04affrnb

June 24, 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
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from prettytable import PrettyTable
from sklearn.metrics import accuracy_score
```

```
C:\Users\ACER\Anaconda3\lib\site-packages\gensim\utils.py:860: UserWarning: detected Windows; a
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:
```
# using SQLite Table to read data.
```

```
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data point
# 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 5


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[2]:

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

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

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

In [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  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                    Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM           Kim Cieszykowski  1348531200      1
        3  #oc-R11O5J5ZVQE25C  B005HG9ET0              Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBE1U     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 [5]: display[display['UserId']=='AZY10LLTJ71NX']
```

```
Out[5]:               UserId   ProductId                    ProfileName        Time  \
        80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

               Score                                               Text  COUNT(*)
        80638      5  I was recommended to try green tea extract to ...         5
```

```
In [6]: display['COUNT(*)'].sum()
```

```
Out[6]: 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 [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  \
        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 [8]: #Sorting data according to ProductId in ascending order
        sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

```
In [9]: #Deduplication of entries
        final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep=
        final.shape
```

```
Out[9]: (87775, 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]: 87.775
```

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 [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  \
         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 [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()
```

```
(87773, 10)
```

```
Out[13]: 1    73592
         0    14181
         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 [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)
```

```
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 [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)
```

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

```
In [16]: # 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 afraid
```

```
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)
```

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

In [19]:
```
#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 [20]:
```
#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 [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 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', 'mig
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
                    'won', "won't", 'wouldn', "wouldn't"])
```

```
In [22]: # 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 stopwo
             preprocessed_reviews.append(sentance.strip())
```

```
100%|| 87773/87773 [00:28<00:00, 3108.53it/s]
```

```
In [23]: preprocessed_reviews[1500]

Out[23]: 'way hot blood took bite jig lol'
```

5 [4] Splitting the data

```
In [24]: X = preprocessed_reviews
         Y = final['Score'].values
```

```
In [25]: #from sklearn.model_selection import train_test_split

         # Here we are splitting the data(X ,Y) into train, cross-validation and test data
         # X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=F
         X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30) # this is r
         X_train, X_cv, Y_train, Y_cv     = train_test_split(X_train, Y_train, test_size=0.30)
```

6 [5] Featurization

6.1 [5.1] BAG OF WORDS

```
In [26]: #BoW
         vectorizerb = CountVectorizer(min_df = 10)
         vectorizerb.fit(X_train) # fit has to happen only on train data
         print(vectorizerb.get_feature_names()[:20])# printing some feature names
         print("="*50)

         # we use the fitted CountVectorizer to convert the text to vector
```

```
X_train_bow = vectorizerb.transform(X_train)
X_cv_bow = vectorizerb.transform(X_cv)
X_test_bow = vectorizerb.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)
print("="*100)

print("the type of count vectorizer ")
print(type(X_train_bow))
print(type(X_cv_bow))
print(type(X_test_bow))
```

```
['aafco', 'ability', 'able', 'absent', 'absolute', 'absolutely', 'absolutly', 'absorb', 'absorl
==================================================
After vectorizations
(43008, 8103) (43008,)
(18433, 8103) (18433,)
(26332, 8103) (26332,)
=====================================================================================================
the type of count vectorizer
<class 'scipy.sparse.csr.csr_matrix'>
<class 'scipy.sparse.csr.csr_matrix'>
<class 'scipy.sparse.csr.csr_matrix'>
```

6.2 [5.2] TF-IDF

In [27]:
```
tfidf_vect = TfidfVectorizer(min_df=10)
tfidf_vect.fit(X_train)
print("some sample features ",tfidf_vect.get_feature_names()[0:10])
print('='*50)

# we use the fitted CountVectorizer to convert the text to vector
X_train_tfidf = tfidf_vect.transform(X_train)
X_cv_tfidf = tfidf_vect.transform(X_cv)
X_test_tfidf = tfidf_vect.transform(X_test)

print("After vectorizations")
print(X_train_tfidf.shape, Y_train.shape)
print(X_cv_tfidf.shape, Y_cv.shape)
print(X_test_tfidf.shape, Y_test.shape)
print("="*100)

print("the type of count vectorizer ")
```

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

```
some sample features  ['aafco', 'ability', 'able', 'absent', 'absolute', 'absolutely', 'absolu
==================================================
After vectorizations
(43008, 8103) (43008,)
(18433, 8103) (18433,)
(26332, 8103) (26332,)
==============================================================================================
the type of count vectorizer
<class 'scipy.sparse.csr.csr_matrix'>
<class 'scipy.sparse.csr.csr_matrix'>
<class 'scipy.sparse.csr.csr_matrix'>
```

7 [6] 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.appliedaico
<li>Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001</
<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 fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

Note: Data Leakage

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

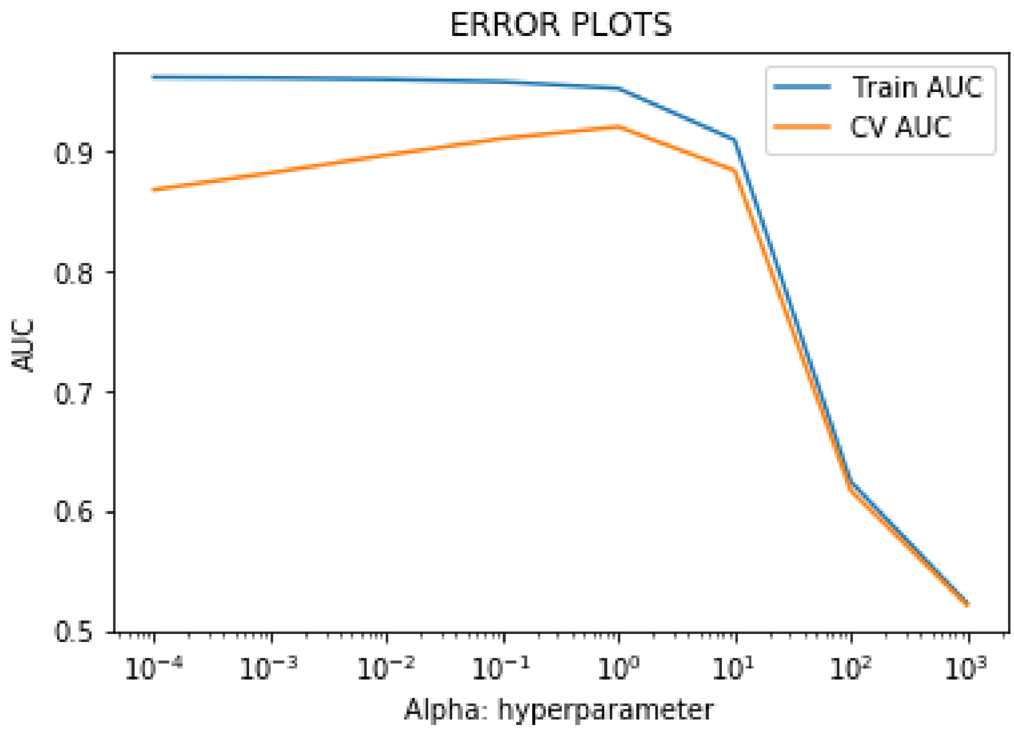
8 [7] Applying Multinomial Naive Bayes

8.1 [7.1] Applying Naive Bayes on BOW, SET 1

8.2 Hyperparameter tuning using GridSearch

```
In [31]: # https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearc

         multib = MultinomialNB(class_prior = [0.5,0.5])
         alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
         parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}
         clf = GridSearchCV(multib, parameters, cv=3, scoring='roc_auc', n_jobs=-1)
         clf.fit(X_train_bow, Y_train)

         train_auc_bow = clf.cv_results_['mean_train_score']
         cv_auc_bow = clf.cv_results_['mean_test_score']
```

```
plt.plot(alpha, train_auc_bow, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039 ,alpha=
plt.plot(alpha, cv_auc_bow, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039


plt.legend()
plt.xscale('log')
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



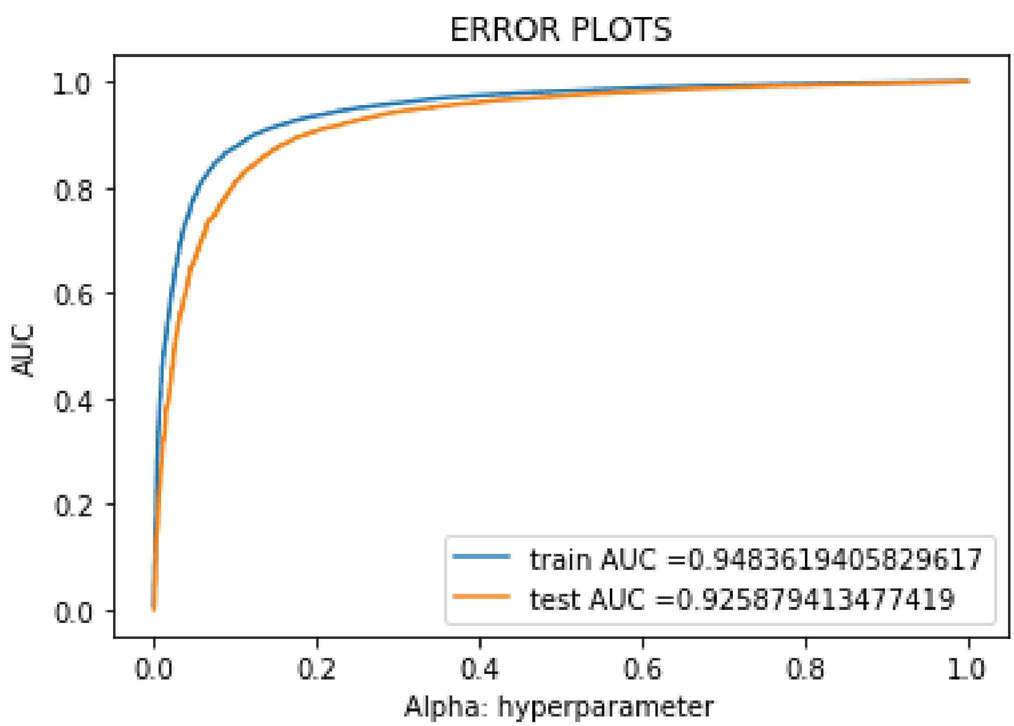

```
In [32]: print(clf.best_estimator_)
```

```
MultinomialNB(alpha=1, class_prior=[0.5, 0.5], fit_prior=True)
```

```
In [33]: #here we are choosing the best_k based on GridSearchCV results
         best_alpha_bbow = 1
```

8.3 Testing with test data

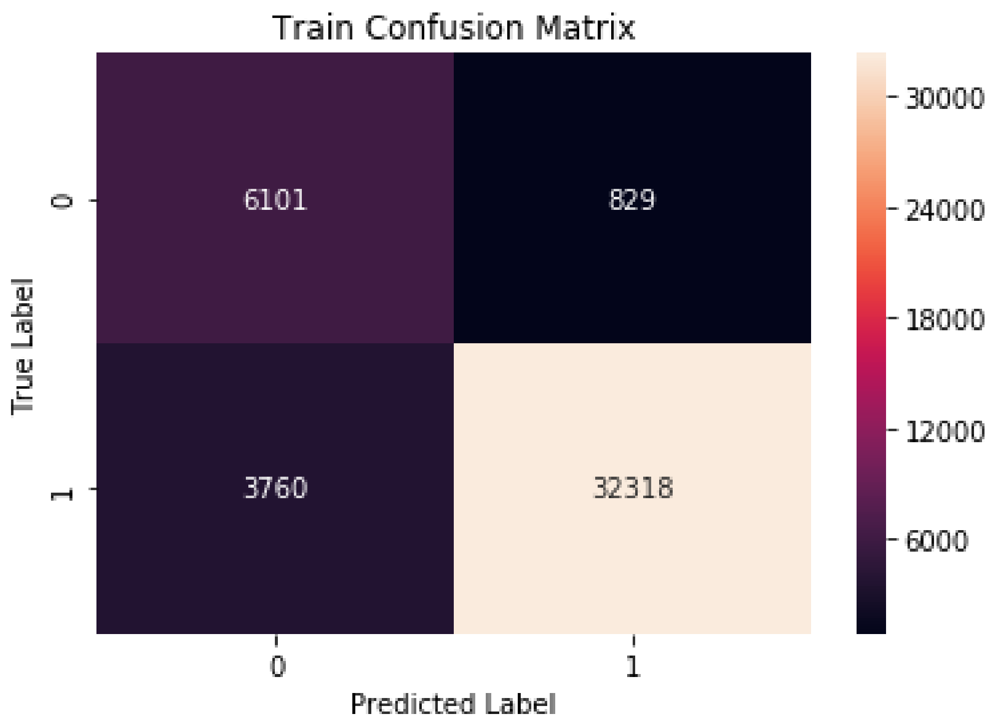
```
In [34]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sk

         multib = MultinomialNB(class_prior = [0.5,0.5])
         multib.fit(X_train_bow, Y_train)
         # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
         # not the predicted outputs

         train_fpr_bow, train_tpr_bow, thresholds_bow = roc_curve(Y_train, multib.predict_proba
         test_fpr_bow, test_tpr_bow, thresholds_bow = roc_curve(Y_test, multib.predict_proba(X_

         plt.plot(train_fpr_bow, train_tpr_bow, label="train AUC ="+str(auc(train_fpr_bow, trai
         plt.plot(test_fpr_bow, test_tpr_bow, label="test AUC ="+str(auc(test_fpr_bow, test_tpr
         plt.legend()
         plt.xlabel("Alpha: hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```



8.4 Confusion Matrix for BOW

```
In [35]: #Creates a confusion matrix for Train data
         cm = confusion_matrix(Y_train, multib.predict(X_train_bow))
```

```
#Confusion matrix using Heatmap
sns.heatmap(cm, annot=True, fmt='d')

plt.title('Train Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```



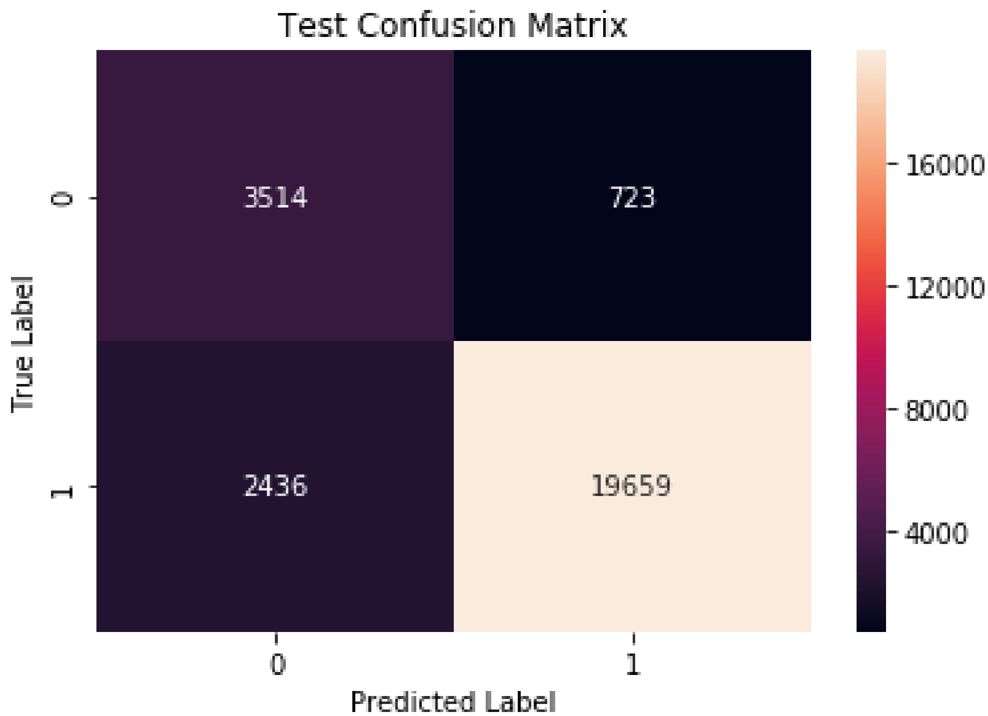
8.4.1 Accuracy Score for Train Data

```
In [36]: predict = multib.predict(X_train_bow)
         acc_train_b = accuracy_score(Y_train, predict)
         print("Accuracy Score : ", acc_train_b)
```

```
Accuracy Score :  0.8932989211309523
```

```
In [37]: #Creates a confusion matrix for Test data
         cm = confusion_matrix(Y_test, multib.predict(X_test_bow))
         #Confusion matrix using Heatmap
         sns.heatmap(cm, annot=True, fmt='d')

         plt.title('Test Confusion Matrix')
         plt.ylabel('True Label')
```

```
plt.xlabel('Predicted Label')
plt.show()
```



8.4.2 Accuracy Score for Test data

```
In [39]: predict = multib.predict(X_test_bow)
         acc_test_b = accuracy_score(Y_test, predict)
         print("Accuracy Score : ", acc_test_b)

Accuracy Score :  0.8800319003493848
```

8.4.3 [7.1.1] Top 10 important features of positive class from SET 1

```
In [40]: # this code is copied from here:https://stackoverflow.com/questions/26976362/how-to-g

         def most_informative_feature_for_binary_classification(vectorizer, classifier, n=10):
             class_labels = classifier.classes_
             feature_names = vectorizer.get_feature_names()
             topn_class1 = sorted(zip(classifier.coef_[0], feature_names))[-n:]


             for coef, feat in reversed(topn_class1):
                 print (class_labels[1], coef, feat)
```

```
most_informative_feature_for_binary_classification(vectorizerb, multib)
```

```
1 -3.6772833631614876 not
1 -4.48719912892669 like
1 -4.6215968128791705 good
1 -4.696074083984588 great
1 -4.834601460833525 one
1 -4.923371370373834 taste
1 -4.9556187206762345 coffee
1 -5.008829152726275 flavor
1 -5.014907514529412 would
1 -5.047031419348574 love
```

8.4.4 [7.1.2] Top 10 important features of negative class from SET 1

In [46]:
```
multib.classes_
```

Out[46]:
```
array([0, 1], dtype=int64)
```

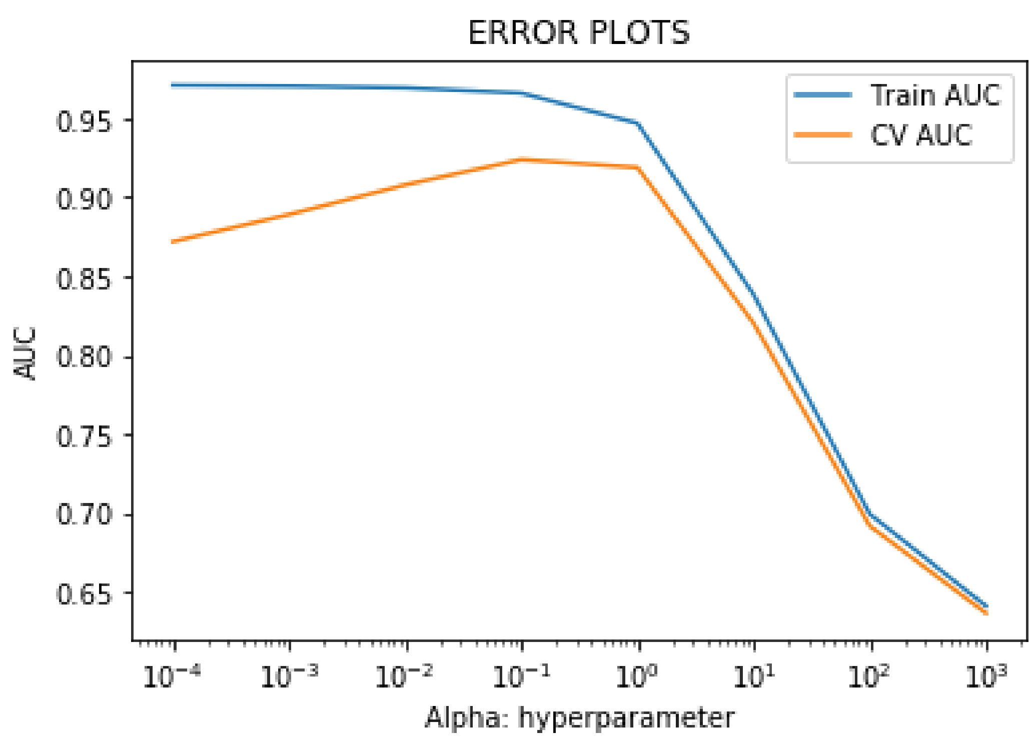
In [41]:
```
# this code is copied from here:https://stackoverflow.com/questions/26976362/how-to-g

def most_informative_feature_for_binary_classification(vectorizer, classifier, n=10):
    class_labels = classifier.classes_
    feature_names = vectorizer.get_feature_names()
    topn_class2 = sorted(zip(classifier.coef_[0], feature_names))[:n]

    for coef, feat in topn_class2:
        print( class_labels[0], coef, feat)


most_informative_feature_for_binary_classification(vectorizerb, multib)
```

```
0 -14.103870900338995 returnable
0 -13.41072371977905 blech
0 -13.41072371977905 canceled
0 -13.41072371977905 improperly
0 -13.41072371977905 mealy
0 -13.41072371977905 nastiest
0 -13.41072371977905 shudder
0 -13.41072371977905 spat
0 -13.005258611670886 aweful
0 -13.005258611670886 beaks
```

8.5 [7.2] Applying Naive Bayes on TFIDF, SET 2

8.6 Hyperparameter tuning with Train data

```
In [43]: # https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearc
         multit = MultinomialNB(class_prior= [0.5,0.5])
         alpha = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
         parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]}
         clf = GridSearchCV(multit, parameters, cv=3, scoring='roc_auc', n_jobs=-1)
         clf.fit(X_train_tfidf, Y_train)

         train_auc_tfidf = clf.cv_results_['mean_train_score']
         cv_auc_tfidf = clf.cv_results_['mean_test_score']


         plt.plot(alpha, train_auc_tfidf, label='Train AUC')
         # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
         plt.plot(alpha, cv_auc_tfidf, label='CV AUC')
         # this code is copied from here: https://stackoverflow.com/a/48803361/4084039

         plt.legend()
         plt.xscale('log')
         plt.xlabel("Alpha: hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```



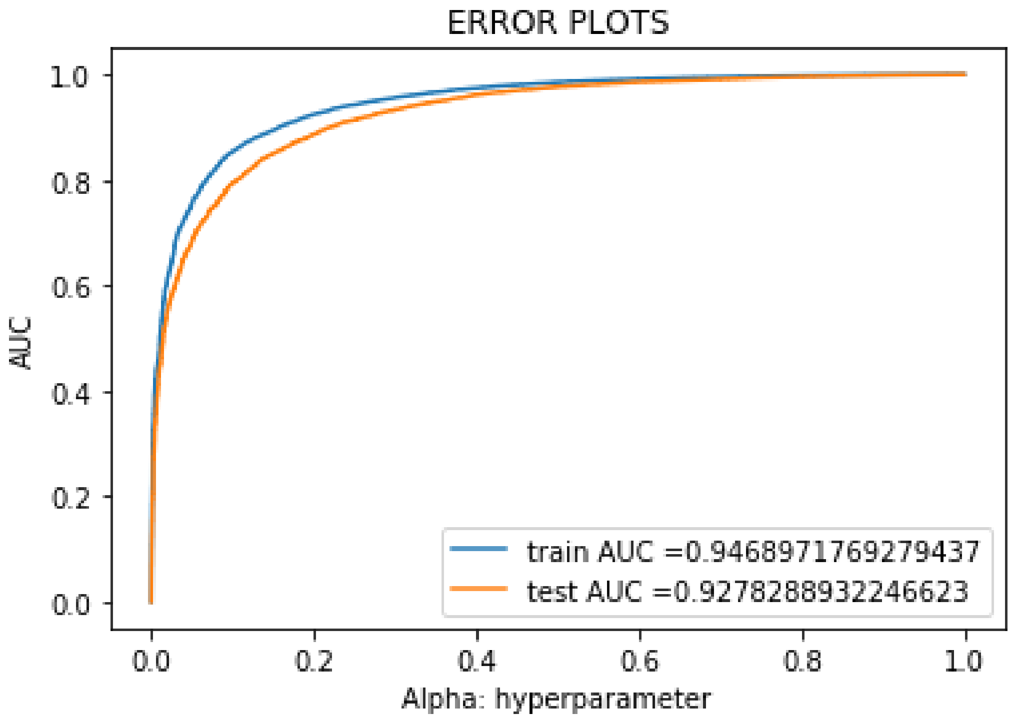
```
In [44]: print(clf.best_estimator_)
```

```
MultinomialNB(alpha=0.1, class_prior=[0.5, 0.5], fit_prior=True)
```

```
In [45]: #here we are choosing the best_k based on GridSearchCV results
         best_alpha_btfidf = 0.1
```

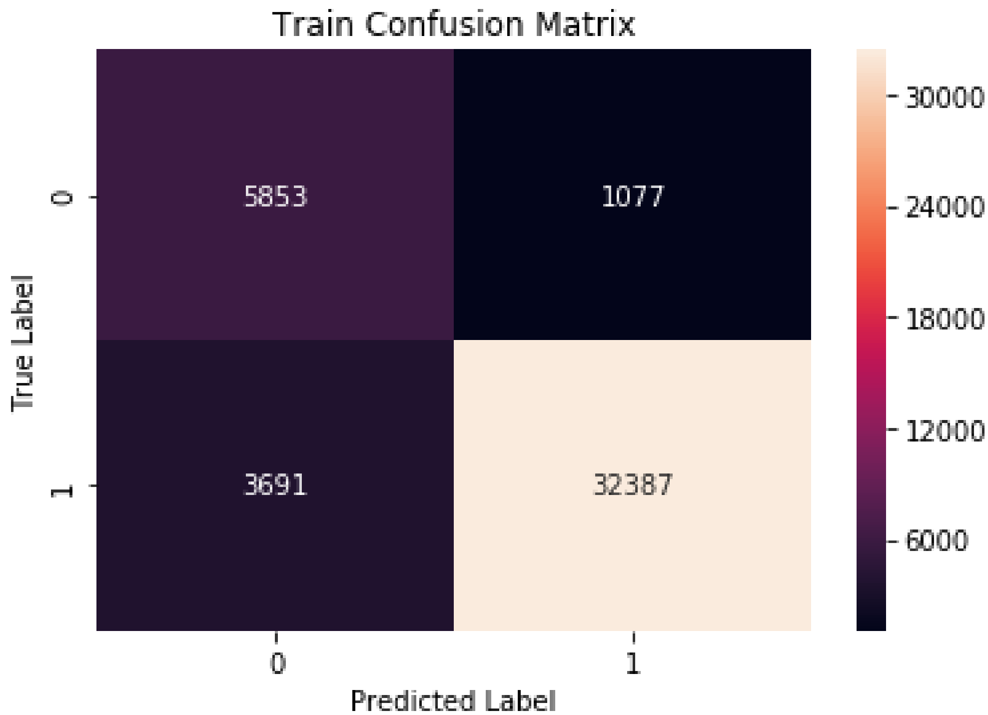
8.7 Testing with test data

```
In [46]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sk

         multit = MultinomialNB(class_prior = [0.5,0.5])
         multit.fit(X_train_tfidf, Y_train)
         # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
         # not the predicted outputs

         train_fpr_tfidf, train_tpr_tfidf, thresholds_tfidf = roc_curve(Y_train, multit.predict
         test_fpr_tfidf, test_tpr_tfidf, thresholds_tfidf = roc_curve(Y_test, multit.predict_p

         plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="train AUC ="+str(auc(train_fpr_tfid
         plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="test AUC ="+str(auc(test_fpr_tfidf, te
         plt.legend()
         plt.xlabel("Alpha: hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```



8.8 Confusion Matrix for TFIDF

In [47]:
```
#Creates a confusion matrix for Train data
cm = confusion_matrix(Y_train, multit.predict(X_train_tfidf))
#Confusion matrix using Heatmap
sns.heatmap(cm, annot=True, fmt='d')

plt.title('Train Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

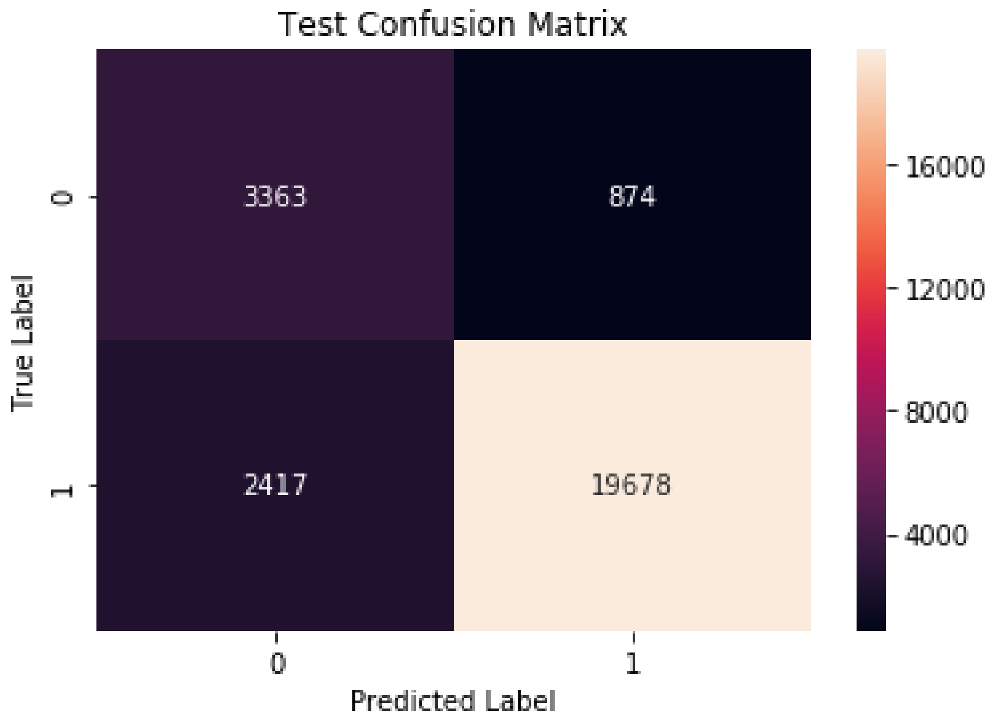


8.8.1 Accuracy Score for Train data

```
In [49]: predict = multit.predict(X_train_tfidf)
         acc_train_tf = accuracy_score(Y_train, predict)
         print("Accuracy Score : ", acc_train_tf)
```

```
Accuracy Score :  0.8891369047619048
```

```
In [50]: #Creates a confusion matrix for Test data
         cm = confusion_matrix(Y_test, multit.predict(X_test_tfidf))
         #Confusion matrix using Heatmap
         sns.heatmap(cm, annot=True, fmt='d')

         plt.title('Test Confusion Matrix')
         plt.ylabel('True Label')
         plt.xlabel('Predicted Label')
         plt.show()
```



8.8.2 Accuracy Score for Test Data

```
In [51]: predict = multit.predict(X_test_tfidf)
         acc_test_tf = accuracy_score(Y_test, predict)
         print("Accuracy Score : ", acc_test_tf)
```

```
Accuracy Score :  0.8750189883032052
```

8.8.3 [7.2.1] Top 10 important features of positive class from SET 2

```
In [52]: # this code is copied from here:https://stackoverflow.com/questions/26976362/how-to-g

         def most_informative_feature_for_binary_classification(vectorizer, classifier, n=10):
             class_labels = classifier.classes_
             feature_names = vectorizer.get_feature_names()
             topn_class1 = sorted(zip(classifier.coef_[0], feature_names))[-n:]


             for coef, feat in reversed(topn_class1):
                 print (class_labels[1], coef, feat)


         most_informative_feature_for_binary_classification(tfidf_vect, multit)
```

```
1 -4.817294371586819 not
1 -5.125059733533149 great
1 -5.196602093573375 good
1 -5.26298745882407 coffee
1 -5.263954150864575 like
1 -5.37734450820485 tea
1 -5.378192751733905 love
1 -5.503437680202993 taste
1 -5.5044188191596435 one
1 -5.521123619042673 flavor
```

8.8.4 [7.2.2] Top 10 important features of negative class from SET 2

```
In [53]: # this code is copied from here:https://stackoverflow.com/questions/26976362/how-to-g

         def most_informative_feature_for_binary_classification(vectorizer, classifier, n=10):
             class_labels = classifier.classes_
             feature_names = vectorizer.get_feature_names()
             topn_class2 = sorted(zip(classifier.coef_[0], feature_names))[:n]

             for coef, feat in topn_class2:
                 print( class_labels[0], coef, feat)


         most_informative_feature_for_binary_classification(tfidf_vect, multit)
```

```
0 -12.09891078807189 returnable
0 -12.042439325556465 improperly
0 -11.958633693771326 blech
0 -11.952731852161447 shudder
0 -11.921041600778596 nastiest
0 -11.901561160756577 mealy
0 -11.900442405175802 cheaply
0 -11.864863362301884 torture
0 -11.855468797827166 redeeming
0 -11.853600717543054 aweful
```

9 [8] Conclusions

```
In [54]: # Please compare all your models using Prettytable library
         name= ["Naive Bayes for BOW", "Naive Bayes for TFIDF"]
         best_alpha = [best_alpha_bbow, best_alpha_btfidf]
         number = [1,2]
         accuracy1 = [acc_train_b, acc_train_tf]
         accuracy2 = [acc_test_b, acc_test_tf]
         #Initializa Prettytable
```

```python
ptable = PrettyTable()
ptable.add_column("Index", number)
ptable.add_column("Model", name)
ptable.add_column("Value for Alpha", best_alpha)
ptable.add_column("Train Accuracy", accuracy1)
ptable.add_column("Test Accuracy", accuracy2)

print(ptable)
```

```
+-------+----------------------+-----------------+--------------------+--------------------+
| Index |        Model         | Value for Alpha |   Train Accuracy   |   Test Accuracy    |
+-------+----------------------+-----------------+--------------------+--------------------+
|   1   |  Naive Bayes for BOW |        1        | 0.8932989211309523 | 0.8800319003493848 |
|   2   | Naive Bayes for TFIDF |      0.1        | 0.8891369047619048 | 0.8750189883032052 |
+-------+----------------------+-----------------+--------------------+--------------------+
```

1. Value of Hyperparameter(alpha) is 1 for BOW and 0.1 for TFIDF
2. BOW model has more accuracy score than TFIDF so BOW is better than TFIDF