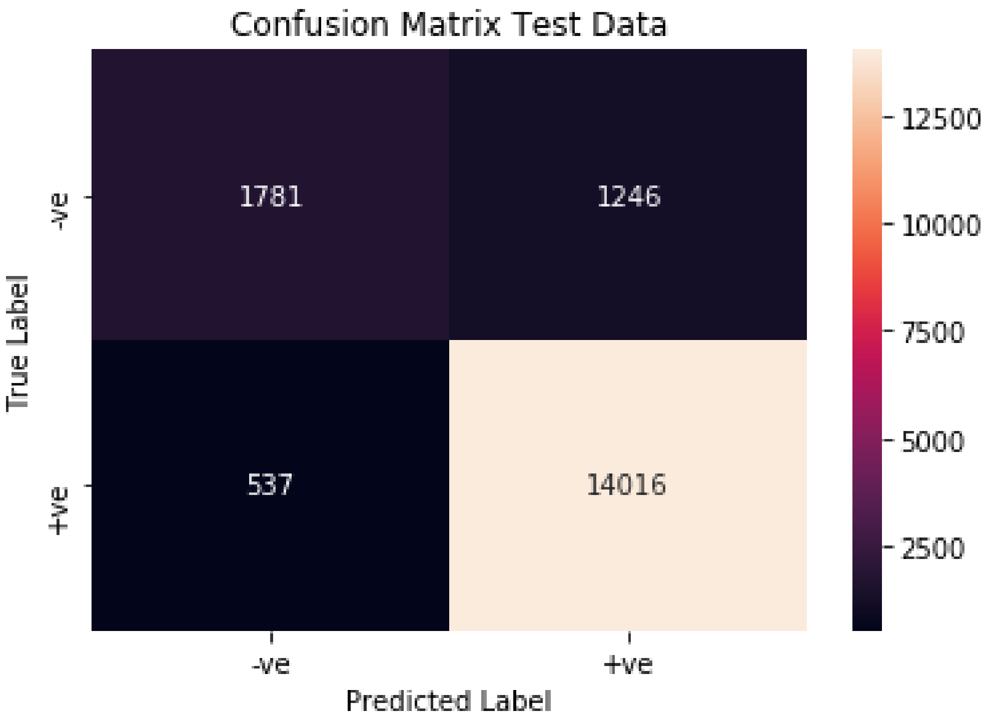
```
In [241]: showConfusionMatrix(train_confusion_matrix, "Confusion Matrix Train Data")
```

```
[[ 6220  2584]
 [ 1646 45802]]
```




```
In [242]: showConfusionMatrix(test_confusion_matrix, "Confusion Matrix Test Data")
```

```
[[ 1781  1246]
 [  537 14016]]
```



Conclusion : total misclassified data = (537+1246) = 1783 out of 17555 data. So, accuracy almost 90%

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

In [243]:
```
feature_log_prob = modelNB.feature_log_prob_
print(feature_log_prob.shape)
#print(feature_log_prob)

featureNames = vectorizer.get_feature_names()

print(featureNames[:10])

df = pd.DataFrame(feature_log_prob, columns=featureNames)
dfT = df.T
#print(dfT.head())
print("Topr 10 features in +ve class: \n",dfT[1].sort_values(ascending=False)[:10])
```

```
(2, 44053)
['aa', 'aaa', 'aaaa', 'aaaaaaaaaaaa', 'aaaaaaaaaaaaaa', 'aaaaah', 'aaaah', 'aaaand', 'aaah',
Topr 10 features in +ve class:
 not        -10.338107
like       -10.523441
good       -10.537423
great      -10.547430
```

```
tea        -10.568377
one        -10.569516
taste      -10.582049
food       -10.587296
love       -10.588114
product    -10.590024
Name: 1, dtype: float64
```

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

```
In [244]: print("Topr 10 features in -ve class: \n",dfT[0].sort_values(ascending=False)[:10])
```

```
Topr 10 features in -ve class:
 not        -10.561181
like       -10.649419
product    -10.658380
would      -10.658795
taste      -10.660817
one        -10.664697
food       -10.669548
good       -10.670115
no         -10.671024
flavor     -10.673237
Name: 0, dtype: float64
```

7.2 [5.2] Applying Naive Bayes on TFIDF, SET 2

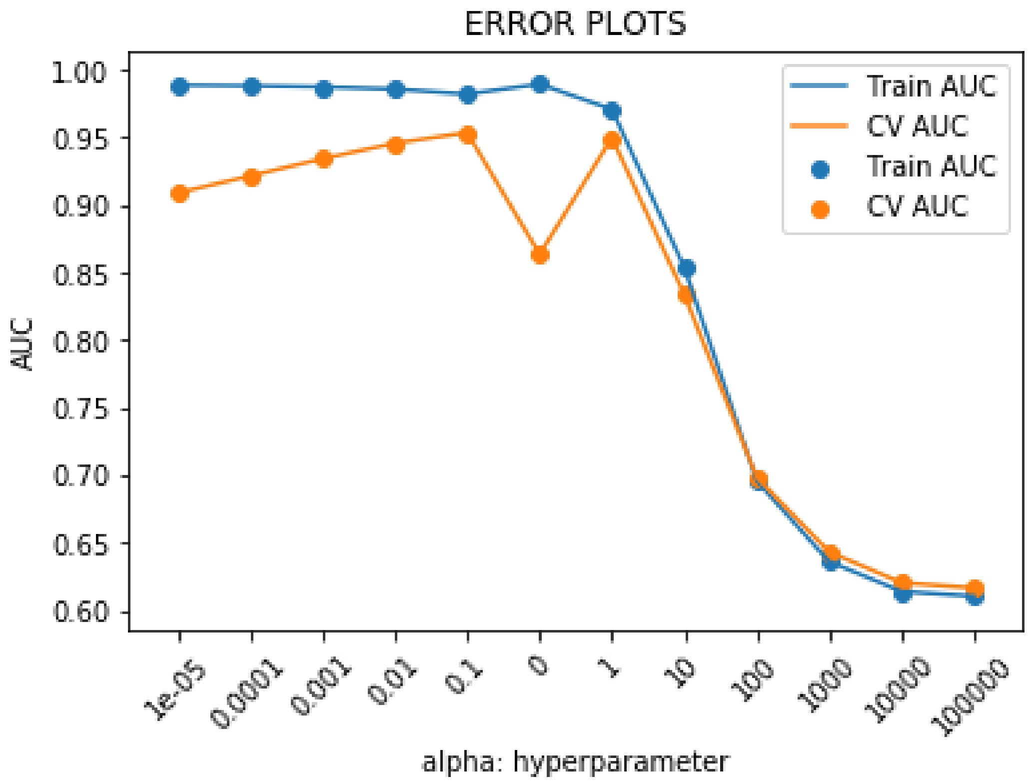
```
In [245]: from sklearn.feature_extraction.text import TfidfVectorizer
          tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
          #Apply tf-idf on splitted data sets
          X_train_tfidf = tf_idf_vect.fit_transform(X_train)
          X_cv_tfidf = tf_idf_vect.transform(X_cv)
          X_test_tfidf = tf_idf_vect.transform(X_test)

          print(type(X_train_tfidf))
          print(type(X_cv_tfidf))
          print(type(X_test_tfidf))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
<class 'scipy.sparse.csr.csr_matrix'>
<class 'scipy.sparse.csr.csr_matrix'>
```

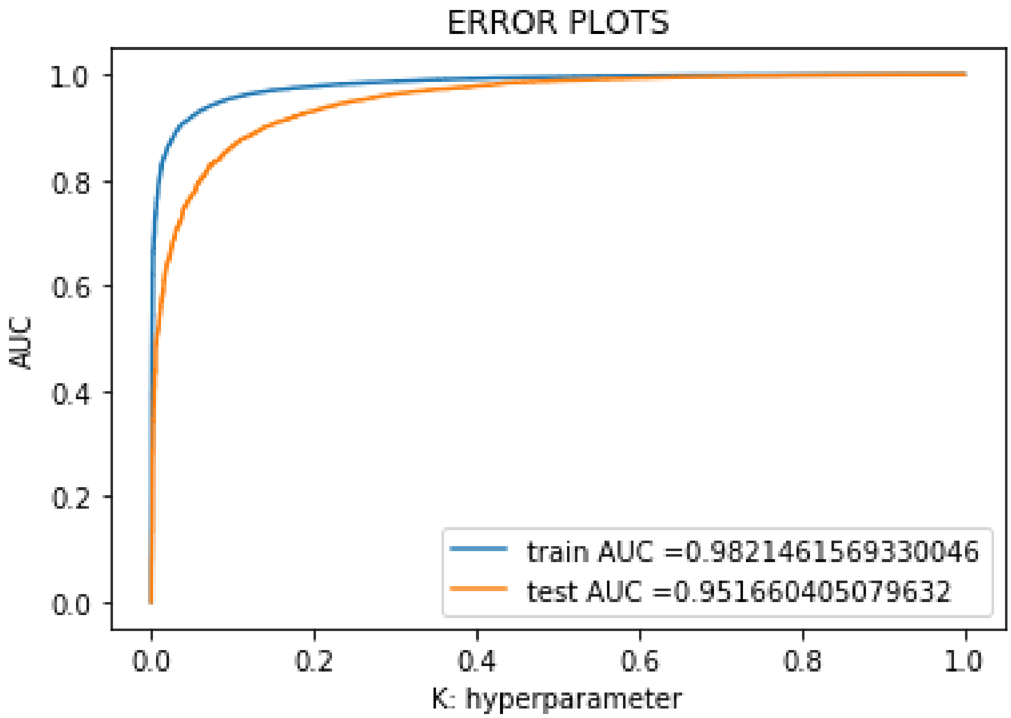
```
In [246]: print(X_train_tfidf.shape)
          print(X_cv_tfidf.shape)
          print(X_test_tfidf.shape)
```

```
(56252, 32863)
(14064, 32863)
(17580, 32863)
```

```
In [247]: modelNB = getOptimalAlpha(X_train_tfidf, y_train, X_cv_tfidf, y_cv)
```

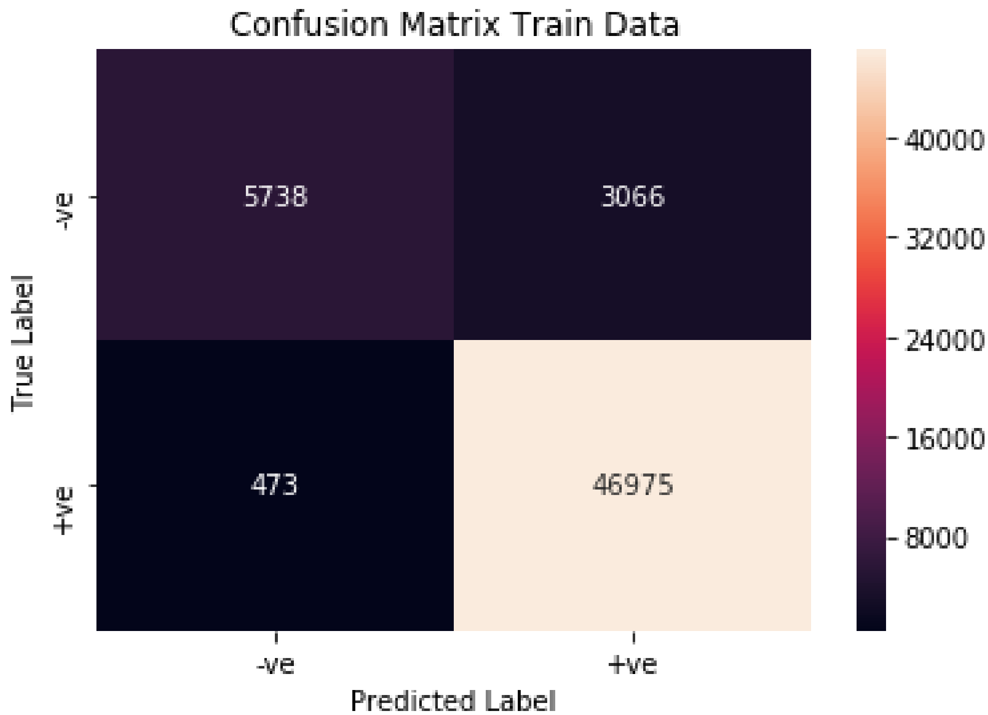



```
In [248]: #Max AUC of for Alpha value 0.1
          train_confusion_matrix, test_confusion_matrix = getNBAnalysis(0.1, X_train_tfidf, y_t
```



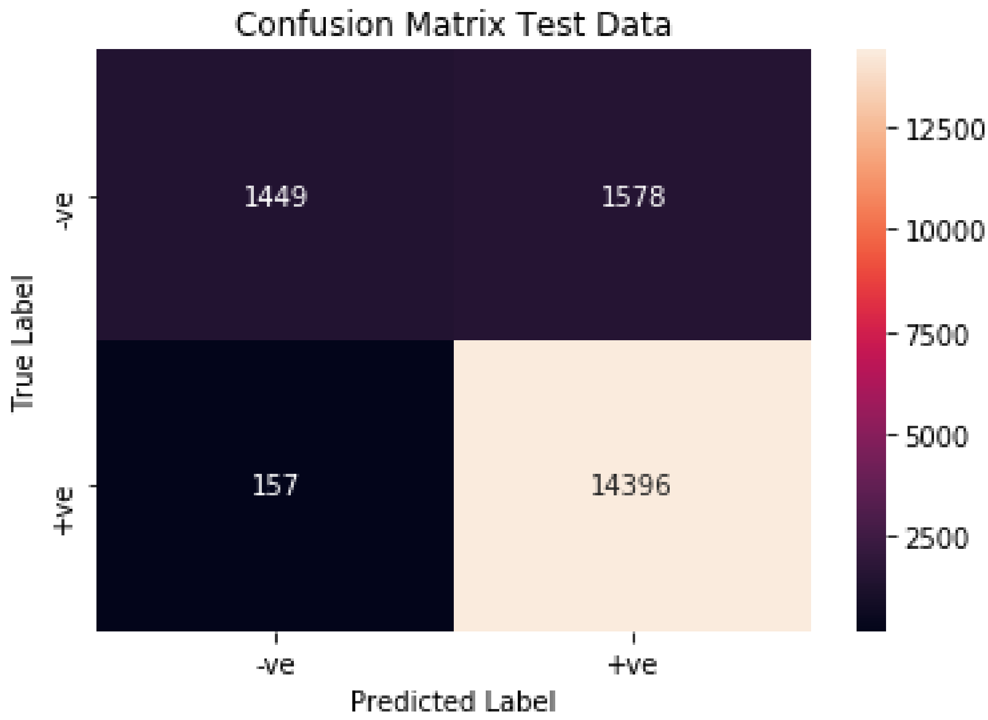
In [249]: showConfusionMatrix(train_confusion_matrix, "Confusion Matrix Train Data")

```
[[ 5738  3066]
 [  473 46975]]
```




```
In [250]: showConfusionMatrix(test_confusion_matrix, "Confusion Matrix Test Data")
```

```
[[ 1449  1578]
 [  157 14396]]
```



Conclusions: Total misclassification = 157 + 1578 = 1735, so accuracy = 90%

7.2.1 [5.2.1] Top 10 important features of positive class from SET 2

In [251]:
```
feature_log_prob = modelNB.feature_log_prob_
print(feature_log_prob.shape)

featureNames = tf_idf_vect.get_feature_names()

print(featureNames[:10])

df = pd.DataFrame(feature_log_prob, columns=featureNames)
dfT = df.T

print("Topr 10 features in +ve class: \n",dfT[1].sort_values(ascending=False)[:10])
```

```
(2, 32863)
['aa', 'aafco', 'abandon', 'abdominal', 'ability', 'able', 'able buy', 'able chew', 'able drin
Topr 10 features in +ve class:
 not       -10.386574
great     -10.390483
good      -10.391042
tea       -10.391244
like      -10.391722
love      -10.392545
```

```
product   -10.393314
one       -10.393526
taste     -10.393653
flavor    -10.393822
Name: 1, dtype: float64
```

7.2.2 [5.2.2] Top 10 important features of negative class from SET 2

```
In [252]: print("Topr 10 features in -ve class: \n",dfT[0].sort_values(ascending=False)[:10])
```

```
Topr 10 features in -ve class:
 not       -10.395442
like      -10.398083
product   -10.398204
would     -10.398320
taste     -10.398334
one       -10.398734
no        -10.398905
food      -10.398920
tea       -10.398982
flavor    -10.398988
Name: 0, dtype: float64
```

7.2.3 [5.3.1] BOW using Review Length as on feature

```
In [254]: stringLen = lambda x : len(x)
          summary_len = list(map(stringLen, preprocessed_summary))

          dfStrLen = pd.DataFrame(summary_len, columns=["len"])
          print(dfStrLen.shape)
          print(type(dfStrLen))
          print(dfStrLen.head())
          textDf = pd.DataFrame({'text':np.asarray(preprocessed_reviews)})
          combinedDf.head()
          #combining text and len of review summary in a data frame
          combinedDf = pd.concat([textDf, dfStrLen], axis=1)
          print(combinedDf.head())
          print(len(preprocessed_reviews))
```

```
(87896, 1)
<class 'pandas.core.frame.DataFrame'>
   len
0   10
1   17
2   18
3   24
4    7
```

```
                                                   text  len
0  dogs loves chicken product china wont buying a...   10
1  dogs love saw pet store tag attached regarding...   17
2  infestation fruitflies literally everywhere fl...   18
3  worst product gotten long time would rate no s...   24
4  wish would read reviews making purchase basica...    7
87896
```

In [255]:
```
X_train, X_test, y_train, y_test = train_test_split(combinedDf, Y, test_size=0.20,shu
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.20,shu
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(56252, 2) (56252,)
(14064, 2) (14064,)
(17580, 2) (17580,)
```

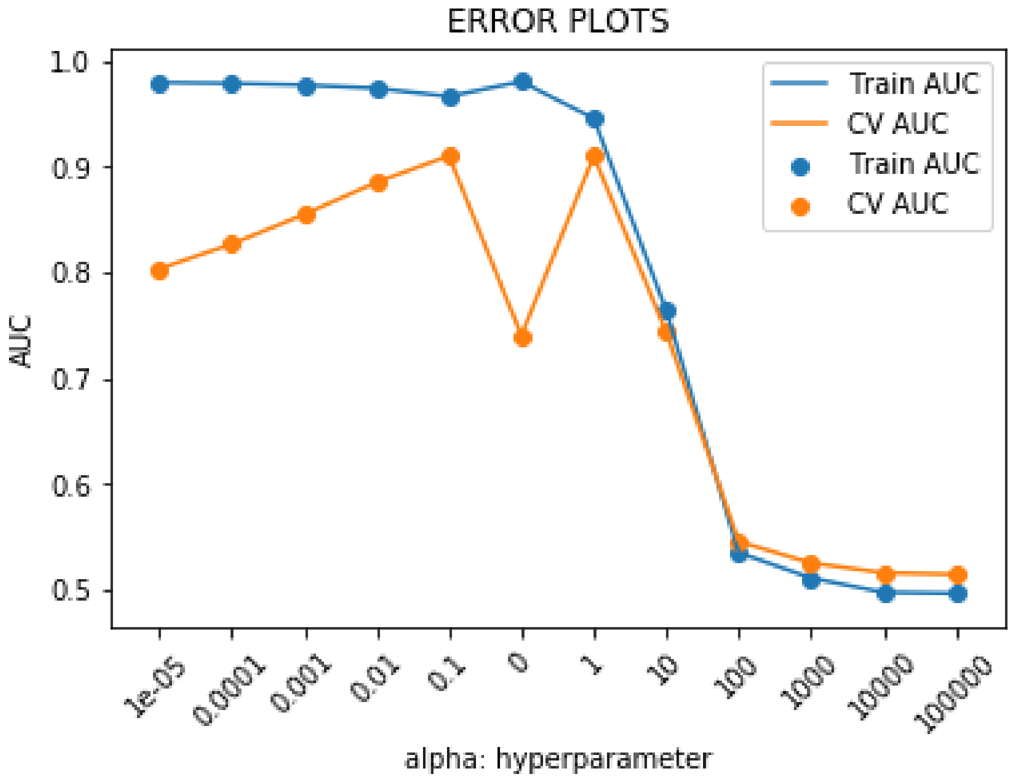
In [256]:
```
#combining bow with review summary len as extra feature
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
#applying the method fit_transform() on you train data, and apply the method transfo
X_train_bow = vectorizer.fit_transform(X_train['text'])
X_cv_bow = vectorizer.transform(X_cv['text'])
X_test_bow = vectorizer.transform(X_test['text'])
```

In [257]:
```
from scipy.sparse import csr_matrix, issparse
from scipy.sparse import coo_matrix, hstack
train = coo_matrix(X_train['len'])
print(train.shape)
print(X_train_bow.shape)
train = train.transpose()
X_train_bow = hstack([X_train_bow,train])

cv = coo_matrix(X_cv['len'])
cv = cv.transpose()
X_cv_bow = hstack([X_cv_bow,cv])

X_test.fillna(X_test.mean())
test = coo_matrix(X_test['len'])
print(test.max())

test = test.transpose()
X_test_bow = hstack([X_test_bow,test])
```

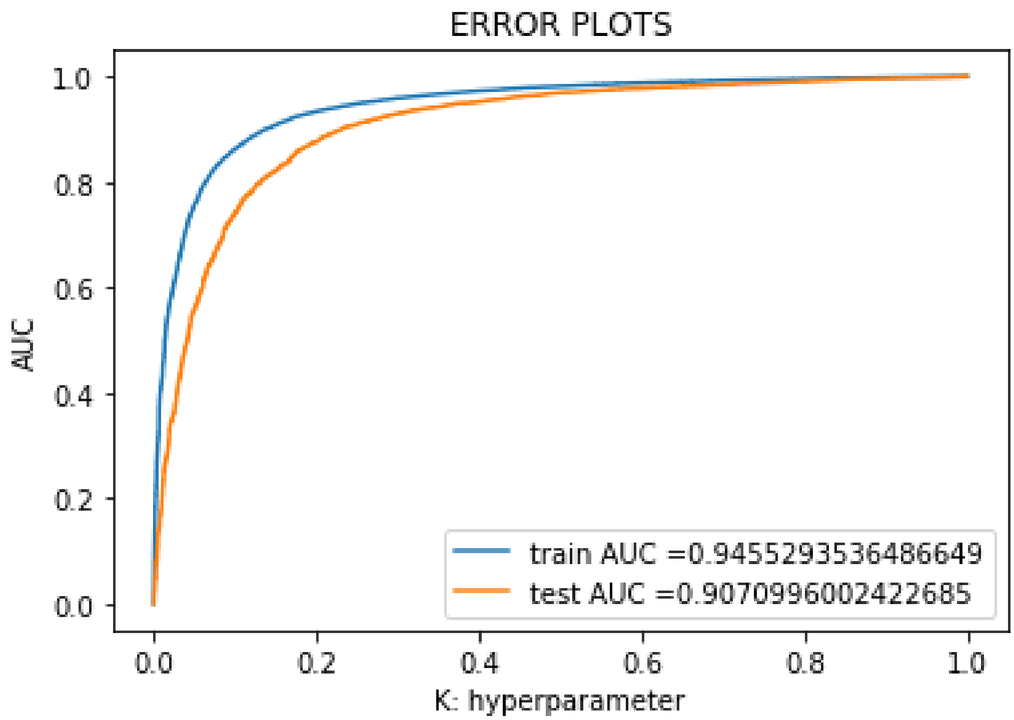
```
(1, 56252)
(56252, 44053)
```

```
114
```

```
In [258]: modelNB = getOptimalAlpha(X_train_bow, y_train, X_cv_bow, y_cv)
```

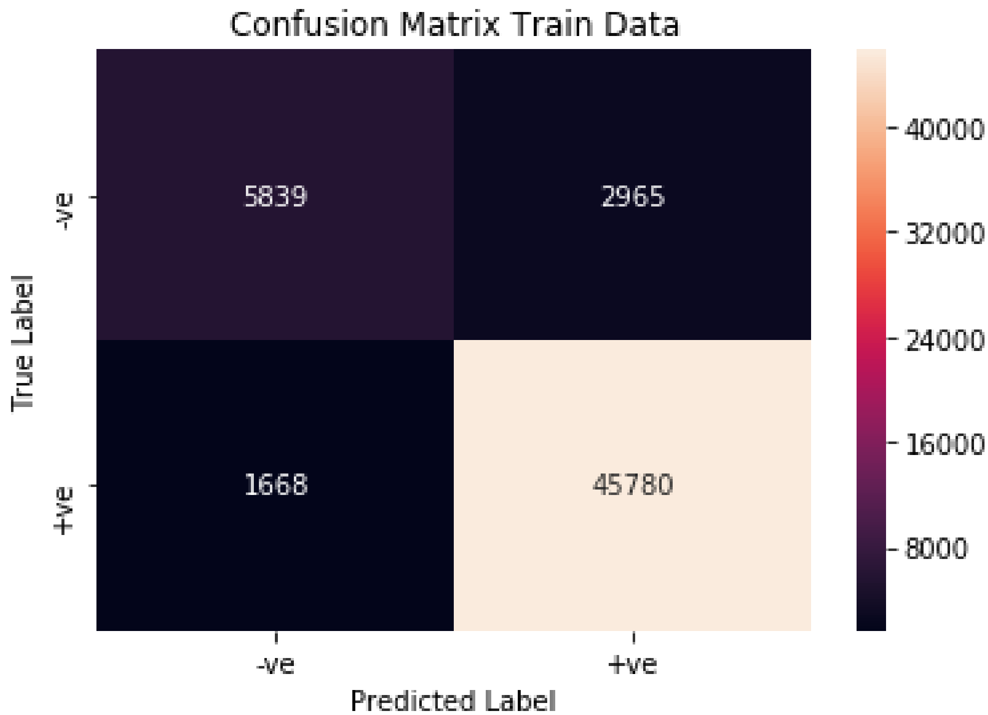



```
In [260]: #Max AUC of for Alpha value 1
          train_confusion_matrix, test_confusion_matrix = getNBAnalysis(1, X_train_bow, y_trai
```



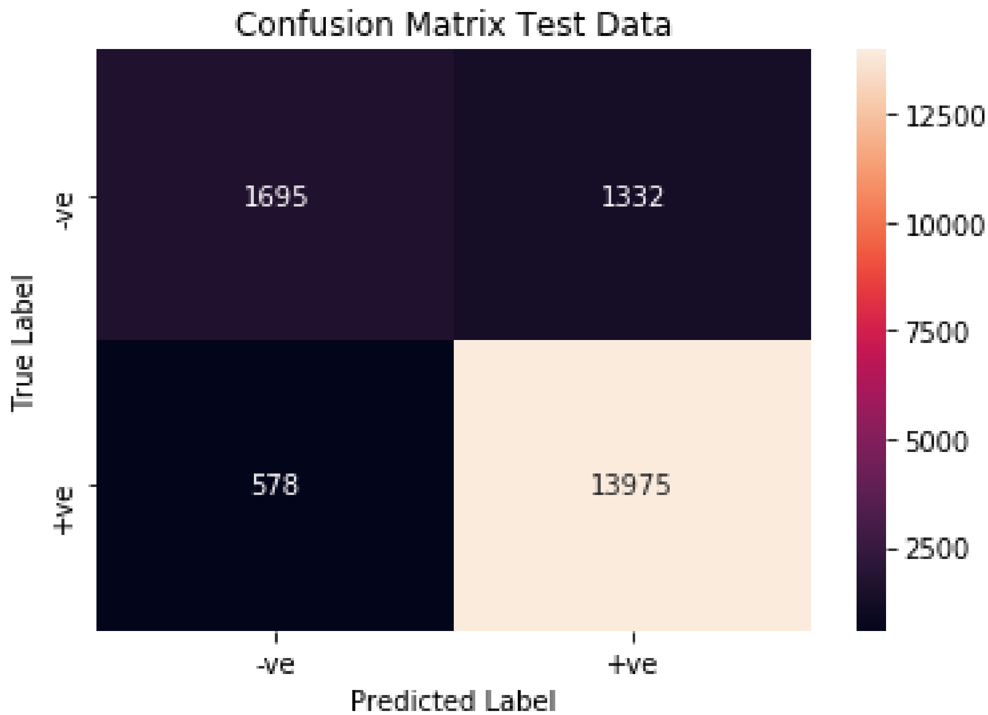

```
In [261]: showConfusionMatrix(train_confusion_matrix, "Confusion Matrix Train Data")
```

```
[[ 5839  2965]
 [ 1668 45780]]
```




```
In [262]: showConfusionMatrix(test_confusion_matrix, "Confusion Matrix Test Data")
```

```
[[ 1695  1332]
 [  578 13975]]
```



Conclusion : Misclassified points = 1910, accuracy = 89%

7.2.4 [5.3.2] Tf-Idf using Review Length as on feature

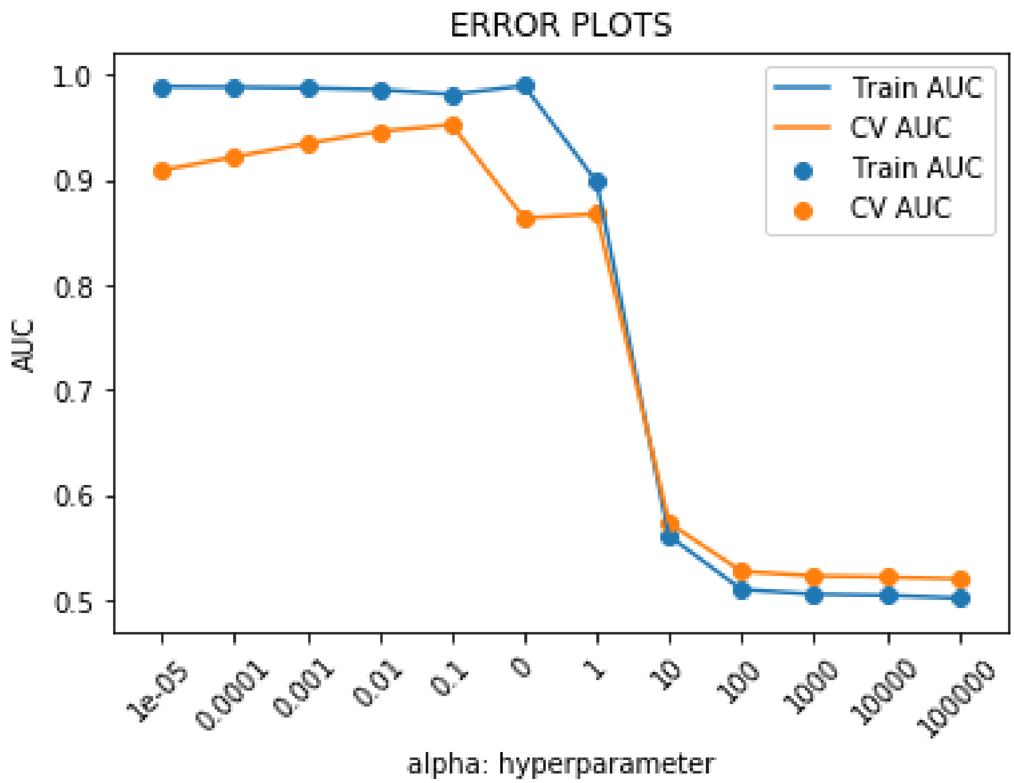
In [264]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)

X_train_bow = tf_idf_vect.fit_transform(X_train['text'])
X_cv_bow = tf_idf_vect.transform(X_cv['text'])
X_test_bow = tf_idf_vect.transform(X_test['text'])
```

In [265]:
```
from scipy.sparse import csr_matrix, issparse
from scipy.sparse import coo_matrix, hstack
train = coo_matrix(X_train['len'])
print(train.shape)
print(X_train_bow.shape)
train = train.transpose()
X_train_bow = hstack([X_train_bow,train])

cv = coo_matrix(X_cv['len'])
cv = cv.transpose()
X_cv_bow = hstack([X_cv_bow,cv])

X_test.fillna(X_test.mean())
test = coo_matrix(X_test['len'])
```

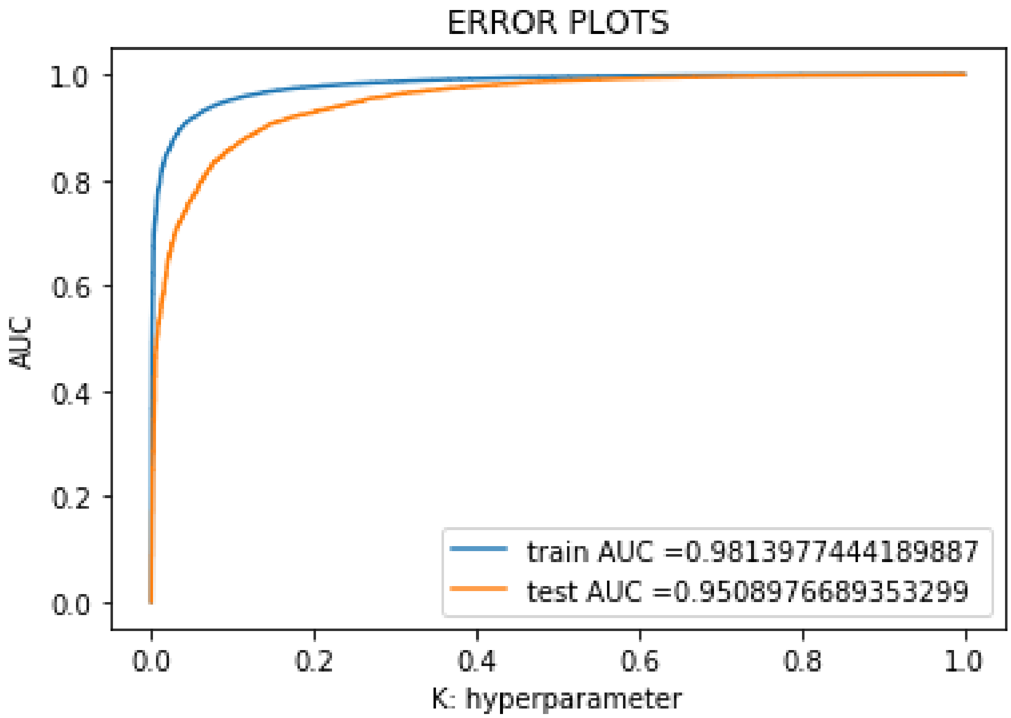
```
        print(test.max())

        test = test.transpose()
        X_test_bow = hstack([X_test_bow,test])
```

```
(1, 56252)
(56252, 32863)
114
```

In [266]:
```
modelNB = getOptimalAlpha(X_train_bow, y_train, X_cv_bow, y_cv)
```

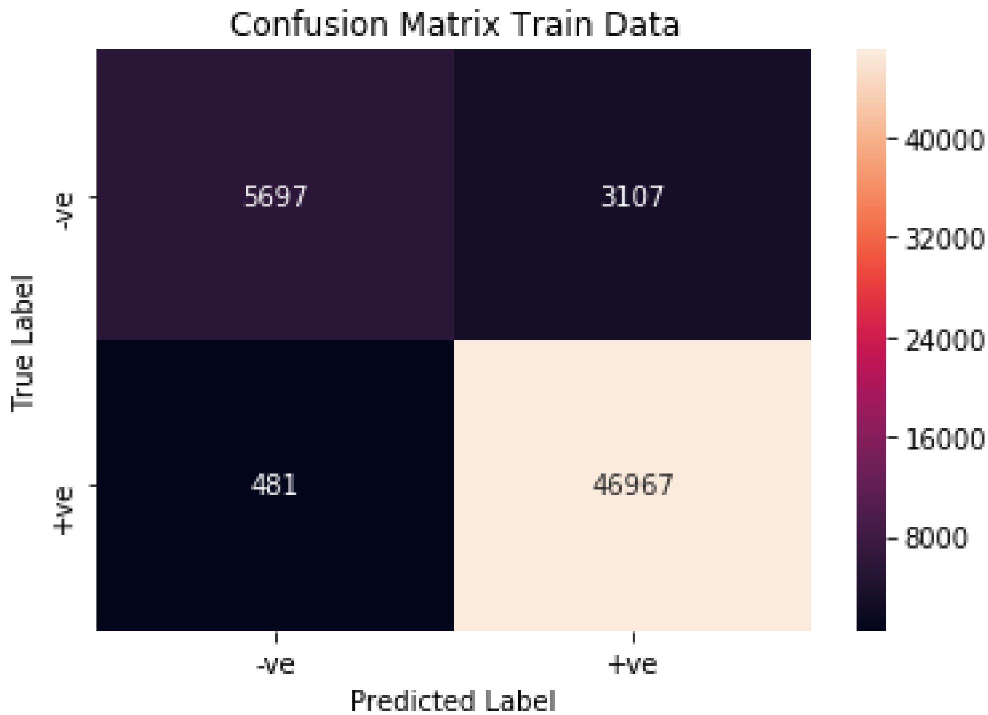


In [267]:
```
#Max AUC of for Alpha value 0.1
train_confusion_matrix, test_confusion_matrix = getNBAnalysis(0.1, X_train_bow, y_tra
```



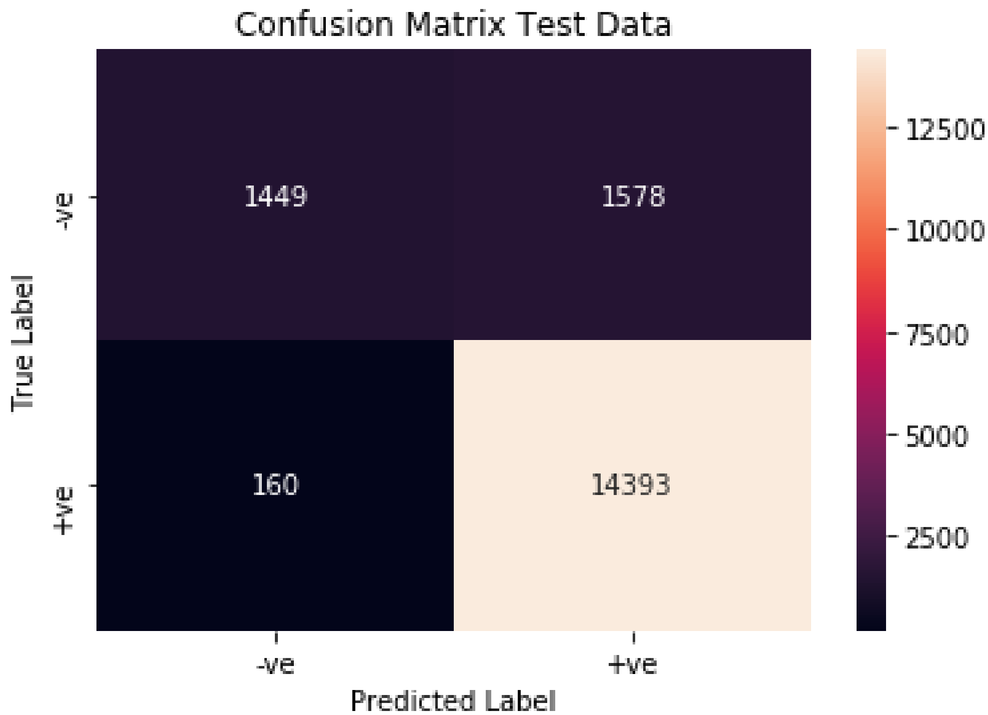

```
In [268]: showConfusionMatrix(train_confusion_matrix, "Confusion Matrix Train Data")
```

```
[[ 5697  3107]
 [  481 46967]]
```




```
In [269]: showConfusionMatrix(test_confusion_matrix, "Confusion Matrix Test Data")
```

```
[[ 1449  1578]
 [  160 14393]]
```



Conclusion: Total misclassified points are = 1578+160 = 1738, so Accuracy is 90%

8 [6] Conclusions

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

        x = PrettyTable()
        x.field_names = ["Vectorizer", "Hyper parameter(Alpha)", "AUC", "Accuracy"]

        x.add_row(["BOW", "1", 0.9014, "90%"])
        x.add_row(["TFIDF", "0.1", 0.95, "90%"])
        x.add_row(["BOW with extra feature", "1", 0.907, "89%"])
        x.add_row(["TFIDF with extra feature", "0.1", 0.951, "90%"])

        print(x)
```

```
+--------------------------+------------------------+--------+----------+
|        Vectorizer        | Hyper parameter(Alpha) |  AUC   | Accuracy |
+--------------------------+------------------------+--------+----------+
|           BOW            |           1            | 0.9014 |   90%    |
|          TFIDF           |          0.1           |  0.95  |   90%    |
|  BOW with extra feature  |           1            | 0.907  |   89%    |
| TFIDF with extra feature |          0.1           | 0.951  |   90%    |
```

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

Though the result of each of the Model is comparable i.e. not having much differences, but TF-IDF with review summary as extra feature is little ahead of the rest.

```
In [ ]:
```