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.

[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 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 [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 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 500000""", co
n)
# 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	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862400
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976000

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	121901760

In [3]:

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

Out[3]:

```
'\ndisplay = pd.read_sql_query("""\nSELECT UserId, ProductId, ProfileName, Time, Score, Text,
COUNT(*)\nFROM Reviews\nGROUP BY UserId\nHAVING COUNT(*)>1\n""", con)\n'
```

In [4]:

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

In [5]:

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

In [6]:

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

[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]:

```
'display= pd.read_sql_query("""\nSELECT *\nFROM Reviews\nWHERE Score != 3 AND
UserId="AR5J8UI46CURR"\nORDER BY ProductID\n""", con)\ndisplay.head()\n'
```

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

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=False, kind='qui
cksort', na_position='last')
```

In [9]:

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inpl
ace=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 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]:

```
'display= pd.read_sql_query("""\nSELECT *\nFROM Reviews\nWHERE Score != 3 AND Id=44737 OR
Id=64422\nORDER BY ProductID\n""", con)\n\ndisplay.head()\n'
```

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 obsereved to be better than Porter Stemming)

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

In [14]:

```
# https://stackoverflow.com/a/47091490/4084039
import re
from bs4 import BeautifulSoup
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 [15]:

```
# 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', 'ourselves', 'you', "y
ou'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', 'e
ach', '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', "do
esn'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"])
```

Time Base Spliting of data

In [16]:

```
SORT_DATA = final.sort_values("Time")
```

In [17]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(SORT_DATA['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 stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|
████████████████████████████████████████████████████████████████████████████████████████████████
██████| 364171/364171 [13:04<00:00, 464.30it/s]
```

In [18]:

```
final.columns
```

Out[18]:

```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

In [19]:

```
DATA = np.array(preprocessed_reviews[0:50000])
LABEL  = np.array(SORT_DATA['Score'][0:50000])
```

In [20]:

```
from sklearn.model_selection import train_test_split
X_train_temp, X_TEST, Y_train_temp, Y_TEST = train_test_split(DATA,LABEL, test_size=0.33,stratify=L
ABEL)
X_TRAIN, X_CV, Y_TRAIN, Y_CV = train_test_split(X_train_temp, Y_train_temp,
test_size=0.33,stratify=Y_train_temp)
```

In [ ]:

```
#X_TRAIN, X_CV, Y_TRAIN, Y_CV = train_test_split(X_train_temp, Y_train_temp,
test_size=0.33,stratify=Y_train_temp)
```

[3.2] Preprocessing Review Summary

In [ ]:

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

[4] Featurization

[4.1] BAG OF WORDS

In [ ]:

```
#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])
'''
```

[4.2] Bi-Grams and n-Grams.

In [ ]:

```
'''
#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/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

# 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_counts.get_shape()[1])
'''
```

[4.3] TF-IDF

In [ ]:

```
'''
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()[0:10])
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.get_shape()[
1])
'''
```

[4.4] Word2Vec

In [ ]:

```
# 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 [ ]:

```
# 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.bin',
binary=True)
        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, to train your
own w2v ")
        '''
```

In [ ]:

```
'''
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])
'''
```

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

[4.4.1.1] Avg W2v

In [ ]:

```
# 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 change this
```

```
to 300 if you use google's w2v
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
'''
```

[4.4.1.2] TFIDF weighted W2v

In [ ]:

```
# 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 [ ]:

```
'''
# 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 list
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
    '''
```

[5] Assignment 5: Apply Logistic Regression

1. **Apply Logistic Regression 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)
   - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
   - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**

   - Find the best hyper parameter which will give the maximum AUC value
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Pertubation Test**

    - Get the weights W after fit your model with the data X i.e Train data.
    - Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e)
    - Fit the model again on data X' and get the weights W'
    - Add a small eps value(to eliminate the divisible by zero error) to W and W' i.e W=W+10^-6 and W' = W'+10^-6
    - Now find the % change between W and W' (| (W-W') / (W) |)*100)
    - Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in the values of percentage_change_vector
    - Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is sudden rise from 1.3 to 34.6, now calculate the 99.1, 99.2, 99.3,..., 100th percentile values and get the proper value after which there is sudden rise the values, assume it is 2.5
    - Print the feature names whose % change is more than a threshold x(in our example it's 2.5)

4. **Sparsity**

    - Calculate sparsity on weight vector obtained after using L1 regularization

    NOTE: Do sparsity and multicollinearity for any one of the vectorizers. Bow or tf-idf is recommended.

5. **Feature importance**

    - Get top 10 important features for both positive and negative classes separately.

6. **Feature engineering**

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

7. **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.
    - 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 with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

8. **Conclusion**

    - 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 link

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.

Applying Logistic Regression

In [41]:

```
def LOG_REG(TRAIN_DATA,TRAIN_LABEL,CV_DATA,CV_LABEL,TEST_DATA,TEST_LABEL,PENALTY):

    from sklearn.metrics import roc_auc_score
    import matplotlib.pyplot as plt
    from sklearn.linear_model import LogisticRegression
    import numpy as NP

    C= [10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2]
```

```
    TRAIN_AUC = []
    CV_AUC = []

    for i in C:
        OBJ = LogisticRegression(C=i,penalty=PENALTY,class_weight='balanced')
        OBJ.fit(TRAIN_DATA,TRAIN_LABEL)

        Y_TRAIN_PRED =  list(OBJ.predict_proba(TRAIN_DATA)[:,1])
        TRAIN_AUC.append(roc_auc_score(TRAIN_LABEL,Y_TRAIN_PRED))

        Y_CV_PRED =  list(OBJ.predict_proba(CV_DATA)[:,1])
        CV_AUC.append(roc_auc_score(CV_LABEL,Y_CV_PRED))

    plt.plot(np.log10(C),TRAIN_AUC, label='Train AUC')
    plt.plot(np.log10(C),CV_AUC, label='CV AUC')
    plt.legend()
    plt.xlabel("C: hyperparameter in LOG Scale")
    plt.ylabel("AUC")
    plt.title("AUC PLOTS")
    plt.show()

    BEST_C = C[CV_AUC.index(max(CV_AUC))]
    print("BEST_C is {}".format(BEST_C))

    OBJ2 = LogisticRegression(C=BEST_C,penalty=PENALTY,class_weight = 'balanced')
    OBJ2.fit(TRAIN_DATA,TRAIN_LABEL)

    PRED_TEST=list(OBJ2.predict(TEST_DATA))

    PRED_TEST = np.array(PRED_TEST)

    PRED_TRAIN = []

    PRED_TRAIN=list(OBJ2.predict(TRAIN_DATA))

    PRED_TRAIN = np.array(PRED_TRAIN)



    #OBJ2 = LogisticRegression(C=BEST_C,penalty=PENALTY)
    #OBJ2.fit(TRAIN_DATA,TRAIN_LABEL)
    TRAIN_PROBA=  list(OBJ2.predict_proba(TRAIN_DATA)[:,1])
    TEST_PROBA =  list(OBJ2.predict_proba(TEST_DATA)[:,1])


    from sklearn import metrics
    fpr_2,tpr_2,tr_2 = metrics.roc_curve(TEST_LABEL,TEST_PROBA)
    fpr_1,tpr_1,tr_1 = metrics.roc_curve(TRAIN_LABEL,TRAIN_PROBA)
    lw=2
    area_train = metrics.auc(fpr_1, tpr_1)
    area_test = metrics.auc(fpr_2, tpr_2)
    plt.plot(fpr_2, tpr_2, color='darkorange',lw=lw, label='ROC curve of Test data (area = %0.2f)'
% area_test)
    plt.plot(fpr_1, tpr_1, color='green',lw=lw, label='ROC curve of Train data(area = %0.2f)' % are
a_train)
    plt.legend()
    plt.title("ROC CURVE")


    from sklearn.metrics import confusion_matrix
    import seaborn as sns

    plt.figure()
    cm = confusion_matrix(TRAIN_LABEL,PRED_TRAIN)
    class_label = ["negative", "positive"]
    df_cm_test = pd.DataFrame(cm, index = class_label, columns = class_label)
    sns.heatmap(df_cm_test , annot = True, fmt = "d")
    plt.title("Confusiion Matrix for TRAIN data")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()

    #from sklearn.metrics import confusion_matrix
    #import seaborn as sns
```

```
    plt.figure()
    cm = confusion_matrix(TEST_LABEL,PRED_TEST)
    class_label = ["negative", "positive"]
    df_cm_test = pd.DataFrame(cm, index = class_label, columns = class_label)
    sns.heatmap(df_cm_test , annot = True, fmt = "d")
    plt.title("Confusiion Matrix for test data")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()

    if(PENALTY=='l1'):
        return ((OBJ2.coef_.size-NP.count_nonzero(OBJ2.coef_))/OBJ2.coef_.size)*100
```

[5.1] Logistic Regression on BOW, SET 1

In [23]:

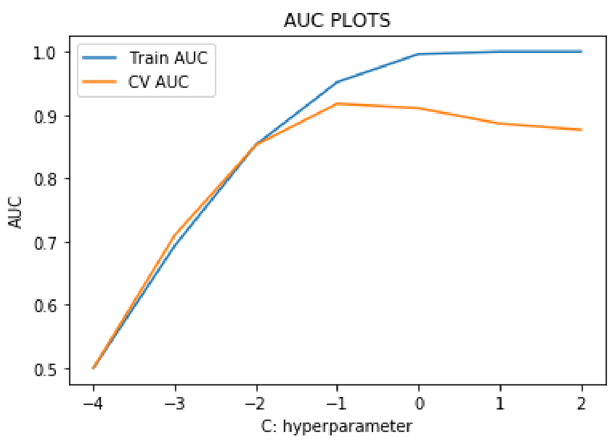
```
#.....CONVERT it into BOW VECTORS....
from sklearn.feature_extraction.text import CountVectorizer
OBJ_BOW = CountVectorizer()
OBJ_BOW.fit(X_TRAIN)


X_TRAIN_BOW =OBJ_BOW.transform(X_TRAIN)
X_CV_BOW = OBJ_BOW.transform(X_CV)
X_TEST_BOW = OBJ_BOW.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)
```

```
After vectorizations
(22445, 29382) (22445,)
(11055, 29382) (11055,)
(16500, 29382) (16500,)
====================================================================================================
```

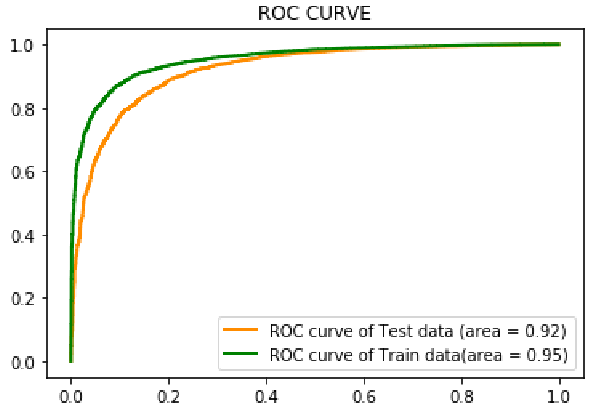
[5.1.1] Applying Logistic Regression with L1 regularization on BOW, SET 1

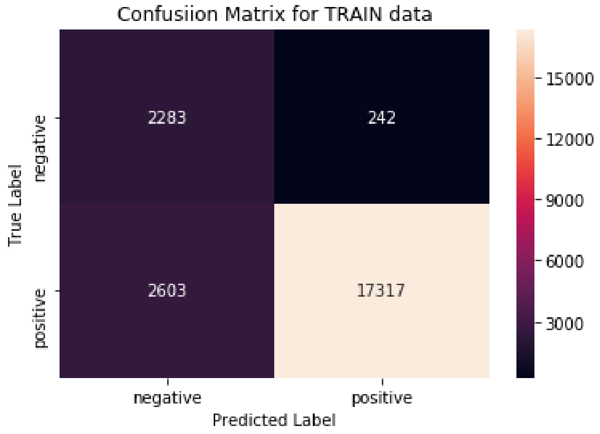
In [42]:

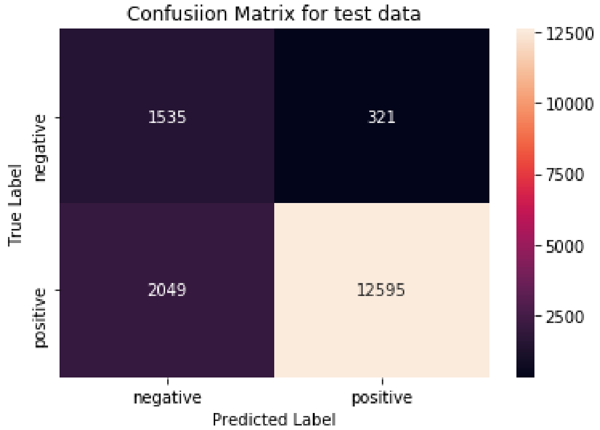
```
# Please write all the code with proper documentation
SPARSITY = LOG_REG(X_TRAIN_BOW,Y_TRAIN,X_CV_BOW,Y_CV,X_TEST_BOW,Y_TEST,'l1')
```




```
BEST_C is 0.1
```







[5.1.1.1] Calculating sparsity on weight vector obtained using L1 regularization on BOW, SET 1

In [25]:

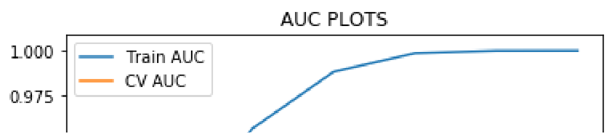
```
print("Sparsity on weight vector obtained using L1 regularization on BOW is {}".format(SPARSITY))
```

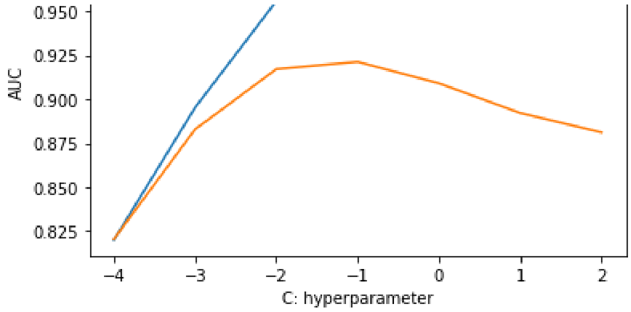
```
Sparsity on weight vector obtained using L1 regularization on BOW is 97.8626369886325
```

[5.1.2] Applying Logistic Regression with L2 regularization on BOW, SET 1

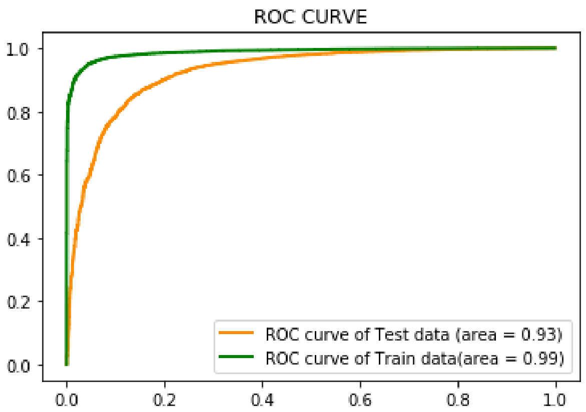
In [123]:

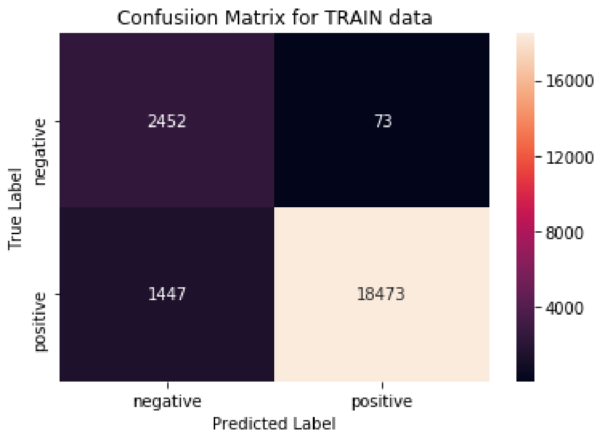
```
LOG_REG(X_TRAIN_BOW,Y_TRAIN,X_CV_BOW,Y_CV,X_TEST_BOW,Y_TEST,'l2')
```

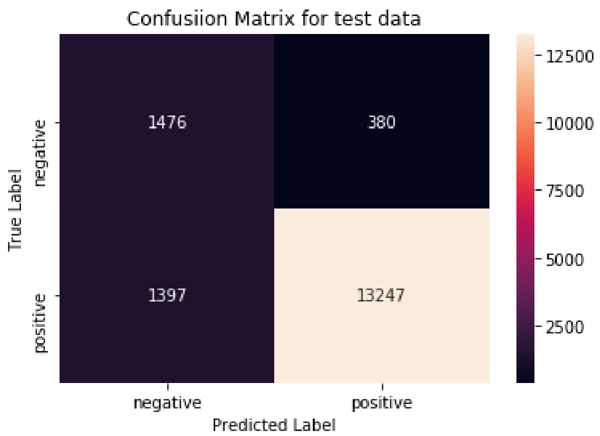





```
BEST_C is 0.1
```







[5.1.3] Feature Importance on BOW, SET 1

In [27]:

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

from sklearn.linear_model import LogisticRegression
```

```
LR = LogisticRegression(penalty='l2',C=0.1,class_weight = 'balanced')
LR.fit(X_TRAIN_BOW,Y_TRAIN)
WT = LR.coef_ #WEIGHTS

count_vect = CountVectorizer()
p = count_vect.fit_transform(X_TRAIN)

p = pd.DataFrame(WT.T,columns=['+ve'])
p['feature'] = count_vect.get_feature_names()
```

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

In [28]:

```
q = p.sort_values(by = '+ve',ascending= False)
print("Top 10  important features of positive class", np.array(q['feature'][:10]))
```

```
Top 10  important features of positive class ['delicious' 'wonderful' 'loves' 'perfect' 'best' 'gr
eat' 'excellent'
 'highly' 'smooth' 'favorite']
```

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

In [29]:

```
# Please write all the code with proper documentation
print("Top 10  important features of negative class",np.array(q.tail(10)['feature']))
```

```
Top 10  important features of negative class ['poor' 'weak' 'awful' 'bland' 'unfortunately'
'disappointing' 'horrible'
 'terrible' 'disappointed' 'worst']
```

[5.1.2.1] Performing pertubation test (multicollinearity check) on BOW, SET 1

CREATING NEW DATA by ADDING NOISE

In [30]:

```
import numpy as NP
NEW_TRAIN_BOW =X_TRAIN_BOW.astype(float)
NEW_TRAIN_BOW.data+=NP.random.uniform(-0.001,0.001,1)
```

Finding Weights of New Data

In [31]:

```
LR = LogisticRegression(penalty='l2',C=0.1,class_weight = 'balanced')
LR.fit(NEW_TRAIN_BOW,Y_TRAIN)
WT2 = LR.coef_ #Wt of new DATA
```

In [32]:

```
WT += 10**-6
WT2 += 10**-6
```

finding % change between W and W' (| (W-W') / (W) |)*100)

In [33]:

```
PERCENT_CHANGE = abs( (WT-WT2) / (WT) )*100
```

In [80]:

```
np.max(PERCENT_CHANGE)
```

Out[80]:

```
328.1213951173647
```

In [34]:

```
VALUES=[]
for i in range(0,101,10):
    VALUES.append(np.percentile(PERCENT_CHANGE,i))
```

In [35]:

```
len(VALUES)
```

Out[35]:

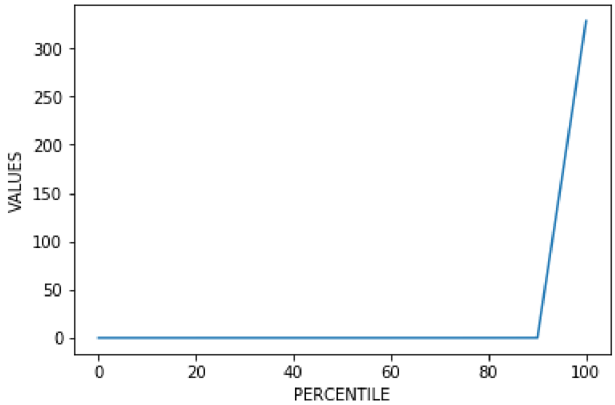
```
11
```

In [36]:

```
plt.plot(range(0,101,10),VALUES)
plt.xlabel("PERCENTILE")
plt.ylabel("VALUES")
```

Out[36]:

```
Text(0,0.5,'VALUES')
```



There is sudden rise between 90th and 100th percentile values.

In [99]:

```
VALUES = []
for i in np.linspace(99,100,500):
    VALUES.append(np.percentile(PERCENT_CHANGE,i))
```

In [102]:

```
j = 0
TEMP=np.linspace(99,100,500)
for i in range(0,499):
    if VALUES[i+1]-VALUES[i]>30:
        print('{} Percentile value is --------> {}'.format(TEMP[i],VALUES[j]))
    j =j+1
```

```
99.9879759519038 Percentile value is --------> 85.7233289296949
99.99599198396794 Percentile value is --------> 135.7505209718152
99.99799599198397 Percentile value is --------> 214.85388703550854
```

From above it is clearly obseverable that there is sudden rise from value 85 to 135 to 214 .

Printing feature words whose % change is more than threshold.....in my case(355)

In [103]:

```
TEMP = np.where(PERCENT_CHANGE>99.98)
```

In [104]:

```
TEMP
```

Out[104]:

```
(array([0, 0, 0, 0], dtype=int64),
 array([  628, 10077, 26369, 28002], dtype=int64))
```

In [105]:

```
OB = count_vect.get_feature_names()

WORDS=[]

for i in TEMP[1]:
    WORDS.append(OB[i])
```

In [106]:

```
print(WORDS)
```

```
['alcholic', 'fop', 'tiger', 'vibrancy']
```

OR ...(Below I have directly printed those feature words whose having Percentage change greater than 30%

It is noticable that all those feature(words) are present above.

In [107]:

```
TEMP2=np.where(PERCENT_CHANGE>30)[1]
WORDS2 = []
for i in TEMP2:
    WORDS2.append(OB[i])
```

In [108]:

```
print(WORDS2)
```

```
['acorn', 'adept', 'alcholic', 'alimentary', 'altria', 'aquired', 'cecco', 'cilatro',
'circulatory', 'conchiglie', 'creamettes', 'deodorize', 'extrusion', 'fop', 'incumbents', 'injun',
'mcmeal', 'megabox', 'meh', 'minn', 'morris', 'napoli', 'obsolete', 'pigging', 'pilau',
'pretense', 'racconto', 'regimented', 'revel', 'rizopia', 'steeping', 'tiger', 'toasted', 'tongs',
'tribal', 'vibrancy', 'workstyles']
```

[5.2] Logistic Regression on TFIDF, SET 2

In [109]:

```
from sklearn.feature_extraction.text import CountVectorizer
OBJ_TFIDF = TfidfVectorizer(ngram_range=(1,2), min_df=10)
OBJ_TFIDF.fit(X_TRAIN)




X_TRAIN_TFIDF =OBJ_TFIDF.transform(X_TRAIN)
```

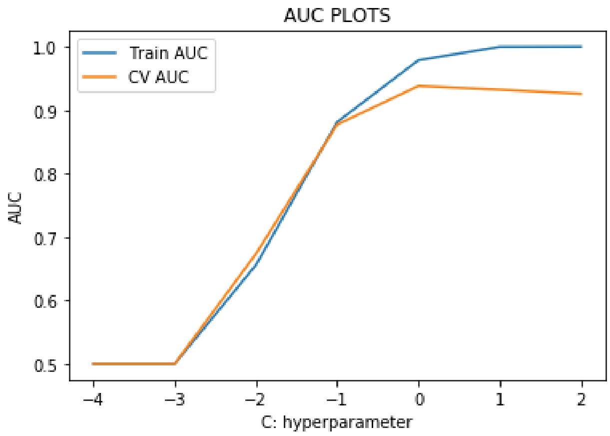
```
X_CV_TFIDF = OBJ_TFIDF.transform(X_CV)
X_TEST_TFIDF = OBJ_TFIDF.transform(X_TEST)




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)
```

```
After vectorizations
(22445, 12837) (22445,)
(11055, 12837) (11055,)
(16500, 12837) (16500,)
====================================================================================================
```

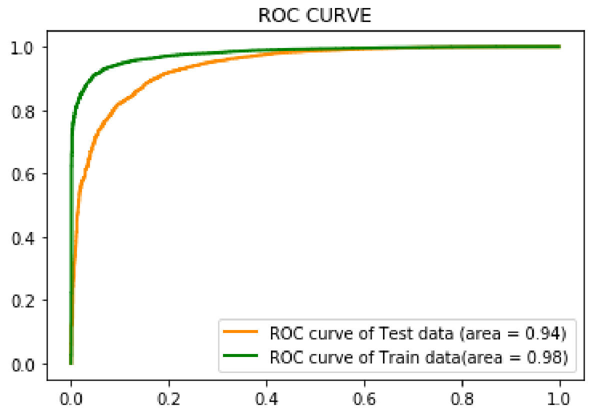
[5.2.1] Applying Logistic Regression with L1 regularization on TFIDF, SET 2

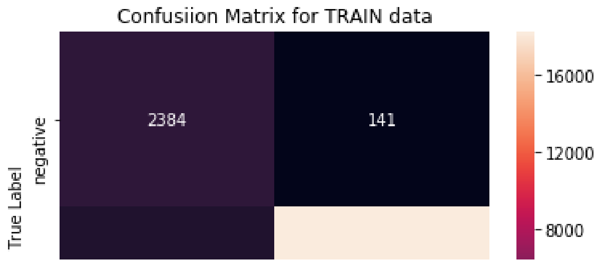
In [110]:

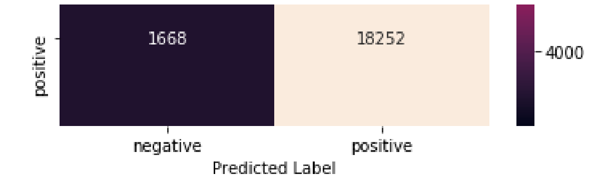
```
# Please write all the code with proper documentation
SPARSITY = LOG_REG(X_TRAIN_TFIDF,Y_TRAIN,X_CV_TFIDF,Y_CV,X_TEST_TFIDF,Y_TEST,'l1')
```

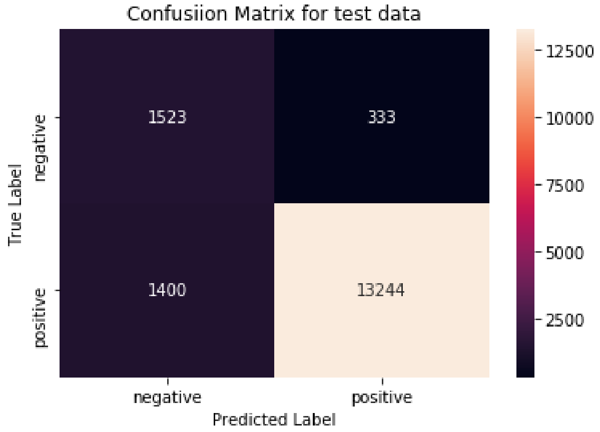



```
BEST_C is 1
```





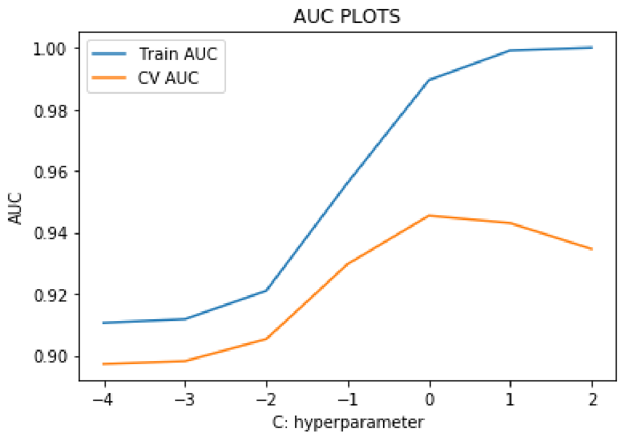




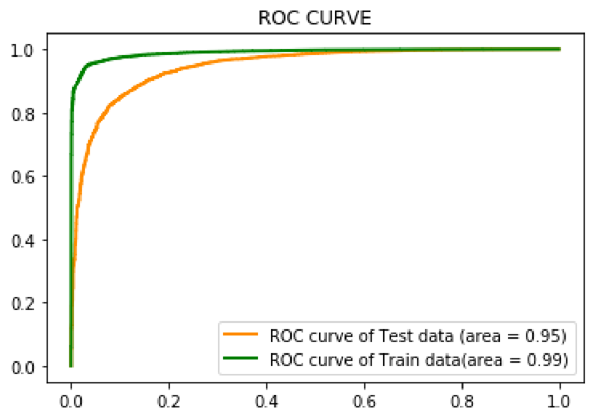
[5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

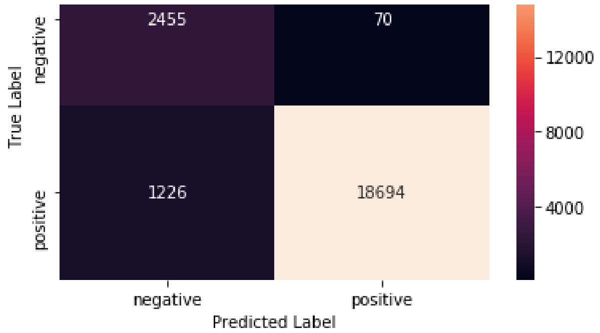
In [111]:

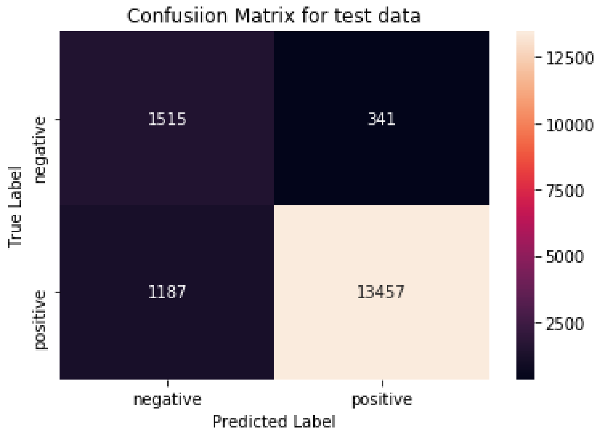
```
# Please write all the code with proper documentation
SPARSITY = LOG_REG(X_TRAIN_TFIDF,Y_TRAIN,X_CV_TFIDF,Y_CV,X_TEST_TFIDF,Y_TEST,'l2')
```




```
BEST_C is 1
```







[5.2.3] Feature Importance on TFIDF, SET 2

In [ ]:

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

from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(penalty='l2',C=1,class_weight = 'balanced')
LR.fit(X_TRAIN_TFIDF,Y_TRAIN)
WT = LR.coef_ #WEIGHTS

OBJ_TFIDF = TfidfVectorizer(ngram_range=(1,2), min_df=10)
p = OBJ_TFIDF.fit_transform(X_TRAIN)

p = pd.DataFrame(WT.T,columns=['+ve'])
p['feature'] = OBJ_TFIDF.get_feature_names()
```

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

In [ ]:

```
q = p.sort_values(by = '+ve',ascending= False)
print("Top 10  important features of positive class", np.array(q['feature'][:10]))
```

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

In [ ]:

```
# Please write all the code with proper documentation
print("Top 10  important features of negative class",np.array(q.tail(10)['feature']))
```

[5.3] Logistic Regression on AVG W2V, SET 3

In [112]:

```
# Train your own Word2Vec model using your own text corpus
i=0
```

```python
i=0
list_of_sentance=[]
for sentance in X_TRAIN:
    list_of_sentance.append(sentance.split())

 # min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)

w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
```

```
number of words that occured minimum 5 times  9211
```

In [113]:

```python
def AVGW2V(X_test):

    i=0
    list_of_sentance=[]
    for sentance in X_test:
        list_of_sentance.append(sentance.split())
    test_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 change t
his to 300 if you use google's w2v
        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
        test_vectors.append(sent_vec)
    return test_vectors
```

In [114]:

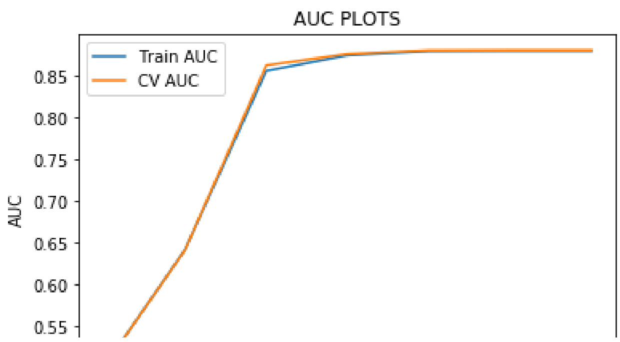
```python
AV_TRAIN = AVGW2V(X_TRAIN)
AV_CV= AVGW2V(X_CV)
AV_TEST = AVGW2V(X_TEST)
```

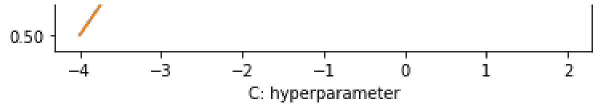
```
100%|
████████████████████████████████████████████████████████████████████████████████████████████████████
████████| 22445/22445 [03:10<00:00, 118.03it/s]
100%|
████████████████████████████████████████████████████████████████████████████████████████████████████
████████| 11055/11055 [01:37<00:00, 113.53it/s]
100%|
████████████████████████████████████████████████████████████████████████████████████████████████████
████████| 16500/16500 [02:13<00:00, 123.33it/s]
```

[5.3.1] Applying Logistic Regression with L1 regularization on AVG W2V SET 3

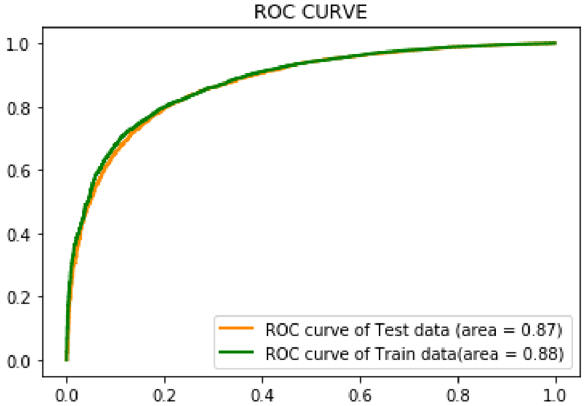
In [115]:

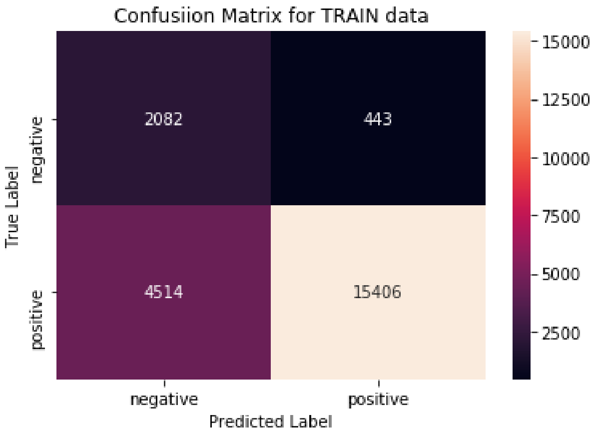
```python
# Please write all the code with proper documentation
SPARSITY = LOG_REG(AV_TRAIN,Y_TRAIN,AV_CV,Y_CV,AV_TEST,Y_TEST,'l1')
```

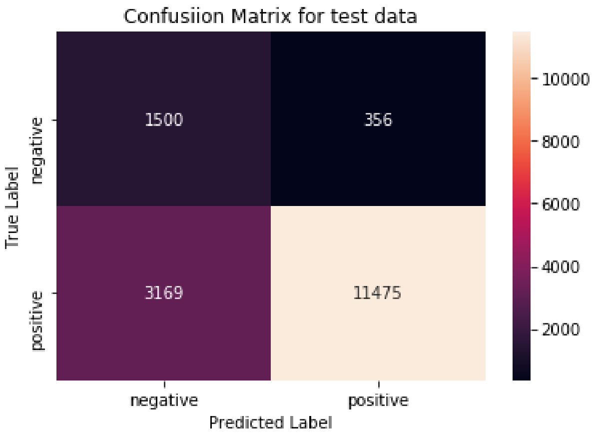





```
BEST_C is 10
```



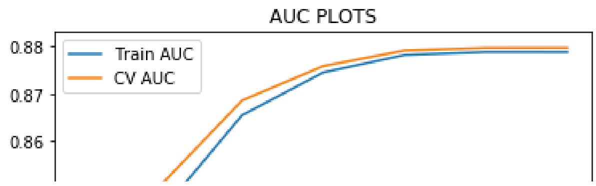


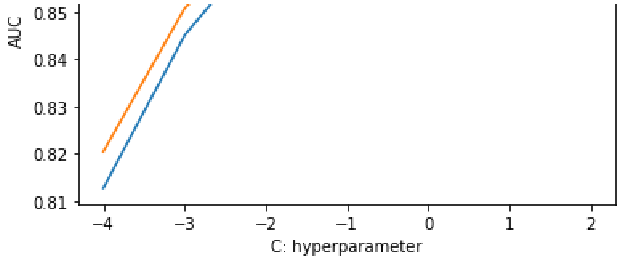


[5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

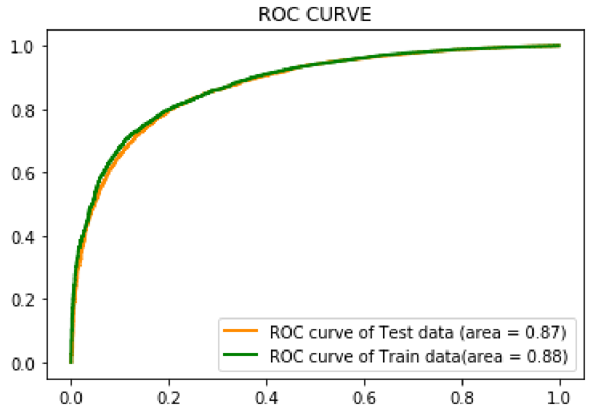
In [116]:

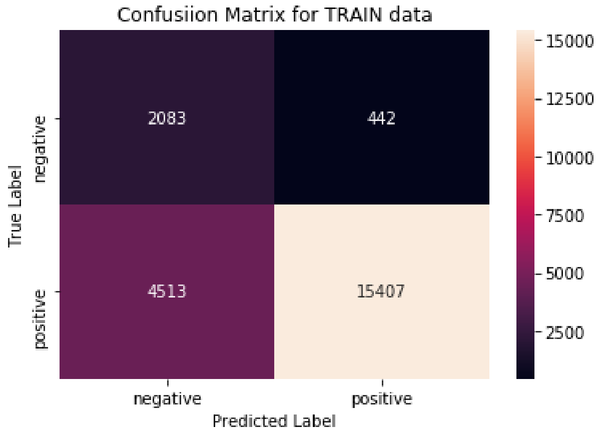
```
# Please write all the code with proper documentation
SPARSITY = LOG_REG(AV_TRAIN,Y_TRAIN,AV_CV,Y_CV,AV_TEST,Y_TEST,'l2')
```

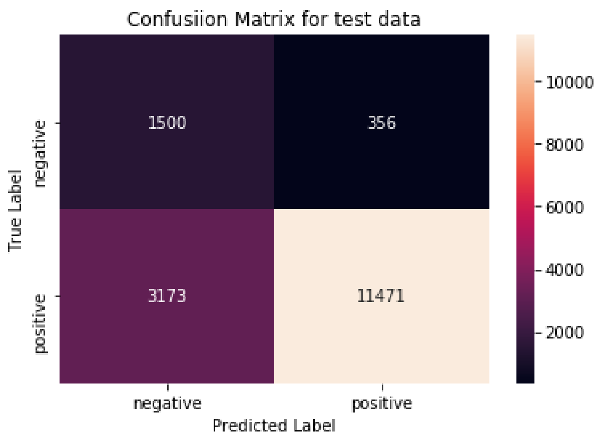





```
BEST_C is 100
```







[5.4] Logistic Regression on TFIDF W2V, SET 4

In [117]:

```
model = TfidfVectorizer()
model.fit(X_TRAIN)

dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
# 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
```

In [118]:

```
def TFIDFW2V(test):
    '''
    Returns tfidf word2vec
    '''
    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
    i=0
    list_of_sentance=[]
    for sentance in test:
        list_of_sentance.append(sentance.split())

    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 = 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)

    return tfidf_sent_vectors
```

In [119]:

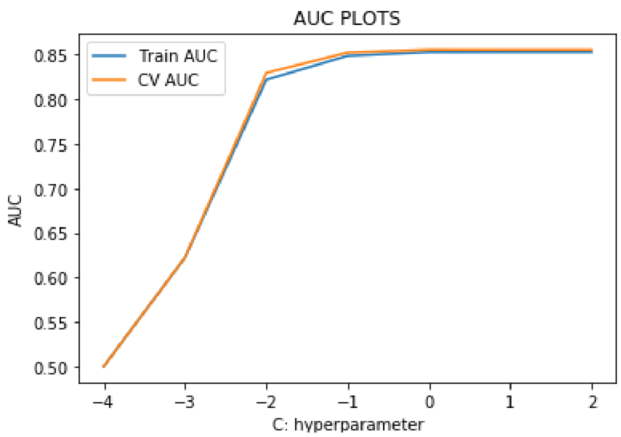
```
AV_TRAIN_TFIDF = TFIDFW2V(X_TRAIN)
AV_CV_TFIDF = TFIDFW2V(X_CV)
AV_TEST_TFIDF = TFIDFW2V(X_TEST)
```

```
100%|██████████████████████████████████████████████████| 22445/22445 [29:58<00:00, 12.63it/s]
100%|██████████████████████████████████████████████████| 11055/11055 [16:52<00:00,  7.95it/s]
100%|██████████████████████████████████████████████████| 16500/16500 [12:56<00:00, 21.26it/s]
```

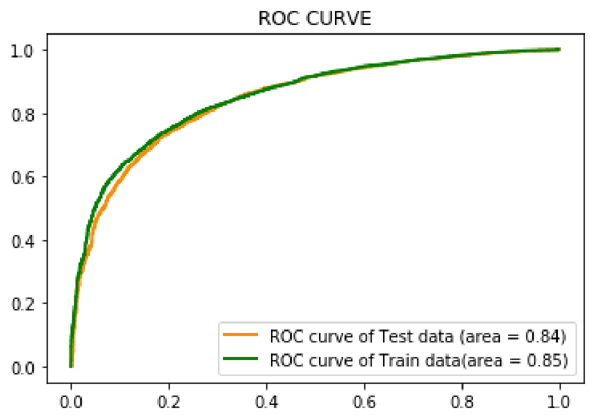
[5.4.1] Applying Logistic Regression with L1 regularization on TFIDF W2V, SET 4

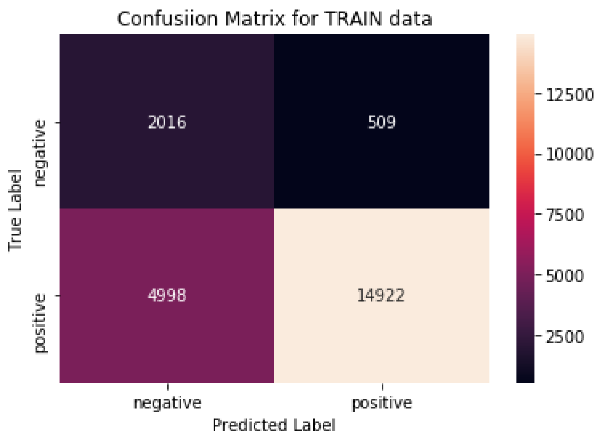
In [120]:

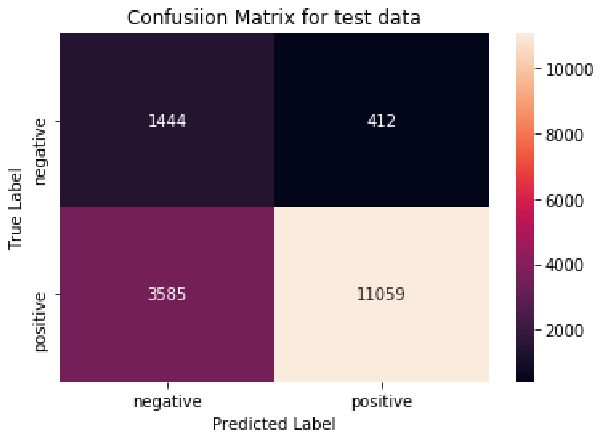
```
# Please write all the code with proper documentation
SPARSITY = LOG_REG(AV_TRAIN_TFIDF,Y_TRAIN,AV_CV_TFIDF,Y_CV,AV_TEST_TFIDF,Y_TEST,'l1')
```



BEST_C is 1



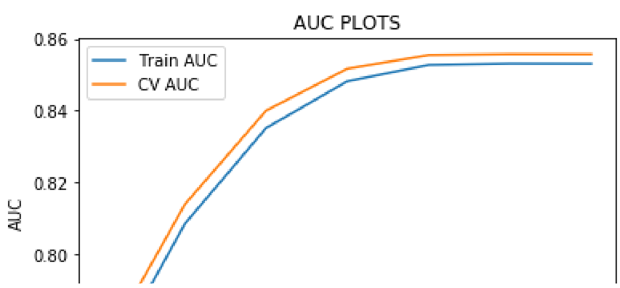


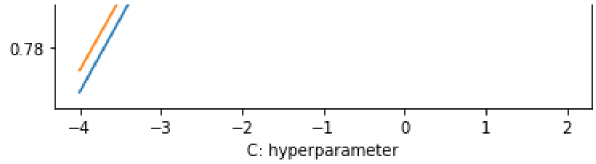


[5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

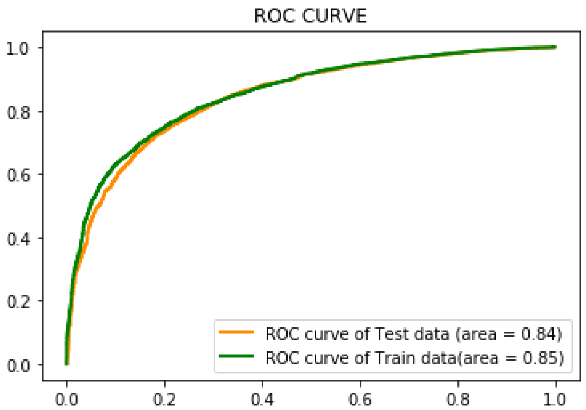
In [121]:

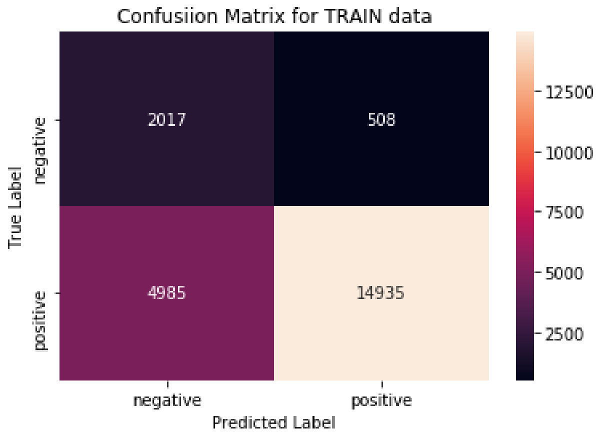
```
# Please write all the code with proper documentation
SPARSITY = LOG_REG(AV_TRAIN_TFIDF,Y_TRAIN,AV_CV_TFIDF,Y_CV,AV_TEST_TFIDF,Y_TEST,'l2')
```

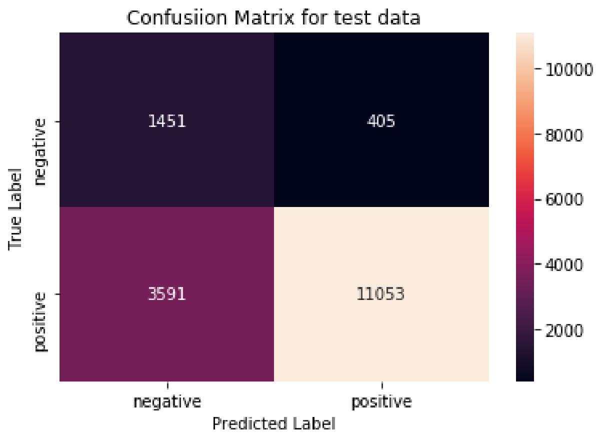





```
BEST_C is 10
```







[6] Conclusions

In [122]:

```
# Please compare all your models using Prettytable librar
from prettytable import PrettyTable
X=  PrettyTable()
X.field_names=['METHOD','Regularization','HyperParameter','Test AUC']
X.add_row(['BOW','L1',0.1,0.92])
X.add_row(['BOW','L2',0.1,0.93])
X.add_row(['TFIDF','L1',1,0.94])
X.add_row(['TFIDF','L2',1,0.95])
```

```
X.add_row(['AVGW2V','L1',10,0.87])
X.add_row(['AVGW2V','L2',100,0.87])
X.add_row(['TFIDFW2v','L1',1,0.84])
X.add_row(['TFIDFW2V','L2',10,0.84])
print(X)
```

```
+----------+----------------+----------------+----------+
|  METHOD  | Regularization | HyperParameter | Test AUC |
+----------+----------------+----------------+----------+
|   BOW    |       L1       |      0.1       |   0.92   |
|   BOW    |       L2       |      0.1       |   0.93   |
|  TFIDF   |       L1       |       1        |   0.94   |
|  TFIDF   |       L2       |       1        |   0.95   |
|  AVGW2V  |       L1       |       10       |   0.87   |
|  AVGW2V  |       L2       |      100       |   0.87   |
| TFIDFW2v |       L1       |       1        |   0.84   |
| TFIDFW2V |       L2       |       10       |   0.84   |
+----------+----------------+----------------+----------+
```

Refrence:https://github.com/omkar1610/Amazon-Fine-Food-Reviews