

Amazon Fine Food Reviews Analysis

Data Source: <https://www.kaggle.com/snap/amazon-fine-food-reviews>
(<https://www.kaggle.com/snap/amazon-fine-food-reviews>)

EDA: <https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/>
(<https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/>)

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

Number of reviews: 568,454

Number of users: 256,059

Number of products: 74,258

Timespan: Oct 1999 - Oct 2012

Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unique identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

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

[Q] How to determine if a review is positive or negative?

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

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

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

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

In [1]:

```
%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 [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect('../Logistic regression/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 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
0000""", con)
# 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 negative
rating(0).
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[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulDenominator
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1



In [3]:

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

In [4]:

```
print(display.shape)
display.head()
```

(80668, 7)

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	C
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:

```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recomme to try gree tea extrac ...

In [6]:

```
display['COUNT(*)'].sum()
```

Out[6]:

393063

[2] Exploratory Data Analysis

[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 [7]:

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2



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 delete 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 [8]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False,
kind='quicksort', na_position='last')
```

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
'first', inplace=False)
final.shape
```

Out[9]:

(364173, 10)

In [10]:

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

Out[10]:

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 calculations

In [11]:

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [12]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:

```
#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[13]:

```
1    307061
0     57110
Name: Score, dtype: int64
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

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

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

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

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

In [14]:

```
# 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 along and he always can sing the refrain. he's learned about whales, India, drooping roses: i love all the new words this book introduces and the silliness of it all. this is a classic book i am willing to bet my son will STILL be able to recite from memory when he is in college

=====

I was really looking forward to these pods based on the reviews. Starbucks is good, but I prefer bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back in 2005 for gosh sakes. I admit that Amazon agreed to credit me for cost plus part of shipping, but geez, 2 years expired!!! I'm hoping to find local San Diego area shoppe that carries pods so that I can try something different than starbucks.

=====

Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belongs in it is Canola oil. Canola or rapeseed is not something a dog would ever find in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food industries have convinced the masses that Canola oil is a safe and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured out a way to fix that. I still like it but it could be better.

=====

Can't do sugar. Have tried scores of SF Syrups. NONE of them can touch the excellence of this product.

Thick, delicious. Perfect. 3 ingredients: Water, Maltitol, Natural Maple Flavor. PERIOD. No chemicals. No garbage.

Have numerous friends & family members hooked on this stuff. My husband & son, who do NOT like "sugar free" prefer this over major label regular syrup.

I use this as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies, etc... Unbelievably delicious...

Can you tell I like it? :)

=====

In [15]:

```
# 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 along and he always can sing the refrain. he's learned about whales, India, drooping roses: i love all the new words this book introduces and the silliness of it all. this is a classic book i am willing to bet my son will STILL be able to recite from memory when he is in college

In [16]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-t
ags-from-an-element
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 along and he always can sing the refrain. he's learned about whales, India, drooping roses: i love all the new words this book introduces and the silliness of it all. this is a classic book i am willing to bet my son will STILL be able to recite from memory when he is in college

=====

I was really looking forward to these pods based on the reviews. Starbucks is good, but I prefer bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back in 2005 for gosh sakes. I admit that Amazon agreed to credit me for cost plus part of shipping, but geez, 2 years expired!!! I'm hoping to find local San Diego area shoppe that carries pods so that I can try something different than starbucks.

=====

Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belongs in it is Canola oil. Canola or rapeseed is not something a dog would ever find in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food industries have convinced the masses that Canola oil is a safe and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured out a way to fix that. I still like it but it could be better.

=====

Can't do sugar. Have tried scores of SF Syrups. NONE of them can touch the excellence of this product. Thick, delicious. Perfect. 3 ingredients: Water, Maltitol, Natural Maple Flavor. PERIOD. No chemicals. No garbage. Have numerous friends & family members hooked on this stuff. My husband & son, who do NOT like "sugar free" prefer this over major label regular syrup. I use this as my SWEETENER in baking: cheesecakes, white brownies, muffins, pumpkin pies, etc... Unbelievably delicious...Can you tell I like it? :)

In [17]:

```
# 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 [18]:

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

Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing I do not think belongs in it is Canola oil. Canola or rapeseed is not something a dog would ever find in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's food industries have convinced the masses that Canola oil is a safe and even better oil than olive or virgin coconut, facts though say otherwise. Until the late 70s it was poisonous until they figured out a way to fix that. I still like it but it could be better.

=====

In [19]:

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub(r"\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 along and he always can sing the refrain. he's learned about whales, India, drooping roses: i love all the new words this book introduces and the silliness of it all. this is a classic book i am willing to bet my son will STILL be able to recite from memory when he is in college

In [20]:

```
#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 thing I do not think belongs in it is Canola oil Canola or rapeseed is not something a dog would ever find in nature and if it did find rapeseed in nature and eat it it would poison them Today is Food industries have convinced the masses that Canola oil is a safe and even better oil than olive or virgin coconut facts though say otherwise Until the late 70s it was poisonous until they figured out a way to fix that I still like it but it could be better

In [21]:

```
# 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 been removed in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", \
    "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', \
    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', \
    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', \
    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', \
    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', \
    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', \
    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
    'won', "won't", 'wouldn', "wouldn't"])
```

In [22]:

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

100%|██| 364171/364171 [10:44<00:00, 565.09 it/s]

In [23]:

```
preprocessed_reviews[1500]
```

Out[23]:

```
'great ingredients although chicken rather chicken broth thing not think b
elongs canola oil canola rapeseed not someting dog would ever find nature
find rapeseed nature eat would poison today food industries convinced mass
es canola oil safe even better oil olive virgin coconut facts though say o
therwise late poisonous figured way fix still like could better'
```

[3.2] Preprocessing Review Summary

In [24]:

```
## Similarly you can do preprocessing for review summary also.
```

[4] Featurization

[5] Assignment 4: Apply Naive Bayes

1. 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)

2. The hyper parameter tuning (find best Alpha)

- Find the best hyper parameter which will give the maximum AUC (<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/>). value
- Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
- Find the best hyper parameter 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

3. Feature importance


- Find the top 10 features of positive class and top 10 features of negative class for both feature sets Set 1 and Set 2 using values of `feature_log_prob_` parameter of MultinomialNB (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print their corresponding feature names


4. Feature engineering


- To increase the performance of your model, you can also experiment with feature engineering like :
 - Taking length of reviews as another feature.
 - Considering some features from review summary as well.

5. Representation of results

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure. Here on X-axis you will have alpha values, since they have a wide range, just to represent those alpha values on the graph, apply log function on those alpha values.

 Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.

 Along with plotting ROC curve, you need to print the confusion matrix (<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tp-r-fpr-fnr-tnr-1/>) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

 (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>)
(<https://seaborn.pydata.org/generated/seaborn.heatmap.html>)
(<https://seaborn.pydata.org/generated/seaborn.heatmap.html>)

(<https://seaborn.pydata.org/generated/seaborn.heatmap.html>)

6. Conclusion (<https://seaborn.pydata.org/generated/seaborn.heatmap.html>)

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library.

(<https://seaborn.pydata.org/generated/seaborn.heatmap.html>) link
(<http://zetcode.com/python/prettitable/>)



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-leakage, 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](https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf). (<https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf>)

Applying Multinomial Naive Bayes

[5.1] Applying Naive Bayes on BOW, SET 1

In [25]:

```
# Please write all the code with proper documentation

# ===== Loading Libraries =====

import warnings
warnings.filterwarnings("ignore")
from sklearn.cross_validation import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn import cross_validation
```

In [26]:

```
df = pd.DataFrame({'Text':preprocessed_reviews})
X = df['Text'].values
y = final['Score'].values
```

In [27]:

```
# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this is random splitting
```

In [28]:

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(163475,) (163475,)
(80519,) (80519,)
(120177,) (120177,)
```

In [29]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
```

```
After vectorizations
(163475, 77484) (163475,)
(80519, 77484) (80519,)
(120177, 77484) (120177,)
```

Simple cross validation

In [30]:

```
from sklearn.naive_bayes import MultinomialNB
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
#for i in range(1,50,2):
for i in K:
    clf = MultinomialNB(alpha= i, class_prior=None, fit_prior=True)
    clf.fit(X_train_bow, y_train)

    # predict the response on the crossvalidation train
    pred = clf.predict(X_cv_bow)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for alpha = %f is %d%%' % (i, acc))
```

CV accuracy for alpha = 0.000100 is 89%

CV accuracy for alpha = 0.001000 is 90%

CV accuracy for alpha = 0.010000 is 90%

CV accuracy for alpha = 0.100000 is 90%

CV accuracy for alpha = 1.000000 is 91%

CV accuracy for alpha = 10.000000 is 85%

CV accuracy for alpha = 100.000000 is 84%

CV accuracy for alpha = 1000.000000 is 84%

CV accuracy for alpha = 10000.000000 is 84%

Simple for loop

In [31]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
#K = [1, 5, 10, 15, 21, 31, 41, 51]
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
for i in K:
    neigh = MultinomialNB(alpha= i, class_prior=None, fit_prior=True)
    neigh.fit(X_train_bow, y_train)

    #y_train_pred = []
    #for i in range(0, X_train.shape[0], 1000):
    #    y_train_pred.extend(neigh.predict_proba(X_train_bow[i:i+1000]))[:,1])

    #y_cv_pred = []
    #for i in range(0, X_cv.shape[0], 1000):
    #    y_cv_pred.extend(neigh.predict_proba(X_cv_bow[i:i+1000]))[:,1])

    y_train_pred = neigh.predict_proba(X_train_bow)[:,1]
    y_cv_pred = neigh.predict_proba(X_cv_bow)[:,1]

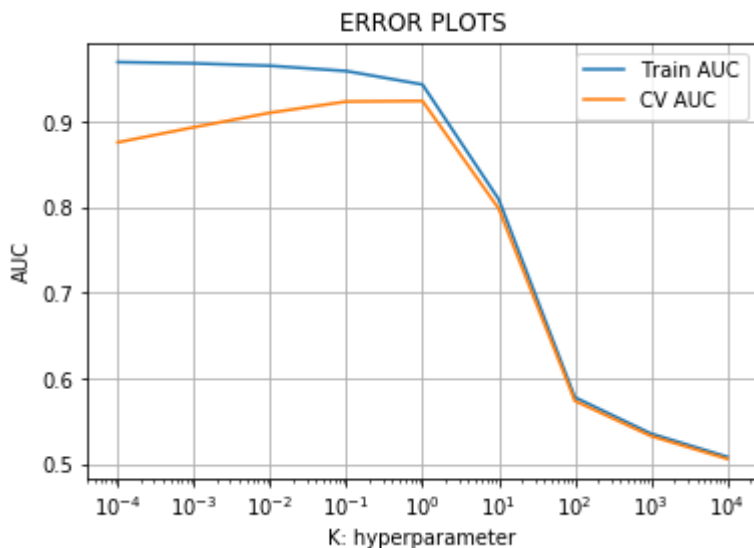
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

plt.semilogx(K, train_auc, label='Train AUC')
plt.semilogx(K, cv_auc, label='CV AUC')
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[31]:

Text(0.5,1,'ERROR PLOTS')



OBSERVATION: The best value of alpha is 1.

Testing with Test data

In [32]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

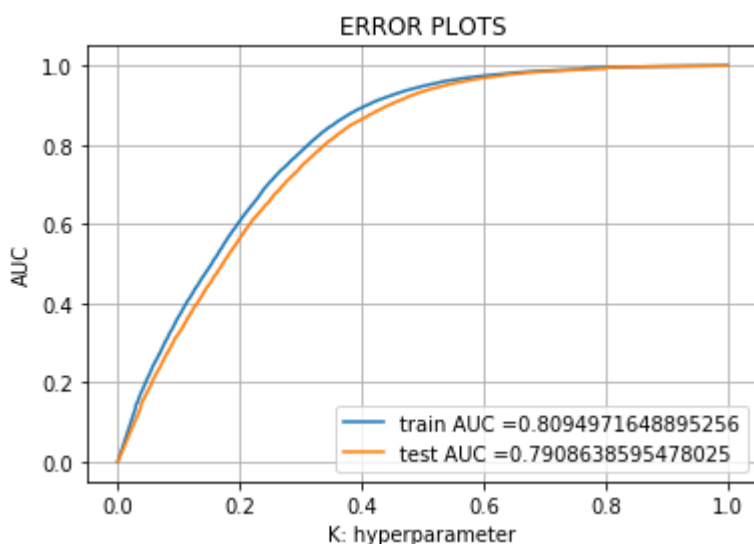
neigh = MultinomialNB(alpha= 10**1, class_prior=None, fit_prior=True)
neigh.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_bow)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_bow)[:,1])

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.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_bow)))
```



```
=====
=====
```

Train confusion matrix

```
[[ 3794 21700]
 [  383 137598]]
```

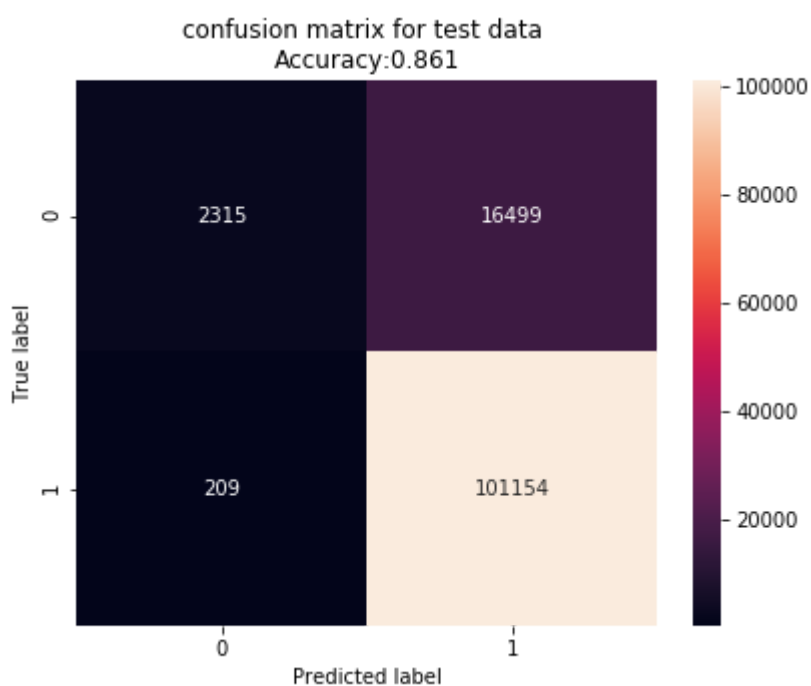
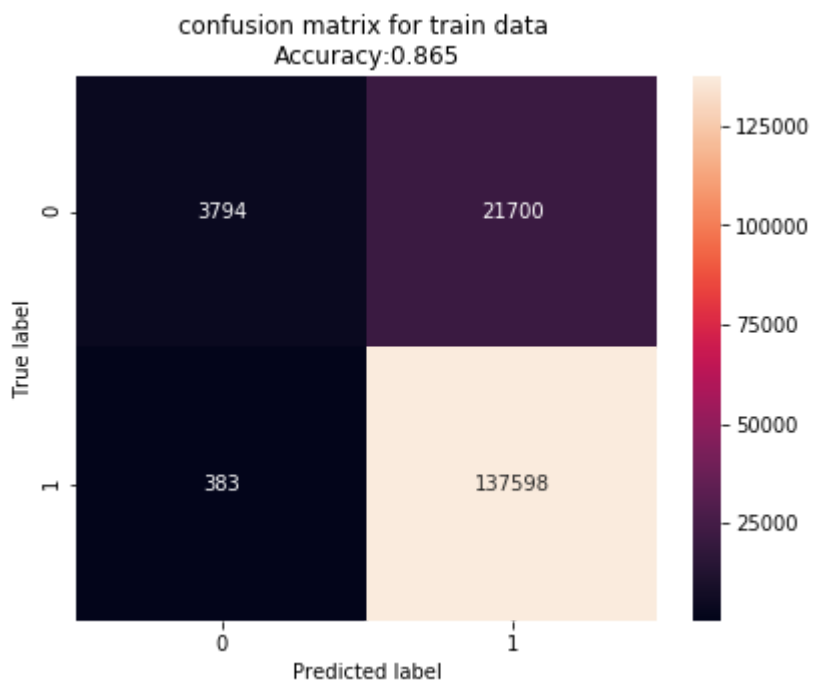
Test confusion matrix

```
[[ 2315 16499]
 [  209 101154]]
```


In [33]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_bow))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_train, neigh.predict(X_train_bow))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_bow))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_test, neigh.predict(X_test_bow))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



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

In [44]:

```
#positive

a = neigh.coef_[0]
a_std = np.argsort(a)
#np.take(df['Text'].values, a_std)
print(np.take(vectorizer.get_feature_names(), a_std)[::-1][:10])

['not' 'like' 'good' 'great' 'one' 'taste' 'tea' 'product' 'coffee'
 'flavor']
```

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

In [56]:

```
#negative

b = neigh.coef_[0]
b_std = np.argsort(b)
#np.take(df['Text'].values, a_std)
print(np.take(vectorizer.get_feature_names(), b_std)[:10])

['babcia' 'verifiy' 'crazzeto' 'ourves' 'returnablestrike' 'fukien'
 'inroads' 'oursecondary' 'returningit' 'rabbinical']
```

[5.2] Applying Naive Bayes on TFIDF, SET 2

In [123]:

```
X = df['Text'].values
y = final['Score'].values
```

In [124]:

```
# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this is random splitting
```

In [125]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tf = tf_idf_vect.transform(X_train)
X_cv_tf = tf_idf_vect.transform(X_cv)
X_test_tf = tf_idf_vect.transform(X_test)

print("After vectorizations")
print(X_train_tf.shape, y_train.shape)
print(X_cv_tf.shape, y_cv.shape)
print(X_test_tf.shape, y_test.shape)
```

```
After vectorizations
(163475, 93988) (163475,)
(80519, 93988) (80519,)
(120177, 93988) (120177,)
```

Simple Cross Validation:

In [126]:

```
from sklearn.naive_bayes import MultinomialNB
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
#for i in range(1,50,2):
for i in K:
    clf = MultinomialNB(alpha= i, class_prior=None, fit_prior=True)
    clf.fit(X_train_tf, y_train)

    # predict the response on the crossvalidation train
    pred = clf.predict(X_cv_tf)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for alpha = %f is %d%%' % (i, acc))
```

CV accuracy for alpha = 0.000100 is 91%

CV accuracy for alpha = 0.001000 is 91%

CV accuracy for alpha = 0.010000 is 91%

CV accuracy for alpha = 0.100000 is 91%

CV accuracy for alpha = 1.000000 is 89%

CV accuracy for alpha = 10.000000 is 84%

CV accuracy for alpha = 100.000000 is 84%

CV accuracy for alpha = 1000.000000 is 84%

CV accuracy for alpha = 10000.000000 is 84%

In [127]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
#K = [1, 5, 10, 15, 21, 31, 41, 51]
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
for i in K:
    neigh = MultinomialNB(alpha= i, class_prior=None, fit_prior=True)
    neigh.fit(X_train_tf, y_train)
    y_train_pred = neigh.predict_proba(X_train_tf)[: ,1]
    y_cv_pred = neigh.predict_proba(X_cv_tf)[: ,1]

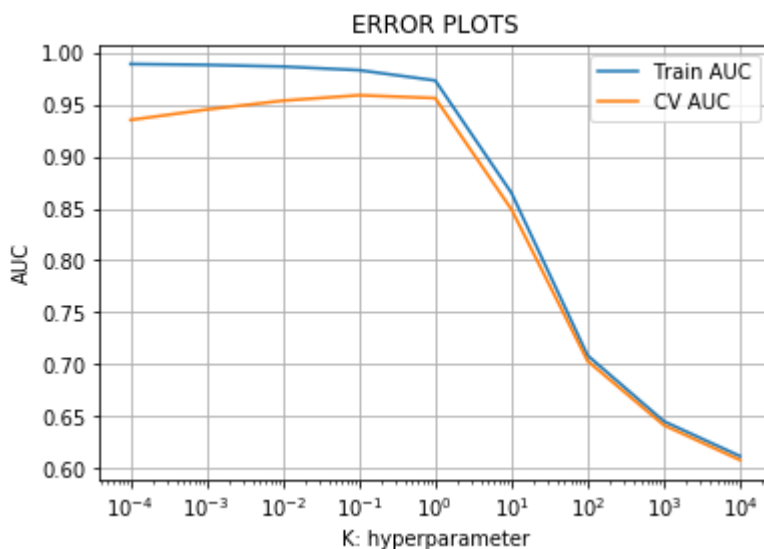
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

plt.semilogx(K, train_auc, label='Train AUC')
plt.semilogx(K, cv_auc, label='CV AUC')
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[127]:

Text(0.5,1,'ERROR PLOTS')



Observation: Best alpha is 10

Testing:

In [128]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

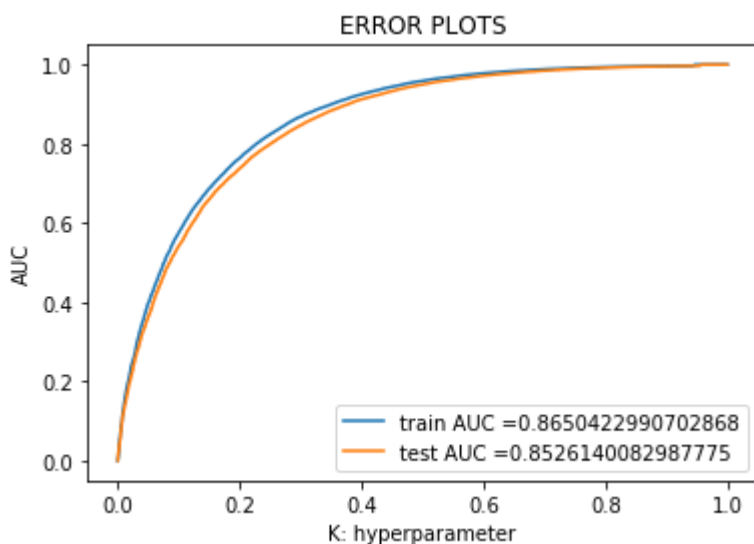
neigh = MultinomialNB(alpha= 10, class_prior=None, fit_prior=True)
neigh.fit(X_train_tf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(X_train_tf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(X_test_tf)[:,1])

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()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(X_train_tf)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(X_test_tf)))
```



```
=====
=====
Train confusion matrix
```

```
[[ 10 25800]
 [  0 137665]]
```

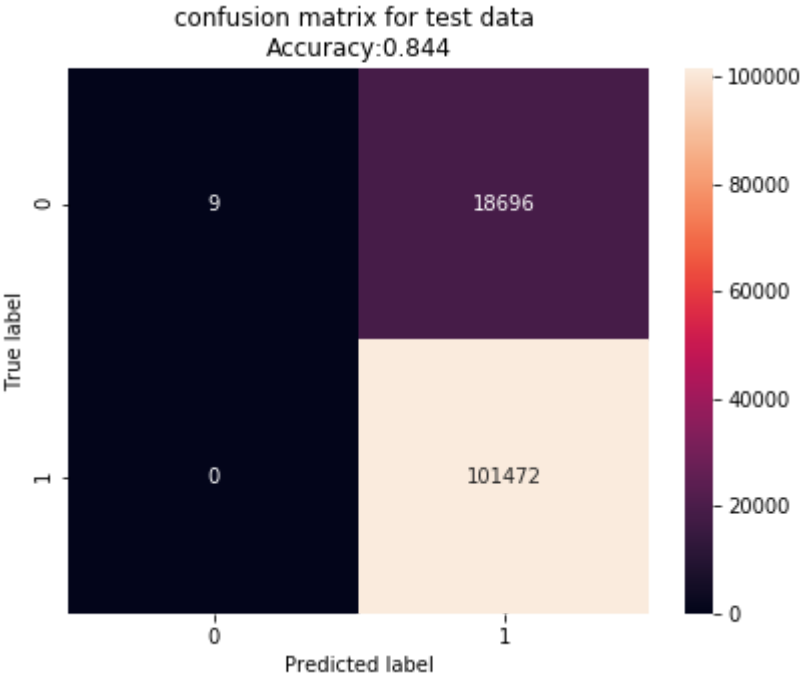
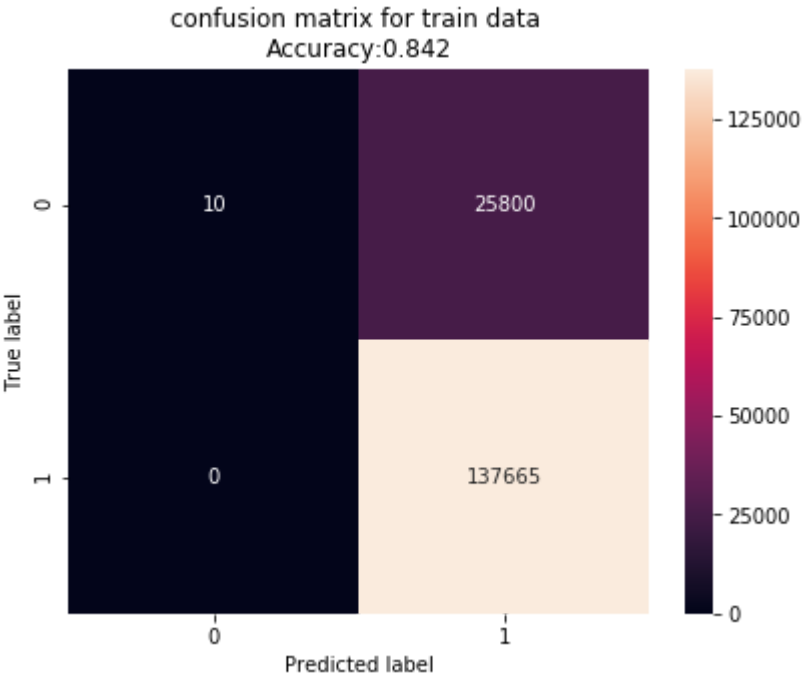
```
Test confusion matrix
```

```
[[  9 18696]
 [  0 101472]]
```

In [129]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(X_train_tf))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_
train, neigh.predict(X_train_tf))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(X_test_tf))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_t
est, neigh.predict(X_test_tf))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



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

In [130]:

```
#positive

a = neigh.feature_log_prob_[1]
a_std = np.argsort(a)
#np.take(df['Text'].values, a_std)
print(np.take(tf_idf_vect.get_feature_names(), a_std)[:10])

['threw remaining' 'completely regret' 'misleading product' 'back refund'
'leftover actual' 'worst ever' 'stuff terrible' 'ended trash'
'bad purchase' 'product poorly']
```

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

In [132]:

```
# negative

b = neigh.feature_log_prob_[0]
b_std = np.argsort(b)
#np.take(df['Text'].values, a_std)
print(np.take(tf_idf_vect.get_feature_names(), b_std)[:10])

['hiking biking' 'impressed tea' 'pampered' 'pamper' 'sprinkle yogurt'
'tropical blend' 'water processed' 'pamela ultimate' 'tropical flavors'
'great single']
```

Feature Engineering:

In [79]:

```
# Calculating the number of words in each line.

df["length1"] = df['Text'].apply(len)
```

In [81]:

```
X = df["length1"].values
y = final['Score'].values
```

In [82]:

```
# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this is random splitting
```

In [83]:

```
#####

# Standarzing the data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train.reshape(-1, 1))

# we use the fit the data to standard scaler
X_train_ss = scaler.transform(X_train.reshape(-1, 1))
X_cv_ss = scaler.transform(X_cv.reshape(-1, 1))
X_test_ss = scaler.transform(X_test.reshape(-1, 1))

print("After Standardization")
print(X_train_ss.shape, y_train.shape)
print(X_cv_ss.shape, y_cv.shape)
print(X_test_ss.shape, y_test.shape)

#####
```

```
After Standardization
(163475, 1) (163475,)
(80519, 1) (80519,)
(120177, 1) (120177,)
```

In [84]:

```
type(X_train_ss)
```

Out[84]:

```
numpy.ndarray
```

In [86]:

```
from scipy import sparse
s_train = sparse.csr_matrix(X_train_ss)
s_cv = sparse.csr_matrix(X_cv_ss)
s_test = sparse.csr_matrix(X_test_ss)
```

Performing BOW

In [87]:

```
X = final['Text'].values
y = final['Score'].values
```

In [88]:

```
# split the data set into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33) # this is ran
dom splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # thi
s is random splitting
```

In [89]:

```
# Applying BOW

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
```

```
After vectorizations
(163475, 77844) (163475,)
(80519, 77844) (80519,)
(120177, 77844) (120177,)
```

In [91]:

```
s_train.shape
```

Out[91]:

```
(163475, 1)
```

In [92]:

```
# Combining the BOW and sentence Length sparse matrix

from scipy.sparse import coo_matrix, hstack
final_train = hstack([X_train_bow, s_train])
final_cv = hstack([X_cv_bow, s_cv])
final_test = hstack([X_test_bow, s_test])
```

In [93]:

```
from sklearn.naive_bayes import MultinomialNB
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
#for i in range(1,50,2):
for i in K:
    clf = MultinomialNB(alpha= i, class_prior=None, fit_prior=True)
    clf.fit(final_train, y_train)

    # predict the response on the crossvalidation train
    pred = clf.predict(final_cv)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for alpha = %f is %d%%' % (i, acc))
```

CV accuracy for alpha = 0.000100 is 89%

CV accuracy for alpha = 0.001000 is 90%

CV accuracy for alpha = 0.010000 is 90%

CV accuracy for alpha = 0.100000 is 90%

CV accuracy for alpha = 1.000000 is 91%

CV accuracy for alpha = 10.000000 is 86%

CV accuracy for alpha = 100.000000 is 84%

CV accuracy for alpha = 1000.000000 is 84%

CV accuracy for alpha = 10000.000000 is 84%

In [96]:

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

train_auc = []
cv_auc = []
#K = [1, 5, 10, 15, 21, 31, 41, 51]
K = [10**-4, 10**-3, 10**-2, 10**-1, 1, 10**1, 10**2, 10**3, 10**4]
for i in K:
    neigh = MultinomialNB(alpha= i, class_prior=None, fit_prior=True)
    neigh.fit(final_train, y_train)

    y_train_pred = neigh.predict_proba(final_train)[:,-1]
    y_cv_pred = neigh.predict_proba(final_cv)[:,-1]

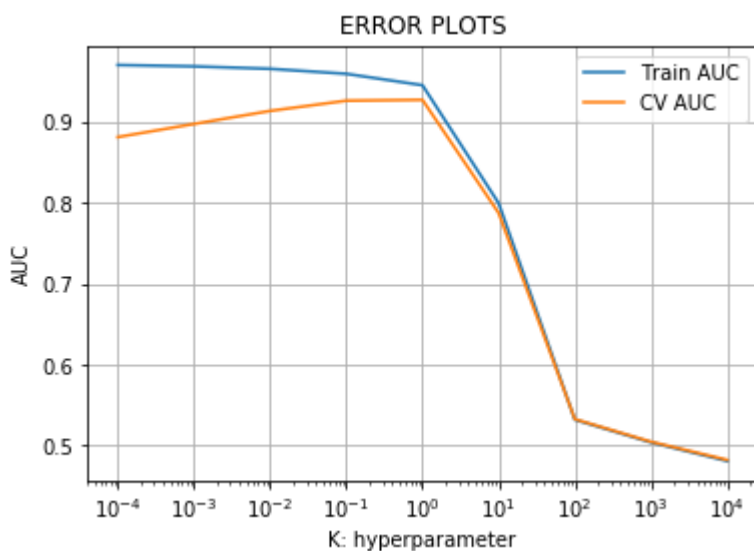
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

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

plt.semilogx(K, train_auc, label='Train AUC')
plt.semilogx(K, cv_auc, label='CV AUC')
plt.legend()
plt.grid()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
```

Out[96]:

Text(0.5,1,'ERROR PLOTS')



Best alpha = 1

Testing:

In [97]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

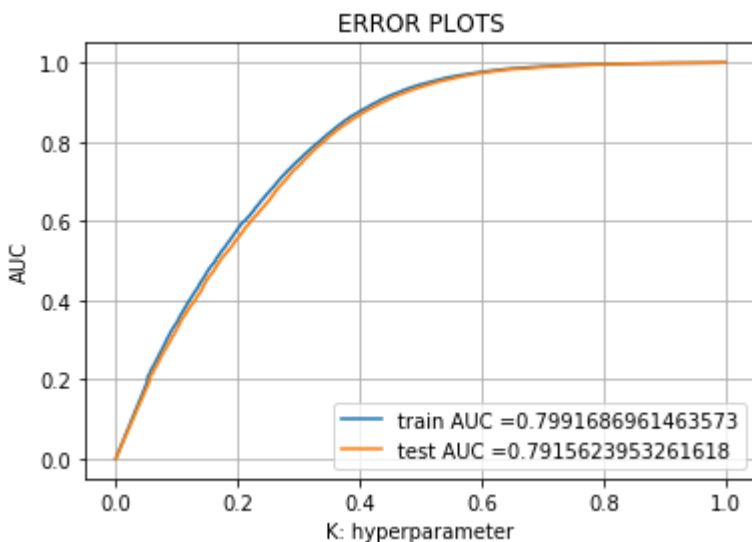
neigh = MultinomialNB(alpha= 10, class_prior=None, fit_prior=True)
neigh.fit(final_train, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train, neigh.predict_proba(final_train)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, neigh.predict_proba(final_test)[:,1])

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.grid()
plt.show()

print("="*100)

from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, neigh.predict(final_train)))
print("Test confusion matrix")
print(confusion_matrix(y_test, neigh.predict(final_test)))
```

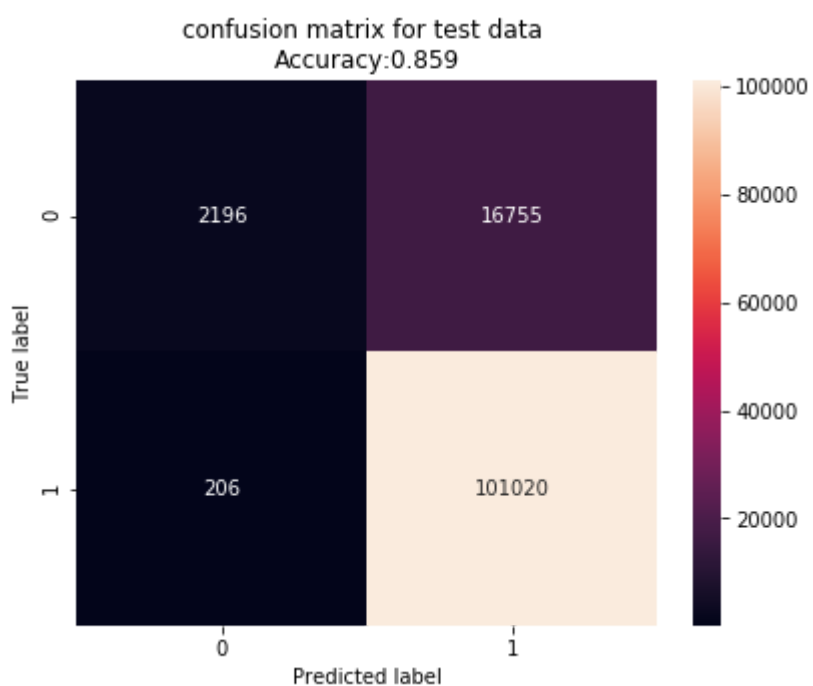
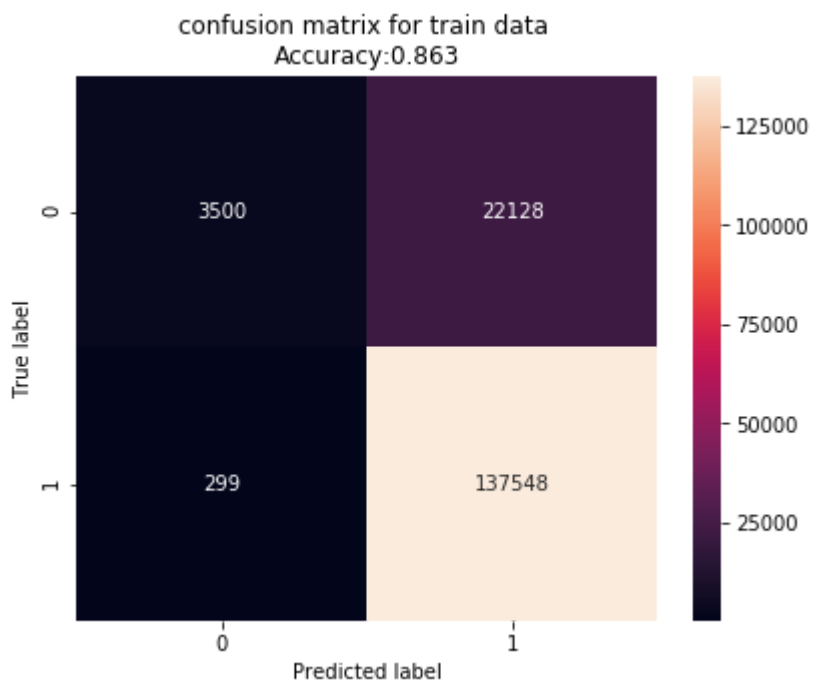


```
=====
=====
Train confusion matrix
[[ 3500  22128]
 [   299 137548]]
Test confusion matrix
[[ 2196  16755]
 [   206 101020]]
```


In [98]:

```
# Creates a confusion matrix for train data
cm = confusion_matrix(y_train, neigh.predict(final_train))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for train data \nAccuracy:{0:.3f}'.format(accuracy_score(y_train, neigh.predict(final_train))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# Creates a confusion matrix for test data
cm = confusion_matrix(y_test, neigh.predict(final_test))
cm_df = pd.DataFrame(cm)
plt.figure(figsize=(6.5,5))
sns.heatmap(cm_df, annot=True, fmt="d")
plt.title('confusion matrix for test data \nAccuracy:{0:.3f}'.format(accuracy_score(y_test, neigh.predict(final_test))))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

[6] Conclusions

In [99]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model", "Hyperparameter(alpha)", "Train AUC", "Test AUC"]

x.add_row(["BOW", 1, 0.81, 0.79])
x.add_row(["TF-IDF", 10, 0.86, 0.85])
x.add_row(["New Feature (BOW)", 1, 0.799, 0.791])

print(x)
```

Model	Hyperparameter(alpha)	Train AUC	Test AUC
BOW	1	0.81	0.79
TF-IDF	10	0.86	0.85
New Feature (BOW)	1	0.799	0.791