# 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 | Tim |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | B001E4KFG0 | A3SGXH7AUHU8GW | delmartian | 1 | 1 | 1 | 130386240 |
| **1** | 2 | B00813GRG4 | A1D87F6ZCVE5NK | dll pa | 0 | 0 | 0 | 134697600 |

|   | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | HelpfulnessDenominator | Score | Tim |
|---|---|---|---|---|---|---|---|---|
| **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]:

```
'\ndisplay= 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

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]:

```python
# 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)
'''
```

Out[14]:

```
'\nsent_0 = final[\'Text\'].values[0]\nprint(sent_0)\nprint("="*50)\n\nsent_1000 =
final[\'Text\'].values[1000]\nprint(sent_1000)\nprint("="*50)\n\nsent_1500 =
final[\'Text\'].values[1500]\nprint(sent_1500)\nprint("="*50)\n\nsent_4900 =
final[\'Text\'].values[4900]\nprint(sent_4900)\nprint("="*50)\n'
```

In [15]:

```python
# 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)
'''
```

Out[15]:

```
'\nsent_0 = re.sub(r"http\\S+", "", sent_0)\nsent_1000 = re.sub(r"http\\S+", "",
sent_1000)\nsent_150 = re.sub(r"http\\S+", "", sent_1500)\nsent_4900 = re.sub(r"http\\S+", "", sen
t_4900)\n\nprint(sent_0)\n'
```

In [16]:

```python
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-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)
'''
```

Out[16]:

```
'\nfrom bs4 import BeautifulSoup\n\nsoup = BeautifulSoup(sent_0, \'lxml\')\ntext =
soup.get_text()\nprint(text)\nprint("="*50)\n\nsoup = BeautifulSoup(sent_1000, \'lxml\')\ntext = s
oup.get_text()\nprint(text)\nprint("="*50)\n\nsoup = BeautifulSoup(sent_1500, \'lxml\')\ntext = so
up.get_text()\nprint(text)\nprint("="*50)\n\nsoup = BeautifulSoup(sent_4900, \'lxml\')\ntext = sou
p.get_text()\nprint(text)\n'
```

In [17]:

```python
# 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]:

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

In [19]:

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

In [20]:

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

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', '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"])
```

In [22]:

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

In [23]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
from bs4 import BeautifulSoup
# 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 [11:39<00:00, 520.77it/s]
```

In [24]:

```
SORT_DATA['Score'].value_counts()
```

Out[24]:

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

In [25]:

```
#preprocessed_reviews[1500]
```

In [26]:

```
DATA = np.array(preprocessed_reviews[0:60000])
LABEL  = np.array(SORT_DATA['Score'][0:60000])
```

In [27]:

```
DATA_2 = np.array(preprocessed_reviews[0:30000])
LABEL_2  = np.array(SORT_DATA['Score'][0:30000])
```

In [28]:

```
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=
LABEL)
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 [29]:

```
from sklearn.model_selection import train_test_split
X_train_temp, X_TEST_RBF, Y_train_temp, Y_TEST_RBF = train_test_split(DATA_2, LABEL_2, test_size=0.
33,stratify=LABEL_2)
X_TRAIN_RBF, X_CV_RBF, Y_TRAIN_RBF, Y_CV_RBF = train_test_split(X_train_temp, Y_train_temp, test_si
ze=0.33,stratify=Y_train_temp)
```

## [3.2] Preprocessing Review Summary

In [30]:

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

## [4] Featurization

### [4.1] BAG OF WORDS

In [31]:

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

Out[31]:

```
'\n#BoW\ncount_vect = CountVectorizer() #in scikit-
learn\ncount_vect.fit(preprocessed_reviews)\nprint("some feature names ",
count_vect.get_feature_names()[:10])\nprint(\'=\'*50)\n\nfinal_counts =
count_vect.transform(preprocessed_reviews)\nprint("the type of count vectorizer
count_vect.transform(preprocessed_reviews)\nprint("the type of count vectorizer
",type(final_counts))\nprint("the shape of out text BOW vectorizer
",final_counts.get_shape())\nprint("the number of unique words ", final_counts.get_shape()[1])\n'
```

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

In [32]:

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

Out[32]:

```
'\n#bi-gram, tri-gram and n-gram\n\n#removing stop words like "not" should be avoided before build
ing n-grams\n# count_vect = CountVectorizer(ngram_range=(1,2))\n# please do read the
CountVectorizer documentation http://scikit-
learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html\n\n# you c
an choose these numebrs min_df=10, max_features=5000, of your choice\ncount_vect =
CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)\nfinal_bigram_counts =
count_vect.fit_transform(preprocessed_reviews)\nprint("the type of count vectorizer
",type(final_bigram_counts))\nprint("the shape of out text BOW vectorizer
",final_bigram_counts.get_shape())\nprint("the number of unique words including both unigrams and
bigrams ", final_bigram_counts.get_shape()[1])\n'
```

## [4.3] TF-IDF

In [33]:

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

Out[33]:

```
'\ntf_idf_vect = TfidfVectorizer(ngram_range=(1,2),
min_df=10)\ntf_idf_vect.fit(preprocessed_reviews)\nprint("some sample features(unique words in the
corpus)",tf_idf_vect.get_feature_names()[0:10])\nprint(\'=\'*50)\n\nfinal_tf_idf =
tf_idf_vect.transform(preprocessed_reviews)\nprint("the type of count vectorizer
",type(final_tf_idf))\nprint("the shape of out text TFIDF vectorizer
",final_tf_idf.get_shape())\nprint("the number of unique words including both unigrams and bigrams
", final_tf_idf.get_shape()[1])\n'
```

## [4.4] Word2Vec

In [34]:

```
# 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 [35]:

```
'''
# 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 ")
'''
```

Out[35]:

```
'\n# Using Google News Word2Vectors\n\n# in this project we are using a pretrained model by
google\n# its 3.3G file, once you load this into your memory \n# it occupies ~9Gb, so please do th
is step only if you have >12G of ram\n# we will provide a pickle file wich contains a dict , \n# a
nd it contains all our courpus words as keys and  model[word] as values\n# To use this code-snippe
t, download "GoogleNews-vectors-negative300.bin" \n# from
https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit\n# it\'s 1.9GB in size.\n\n\n# h
ttp://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY\n# you can comment th
is whole cell\n# or change these varible according to your
need\n\nis_your_ram_gt_16g=False\nwant_to_use_google_w2v = False\nwant_to_train_w2v = True\n\nif w
ant_to_train_w2v:\n    # min_count = 5 considers only words that occured atleast 5 times\n    w2v_m
odel=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)\n
print(w2v_model.wv.most_similar(\'great\'))\n    print(\'=\'*50)\n
print(w2v_model.wv.most_similar(\'worst\'))\n    \nelif want_to_use_google_w2v and
is_your_ram_gt_16g:\n    if os.path.isfile(\'GoogleNews-vectors-negative300.bin\'):\n        w2v_mo
del=KeyedVectors.load_word2vec_format(\'GoogleNews-vectors-negative300.bin\', binary=True)\n
print(w2v_model.wv.most_similar(\'great\'))\n        print(w2v_model.wv.most_similar(\'worst\'))\n
else:\n        print("you don\'t have gogole\'s word2vec file, keep want_to_train_w2v = True, to tr
ain your own w2v ")\n'
```

In [36]:

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

Out[36]:

```
'\nw2v_words = list(w2v_model.wv.vocab)\nprint("number of words that occured minimum 5 times
",len(w2v_words))\nprint("sample words ", w2v_words[0:50])\n'
```

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

#### [4.4.1.1] Avg W2v

In [37]:

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

Out[37]:

```
"\n# average Word2Vec\n# compute average word2vec for each review.\nsent_vectors = []; # the avg-w
2v for each sentence/review is stored in this list\nfor sent in tqdm(list_of_sentance): # for each
review/sentence\n    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might ne
ed to change this to 300 if you use google's w2v\n    cnt_words =0; # num of words with a valid ve
ctor in the sentence/review\n    for word in sent: # for each word in a review/sentence\n        if
word in w2v_words:\n            vec = w2v_model.wv[word]\n            sent_vec += vec\n
nt_words += 1\n    if cnt_words != 0:\n        sent_vec /= cnt_words\n
sent_vectors.append(sent_vec)\nprint(len(sent_vectors))\nprint(len(sent_vectors[0]))\n"
```

#### [4.4.1.2] TFIDF weighted W2v

In [38]:

```
'''
# 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_)))
'''
```

Out[38]:

```
'\n# S = ["abc def pqr", "def def def abc", "pqr pqr def"]\nmodel =
TfidfVectorizer()\ntf_idf_matrix = model.fit_transform(preprocessed_reviews)\n# we are converting
a dictionary with word as a key, and the idf as a value\ndictionary =
dict(zip(model.get_feature_names(), list(model.idf_)))\n'
```

In [39]:

```
'''
# 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
'''
```

Out[39]:

```
'\n# TF-IDF weighted Word2Vec\ntfidf_feat = model.get_feature_names() # tfidf words/col-names\n# f
inal_tf_idf is the sparse matrix with row= sentence, col=word and cell_val =
tfidf\n\ntfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list\
nrow=0;\nfor sent in tqdm(list_of_sentance): # for each review/sentence \n    sent_vec =
np.zeros(50) # as word vectors are of zero length\n    weight_sum =0; # num of words with a valid
vector in the sentence/review\n    for word in sent: # for each word in a review/sentence\n
if word in w2v_words and word in tfidf_feat:\n            vec = w2v_model.wv[word]\n#            t
f_idf = tf_idf_matrix[row, tfidf_feat.index(word)]\n            # to reduce the computation we are
\n            # dictionary[word] = idf value of word in whole courpus\n            #
sent.count(word) = tf valeus of word in this review\n            tf_idf = dictionary[word]*(sent.cc
unt(word)/len(sent))\n            sent_vec += (vec * tf_idf)\n            weight_sum += tf_idf\n
if weight_sum != 0:\n        sent_vec /= weight_sum\n    tfidf_sent_vectors.append(sent_vec)\n    r
ow += 1\n'
```

## [5] Assignment 7: SVM

1. **Apply SVM 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. **Procedure**
    - You need to work with 2 versions of SVM
        - Linear kernel
        - RBF kernel
    - When you are working with linear kernel, use SGDClassifier' with hinge loss because it is computationally less expensive.
    - When you are working with 'SGDClassifier' with hinge loss and trying to find the AUC score, you would have to use CalibratedClassifierCV
    - Similarly, like kdtree of knn, when you are working with RBF kernel it's better to reduce the number of dimensions. You can put min_df = 10, max_features = 500 and consider a sample size of 40k points.

3. **Hyper paramter tuning (find best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**
    - 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

4. **Feature importance**
    - When you are working on the linear kernel with BOW or TFIDF please print the top 10 best features for each of the positive and negative classes.

5. **Feature engineering**

5. **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.

6. **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.

7. **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 SVM

## [5.1] Linear SVM

In [40]:

```
#---------------------LINEAR _SVM FUNCTION------------------------
```

In [41]:

```
def LINEAR_SVM(X_train,Y_TRAIN,X_cv,Y_CV,X_test,Y_TEST,PENALTY,VAR,count_vect):
    from sklearn.preprocessing import StandardScaler
    from sklearn.linear_model import SGDClassifier
    from sklearn.calibration import CalibratedClassifierCV
    from sklearn.metrics import roc_auc_score
    import matplotlib.pyplot as plt
    import pandas as pd
    import numpy as np

    scalar = StandardScaler(with_mean=False)
    X_TRAIN = scalar.fit_transform(X_train)
    X_TEST= scalar.transform(X_test)
    X_CV=scalar.transform(X_cv)

    ALPHA = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]      #alpha=1/C
    AUC_TRAIN = []
    AUC_CV =[]

    for a in ALPHA:
        MODEL=SGDClassifier(alpha=a,penalty=PENALTY)    #loss default hinge
        SVM=CalibratedClassifierCV(MODEL,cv=3) #calibrated classifier cv for calculation of predic_
prob
        SVM.fit(X_TRAIN,Y_TRAIN)
        PROB_CV=SVM.predict_proba(X_CV)[:,1]
        AUC_CV.append(roc_auc_score(Y_CV,PROB_CV))
        PROB_TRAIN=SVM.predict_proba(X_TRAIN)[:,1]
        AUC_TRAIN.append(roc_auc_score(Y_TRAIN,PROB_TRAIN))

    plt.figure()
    plt.plot(ALPHA,AUC_TRAIN, label='Train AUC')
    plt.plot(ALPHA,AUC_CV, label='CV AUC')
    plt.legend()
    plt.xlabel("K: hyperparameter")
    plt.ylabel("AUC")
    plt.title("AUC PLOTS")
    plt.show()

    BEST_ALPHA = ALPHA[AUC_CV.index(max(AUC_CV))]
    print("BEST_ALPHA is {}".format(BEST_ALPHA))

    model=SGDClassifier(alpha=BEST_ALPHA,penalty=PENALTY)
    svm=CalibratedClassifierCV(model, cv=3)
    svm.fit(X_TRAIN,Y_TRAIN)


    TRAIN_PROBA=  list(svm.predict_proba(X_TRAIN)[:,1])
    TEST_PROBA =  list(svm.predict_proba(X_TEST)[:,1])

    from sklearn import metrics
    fpr_2,tpr_2,tr_2 = metrics.roc_curve(Y_TEST,TEST_PROBA)
    fpr_1,tpr_1,tr_1 = metrics.roc_curve(Y_TRAIN,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")

    PRED_TEST=list(svm.predict(X_TEST))
    PRED_TEST = np.array(PRED_TEST)


    PRED_TRAIN=list(svm.predict(X_TRAIN))
    PRED_TRAIN = np.array(PRED_TRAIN)

    from sklearn.metrics import confusion_matrix
    import seaborn as sns

    plt.figure()
    cm = confusion_matrix(Y_TEST,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()

    plt.figure()
    cm = confusion_matrix(Y_TRAIN,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()

    if VAR == 'YES':
        #top 10 positive features
        FEATURES = count_vect.get_feature_names()
        model=SGDClassifier(alpha=BEST_ALPHA,penalty = PENALTY)
        model.fit(X_TRAIN,Y_TRAIN)
        weight=model.coef_
        pos_indx=np.argsort(weight)[:,::-1]
        neg_indx=np.argsort(weight)
        print('Top 10 positive features :')
        for i in list(pos_indx[0][0:10]):
            print(FEATURES[i])
        print("="*100)
        print('Top 10 negative features :')
        for i in list(neg_indx[0][0:10]):
            print(FEATURES[i])
```

## BOW

In [42]:

```
#.....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)
```

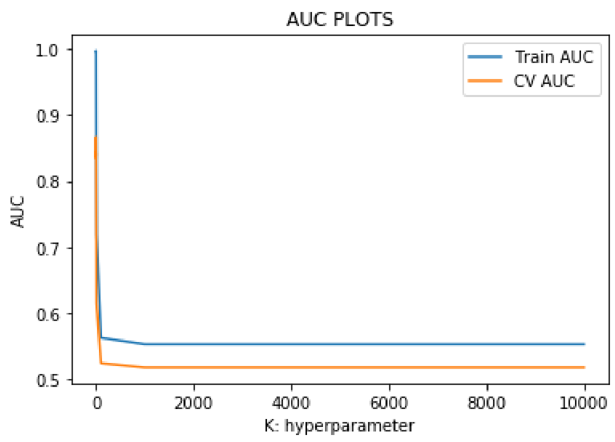
```
After vectorizations
(26934, 31491) (26934,)
(13266, 31491) (13266,)
(19800, 31491) (19800,)
===================================================================================================
```

### [5.1.1] Applying Linear SVM on BOW, SET 1

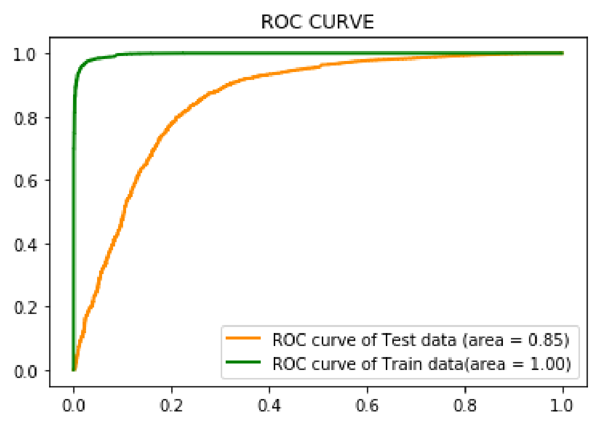
In [43]:

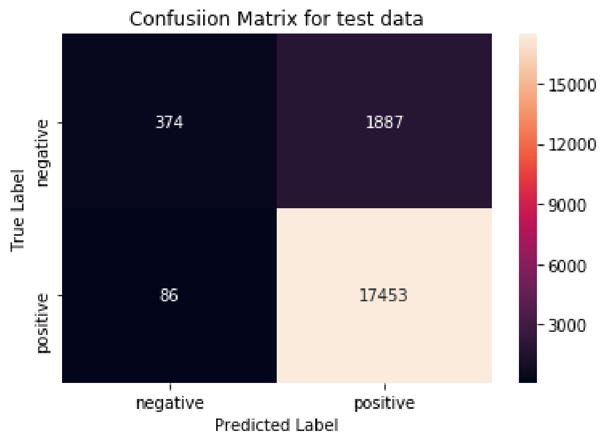
```
# Please write all the code with proper documentation
LINEAR_SVM(X_TRAIN_BOW,Y_TRAIN,X_CV_BOW,Y_CV,X_TEST_BOW,Y_TEST,"l2",'YES',OBJ_BOW)
```

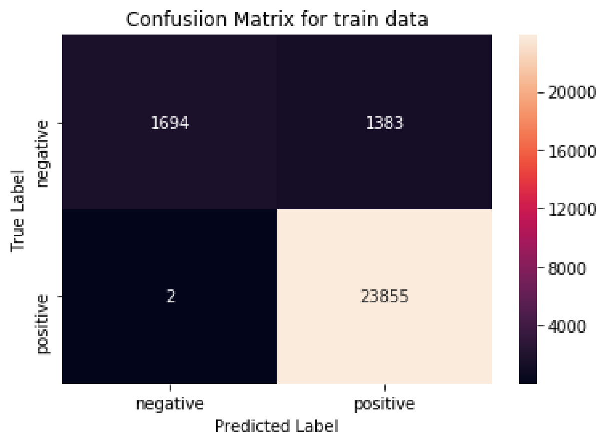
```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```



```
BEST_ALPHA is 0.1
```







```
Top 10 positive features :
great
good
best
love
delicious
excellent
favorite
loves
tasty
perfect
==============================================================================================

Top 10 negative features :
not
worst
awful
disappointed
terrible
waste
```

```
thought
tasteless
maybe
disappointment
```

## TFIDF

In [46]:

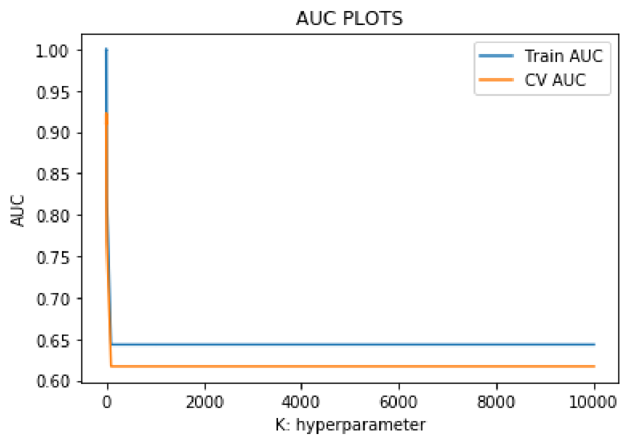
```
from sklearn.feature_extraction.text import TfidfVectorizer
VECTORIZER_TF_IDF = TfidfVectorizer(ngram_range=(1,2), min_df=10)
VECTORIZER_TF_IDF.fit(X_TRAIN)


X_TRAIN_TFIDF = VECTORIZER_TF_IDF.transform(X_TRAIN)
X_CV_TFIDF = VECTORIZER_TF_IDF.transform(X_CV)
X_TEST_TFIDF = VECTORIZER_TF_IDF.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
(26934, 15117) (26934,)
(13266, 15117) (13266,)
(19800, 15117) (19800,)
====================================================================================================
```

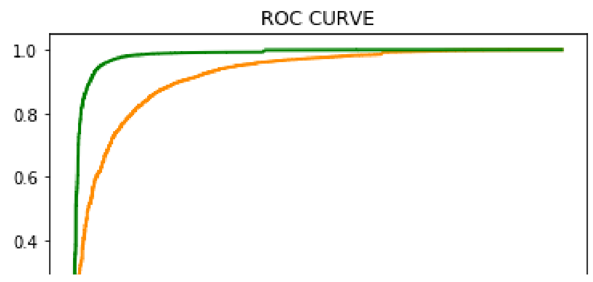
### [5.1.2] Applying Linear SVM on TFIDF, SET 2

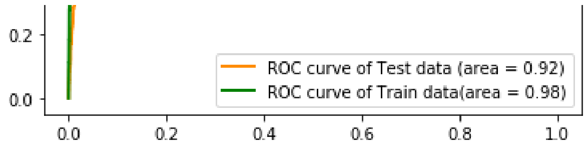
In [48]:

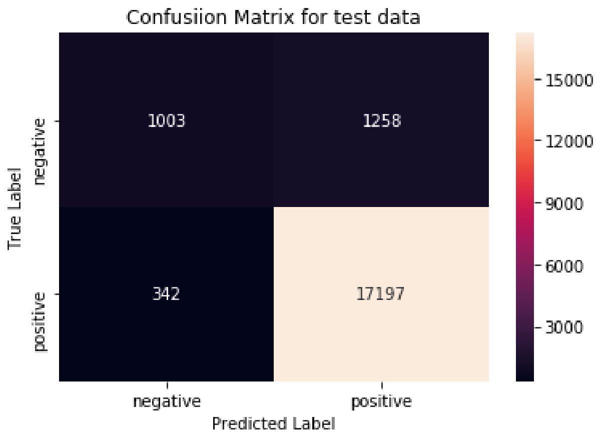
```
# Please write all the code with proper documentation
LINEAR_SVM(X_TRAIN_TFIDF,Y_TRAIN,X_CV_TFIDF,Y_CV,X_TEST_TFIDF,Y_TEST,"l2",'YES',VECTORIZER_TF_IDF)
```

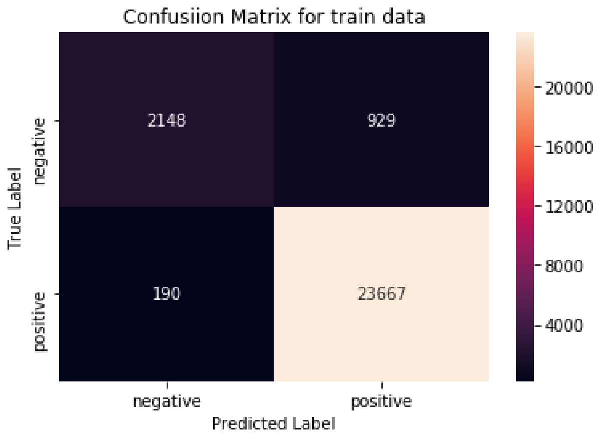


```
BEST_ALPHA is 1
```









```
Top 10 positive features :
great
good
best
love
delicious
excellent
loves
favorite
wonderful
perfect
====================================================================================================

Top 10 negative features :
not worth
worst
disappointed
terrible
threw
awful
not recommend
not buy
disappointment
horrible
```

## AVGW2V

In [49]:

```
# Train your own Word2Vec model using your own text corpus
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  9998
```

In [50]:

```
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 [51]:

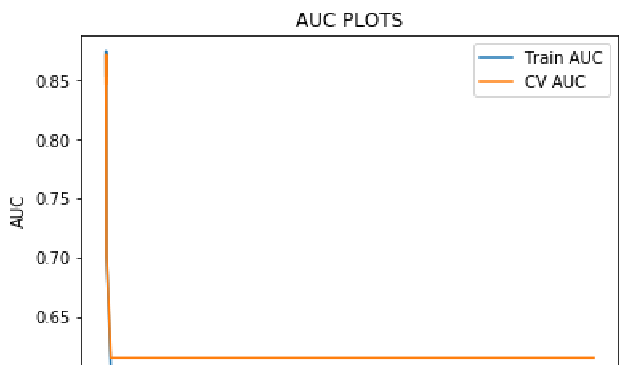
```
AV_TRAIN_BOW = AVGW2V(X_TRAIN)
AV_CV_BOW = AVGW2V(X_CV)
AV_TEST_BOW = AVGW2V(X_TEST)
```

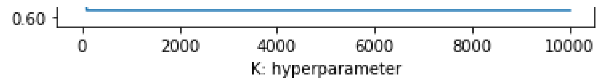
```
100%|
███████████████████████████████████████████████████████████████████████████████████████████████████
██████████| 26934/26934 [02:56<00:00, 152.41it/s]
100%|
███████████████████████████████████████████████████████████████████████████████████████████████████
██████████| 13266/13266 [01:28<00:00, 149.77it/s]
100%|
███████████████████████████████████████████████████████████████████████████████████████████████████
██████████| 19800/19800 [02:11<00:00, 150.69it/s]
```

## [5.1.3] Applying Linear SVM on AVG W2V, SET 3

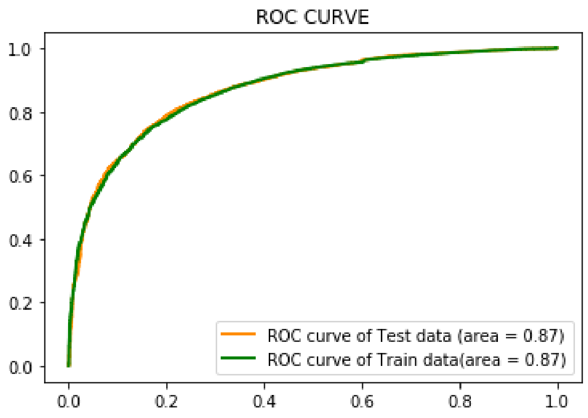
In [52]:

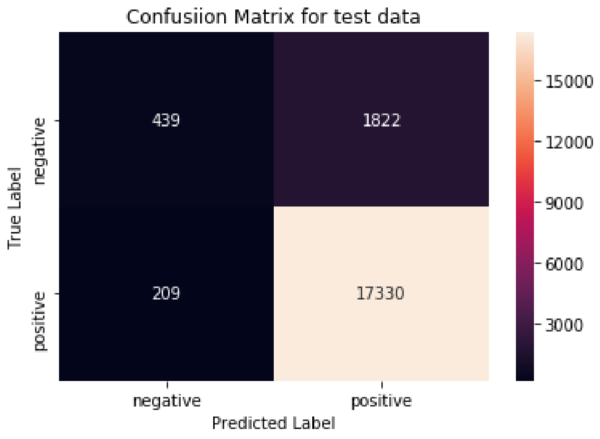
```
# Please write all the code with proper documentation
LINEAR_SVM(AV_TRAIN_BOW,Y_TRAIN,AV_CV_BOW,Y_CV,AV_TEST_BOW,Y_TEST,"l2",'NOPE',VECTORIZER_TF_IDF)
```

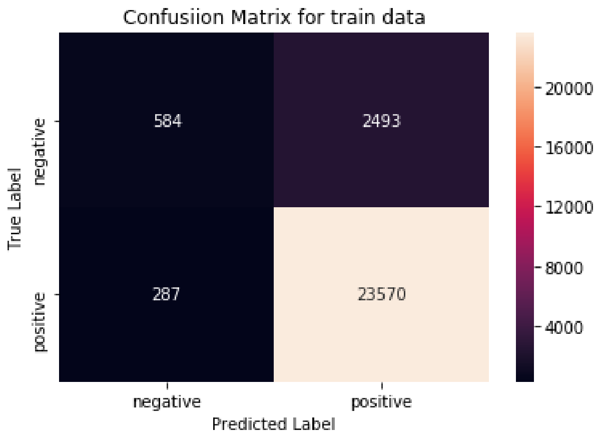




```
BEST_ALPHA is 0.001
```







## TFIDF W2V

In [53]:

```
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 [54]:

In [54]:

```python
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


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

```
100%|
██████████████████████████████████████████████████████████████████████████████████████
███████| 26934/26934 [25:17<00:00, 23.54it/s]
100%|
██████████████████████████████████████████████████████████████████████████████████████
███████| 13266/13266 [12:18<00:00, 17.18it/s]
100%|
██████████████████████████████████████████████████████████████████████████████████████
███████| 19800/19800 [18:26<00:00, 19.86it/s]
```

In [55]:

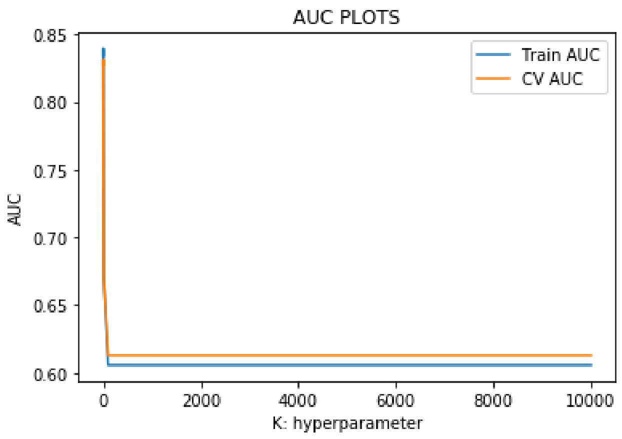
```python
AV_TRAIN_TFIDF = TFIDFW2V(X_TRAIN)
AV_CV_TFIDF = TFIDFW2V(X_CV)
AV_TEST_TFIDF = TFIDFW2V(X_TEST)
```

```
100%|
██████████████████████████████████████████████████████████████████████████████████████
███████| 26934/26934 [25:10<00:00, 17.84it/s]
100%|
██████████████████████████████████████████████████████████████████████████████████████
███████| 13266/13266 [12:20<00:00, 18.10it/s]
100%|
██████████████████████████████████████████████████████████████████████████████████████
███████| 19800/19800 [18:26<00:00, 17.89it/s]
```

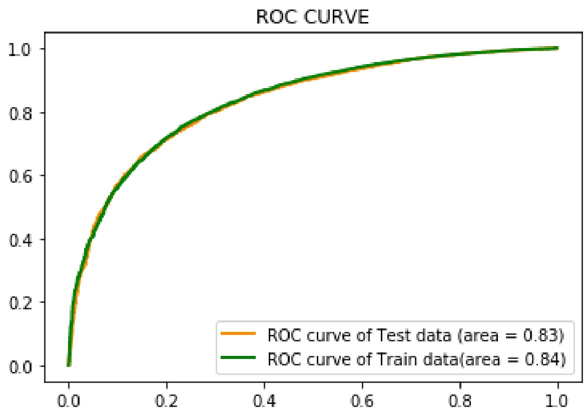
### [5.1.4] Applying Linear SVM on TFIDF W2V, SET 4

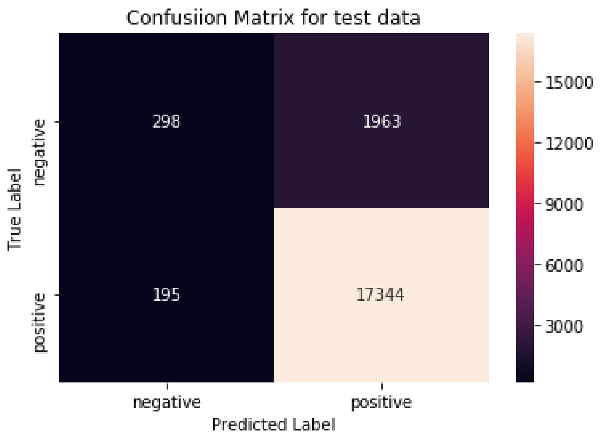
In [56]:

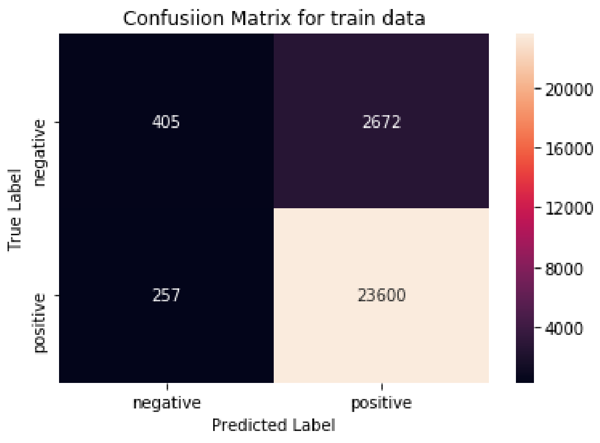
```python
# Please write all the code with proper documentation
LINEAR_SVM(AV_TRAIN_TFIDF,Y_TRAIN,AV_CV_TFIDF,Y_CV,AV_TEST_TFIDF,Y_TEST,"l2",'NOPE',VECTORIZER_TF_I
DF)
```



```
BEST_ALPHA is 0.01
```







## [5.2] RBF SVM

In [57]:

```
def RBF_SVM(X_train,Y_TRAIN,X_cv,Y_CV,X_test,Y_TEST):
    from sklearn.preprocessing import StandardScaler
    from sklearn.svm import SVC
    from sklearn.calibration import CalibratedClassifierCV
    from sklearn.metrics import roc_auc_score
    import matplotlib.pyplot as plt
    import pandas as pd
    import numpy as np

    scalar = StandardScaler(with_mean=False)
    X_TRAIN = scalar.fit_transform(X_train)
    X_TEST= scalar.transform(X_test)
    X_CV=scalar.transform(X_cv)

    ALPHA = [10**-4, 10**-3,10**-2,10**-1,1,10,10**2,10**3,10**4]     #alpha=1/C
    AUC_TRAIN = []
    AUC_CV =[]

    for a in ALPHA:
        SVM=SVC(C=a,probability=True)   #loss default hinge
        #SVM=CalibratedClassifierCV(MODEL,cv=3) #calibrated classifier cv for calculation of
predic_prob
        SVM.fit(X_TRAIN,Y_TRAIN)
        PROB_CV=SVM.predict_proba(X_CV)[:,1]
        AUC_CV.append(roc_auc_score(Y_CV,PROB_CV))
        PROB_TRAIN=SVM.predict_proba(X_TRAIN)[:,1]
        AUC_TRAIN.append(roc_auc_score(Y_TRAIN,PROB_TRAIN))

    plt.figure()
    plt.plot(ALPHA,AUC_TRAIN, label='Train AUC')
    plt.plot(ALPHA,AUC_CV, label='CV AUC')
    plt.legend()
    plt.xlabel("K: hyperparameter")
    plt.ylabel("AUC")
    plt.title("AUC PLOTS")
    plt.show()

    BEST_ALPHA = ALPHA[AUC_CV.index(max(AUC_CV))]
    print("BEST_ALPHA is {}".format(BEST_ALPHA))

    svm=SVC(C=BEST_ALPHA,probability=True)
    #svm=CalibratedClassifierCV(model, cv=3)
    svm.fit(X_TRAIN,Y_TRAIN)


    TRAIN_PROBA=  list(svm.predict_proba(X_TRAIN)[:,1])
    TEST_PROBA =  list(svm.predict_proba(X_TEST)[:,1])

    from sklearn import metrics
    fpr_2,tpr_2,tr_2 = metrics.roc_curve(Y_TEST,TEST_PROBA)
    fpr_1,tpr_1,tr_1 = metrics.roc_curve(Y_TRAIN,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")

    PRED_TEST=list(svm.predict(X_TEST))
    PRED_TEST = np.array(PRED_TEST)


    PRED_TRAIN=list(svm.predict(X_TRAIN))
    PRED_TRAIN = np.array(PRED_TRAIN)

    from sklearn.metrics import confusion_matrix
    import seaborn as sns

    plt.figure()
    cm = confusion_matrix(Y_TEST,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()

    plt.figure()
    cm = confusion_matrix(Y_TRAIN,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()
```

# BOW

In [58]:

```
#.....CONVERT it into BOW VECTORS....
from sklearn.feature_extraction.text import CountVectorizer
OBJ_BOW = CountVectorizer(min_df=10, max_features=500)
OBJ_BOW.fit(X_TRAIN_RBF)





X_TRAIN_BOW =OBJ_BOW.transform(X_TRAIN_RBF)
X_CV_BOW = OBJ_BOW.transform(X_CV_RBF)
X_TEST_BOW = OBJ_BOW.transform(X_TEST_RBF)





print("After vectorizations")
print(X_TRAIN_BOW.shape, Y_TRAIN_RBF.shape)
print(X_CV_BOW.shape,Y_CV_RBF.shape)
print(X_TEST_BOW.shape, Y_TEST_RBF.shape)
print("="*100)
```

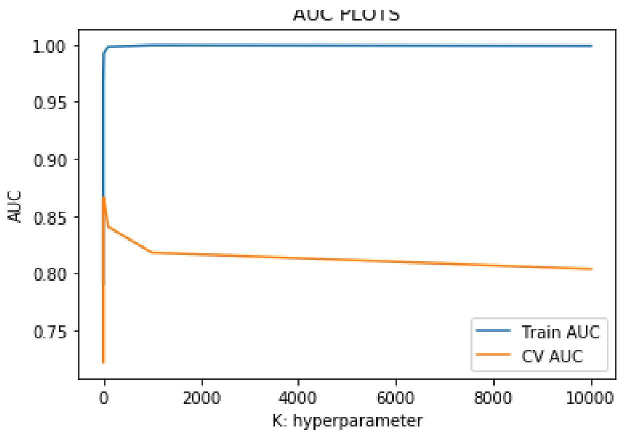
```
After vectorizations
(13467, 500) (13467,)
(6633, 500) (6633,)
(9900, 500) (9900,)
====================================================================================================
```

### [5.2.1] Applying RBF SVM on BOW, SET 1

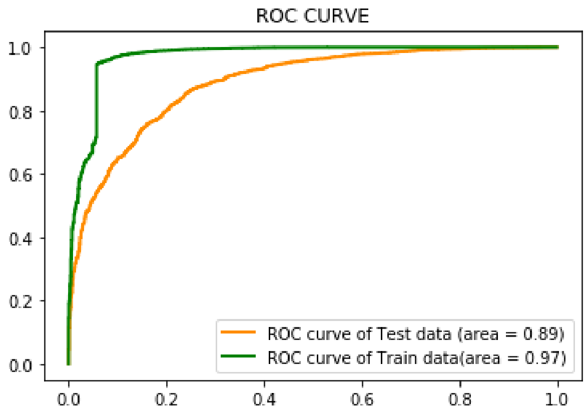
In [59]:

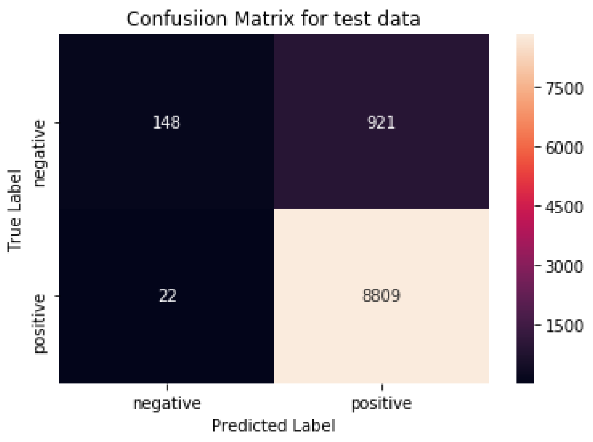
```
# Please write all the code with proper documentation
RBF_SVM(X_TRAIN_BOW,Y_TRAIN_RBF,X_CV_BOW,Y_CV_RBF,X_TEST_BOW,Y_TEST_RBF)
```

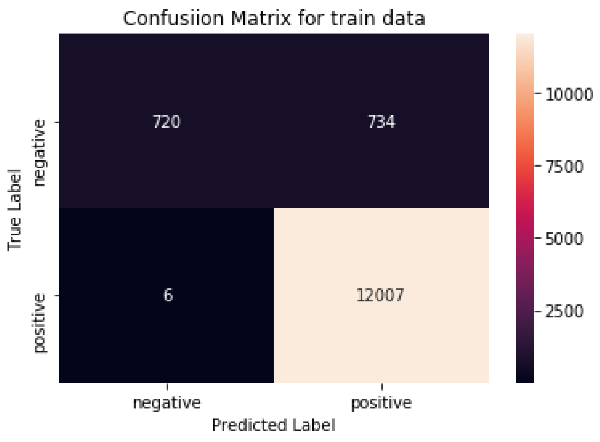
```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:
Data with input dtype int64 was converted to float64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```



```
BEST_ALPHA is 1
```







## TFIDF

In [60]:

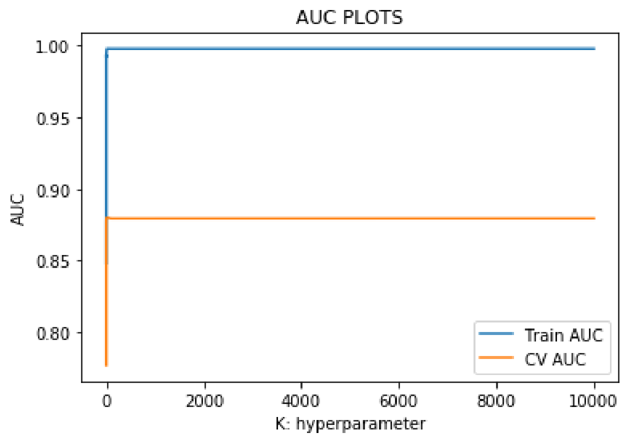
```
from sklearn.feature_extraction.text import TfidfVectorizer
VECTORIZER_TF_IDF = TfidfVectorizer(ngram_range=(1,2), min_df=10,max_features=500)
VECTORIZER_TF_IDF.fit(X_TRAIN_RBF)


X_TRAIN_TFIDF = VECTORIZER_TF_IDF.transform(X_TRAIN_RBF)
X_CV_TFIDF = VECTORIZER_TF_IDF.transform(X_CV_RBF)
X_TEST_TFIDF = VECTORIZER_TF_IDF.transform(X_TEST_RBF)



print("After vectorizations")
print(X_TRAIN_TFIDF.shape, Y_TRAIN_RBF.shape)
print(X_CV_TFIDF.shape,Y_CV_RBF.shape)
print(X_TEST_TFIDF.shape, Y_TEST_RBF.shape)
print("="*100)
```

```
After vectorizations
(13467, 500) (13467,)
(6633, 500) (6633,)
(9900, 500) (9900,)
====================================================================================================
```

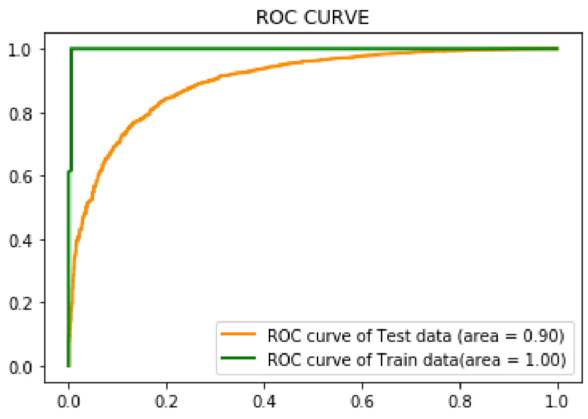
### [5.2.2] Applying RBF SVM on TFIDF, SET 2

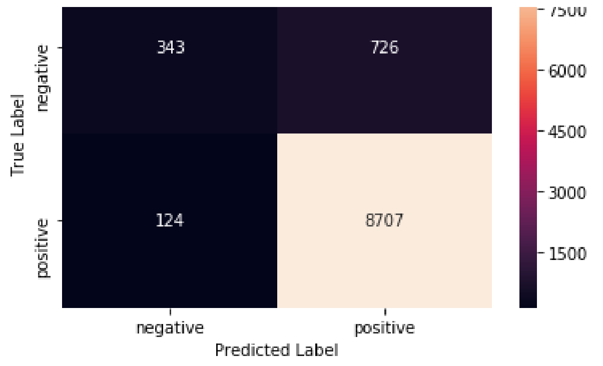
In [61]:

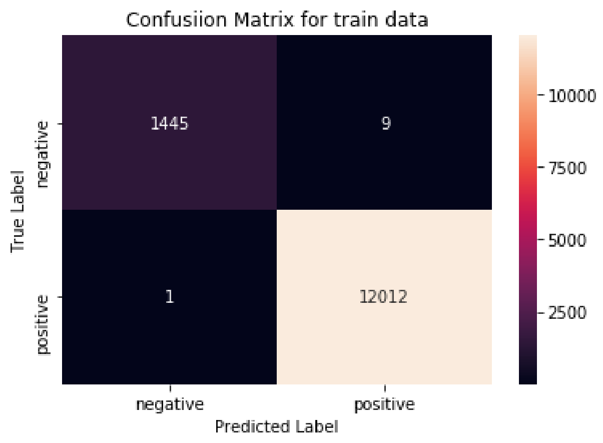
```
# Please write all the code with proper documentation
RBF_SVM(X_TRAIN_TFIDF,Y_TRAIN_RBF,X_CV_TFIDF,Y_CV_RBF,X_TEST_TFIDF,Y_TEST_RBF)
```



```
BEST_ALPHA is 10
```







## AVG W2V

In [62]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in X_TRAIN_RBF:
    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  7001
```

In [63]:

```
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 [64]:

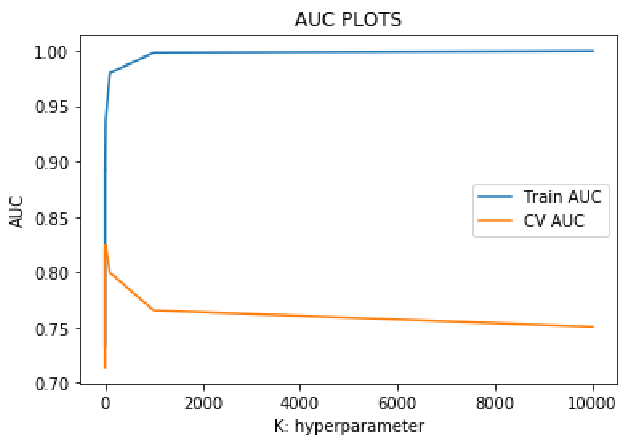
```
AV_TRAIN_BOW = AVGW2V(X_TRAIN_RBF)
AV_CV_BOW = AVGW2V(X_CV_RBF)
AV_TEST_BOW = AVGW2V(X_TEST_RBF)
```

```
100%|
████████████████████████████████████████████████████████████████
██████| 13467/13467 [01:04<00:00, 209.84it/s]
100%|
████████████████████████████████████████████████████████████████
████████| 6633/6633 [00:32<00:00, 204.67it/s]
100%|
████████████████████████████████████████████████████████████████
████████| 9900/9900 [00:56<00:00, 175.47it/s]
```

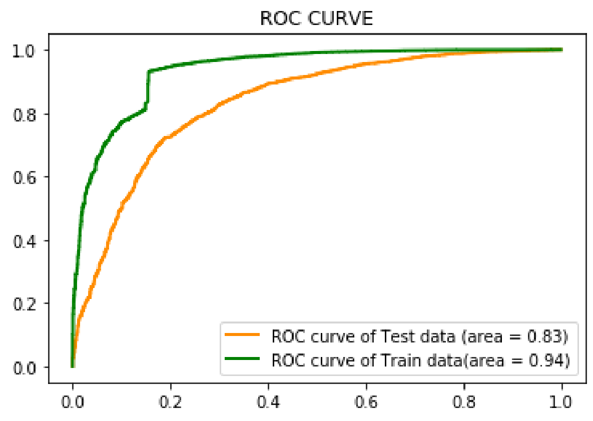
### [5.2.3] Applying RBF SVM on AVG W2V, SET 3

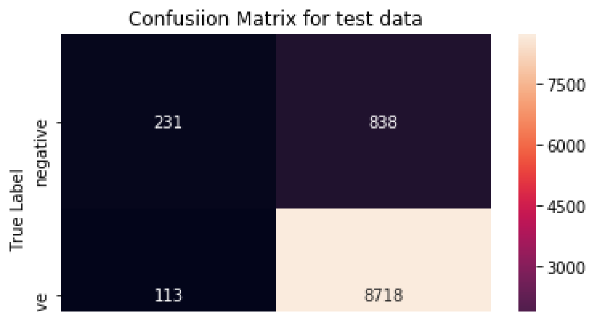
In [66]:

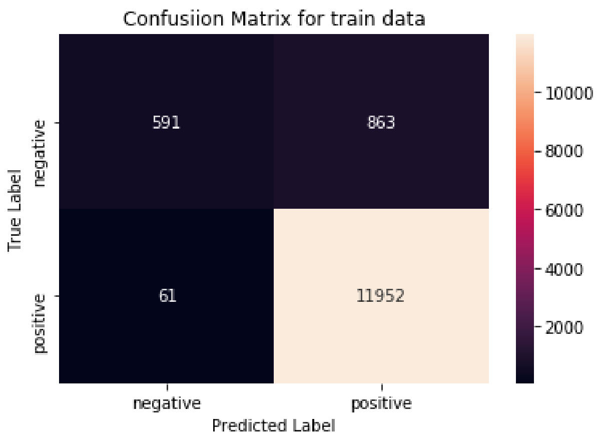
```
# Please write all the code with proper documentation
RBF_SVM(AV_TRAIN_BOW,Y_TRAIN_RBF,AV_CV_BOW,Y_CV_RBF,AV_TEST_BOW,Y_TEST_RBF)
```



```
BEST_ALPHA is 10
```







## TFIDF W2V

In [67]:

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

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 [68]:

```
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 [69]:

```
AV_TRAIN_TFIDF = TFIDFW2V(X_TRAIN_RBF)
AV_CV_TFIDF = TFIDFW2V(X_CV_RBF)
AV_TEST_TFIDF = TFIDFW2V(X_TEST_RBF)
```

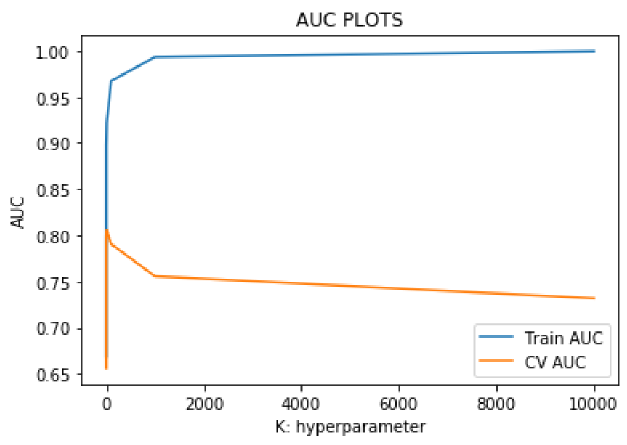
```
100%|██████████████████████████████████████████████████████████████
```

```
███████████| 13467/13467 [13:16<00:00, 16.90it/s]
100%|
████████████████████████████████████████████████████████████████████████████████████████
█████████████| 6633/6633 [05:27<00:00, 20.28it/s]
100%|
████████████████████████████████████████████████████████████████████████████████████████
█████████████| 9900/9900 [08:05<00:00, 17.04it/s]
```

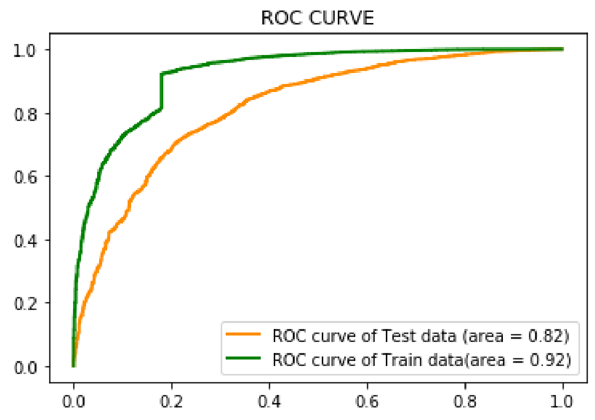
### [5.2.4] Applying RBF SVM on TFIDF W2V, SET 4

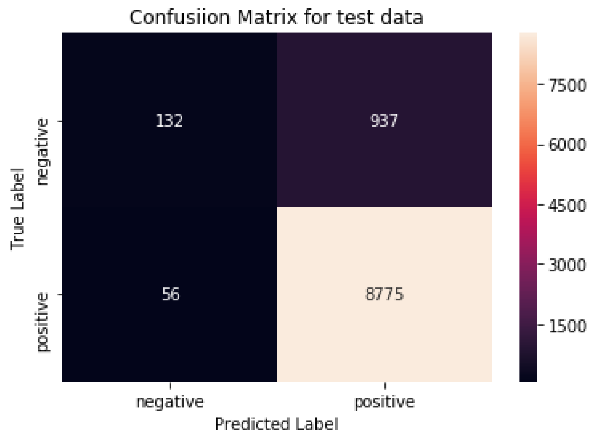
In [70]:

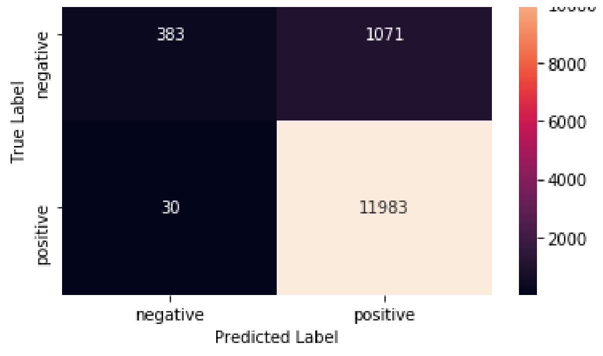
```python
# Please write all the code with proper documentation
RBF_SVM(AV_TRAIN_TFIDF,Y_TRAIN_RBF,AV_CV_TFIDF,Y_CV_RBF,AV_TEST_TFIDF,Y_TEST_RBF)
```



```
BEST_ALPHA is 10
```







## [6] Conclusions

In [72]:

```
# Please compare all your models using Prettytable library

from prettytable import PrettyTable
X = PrettyTable()
print(" "*40+"CONCLUSION")
print("="*100)


X.field_names = ["METHOD","VECTORIZER", "BEST_ALPHA", "TEST_AUC"]
X.add_row(["LINEAR","BOW",0.1,0.85])
X.add_row(["LINEAR","TFIDF",1,0.92])
X.add_row(["LINEAR","AVGW2V",0.001,0.87])
X.add_row(["LINEAR","TFIDFW2V",0.01,0.83])

X.add_row(["RBF","BOW",1,0.89])
X.add_row(["RBF","TFIDF",10,0.90])
X.add_row(["RBF","AVGW2V",10,0.83])
X.add_row(["RBF","TFIDFW2V",10,0.82])
print(X)
```

```
                                        CONCLUSION
====================================================================================================

+--------+------------+------------+----------+
| METHOD | VECTORIZER | BEST_ALPHA | TEST_AUC |
+--------+------------+------------+----------+
| LINEAR |    BOW     |    0.1     |   0.85   |
| LINEAR |   TFIDF    |     1      |   0.92   |
| LINEAR |   AVGW2V   |   0.001    |   0.87   |
| LINEAR |  TFIDFW2V  |    0.01    |   0.83   |
|  RBF   |    BOW     |     1      |   0.89   |
|  RBF   |   TFIDF    |     10     |   0.9    |
|  RBF   |   AVGW2V   |     10     |   0.83   |
|  RBF   |  TFIDFW2V  |     10     |   0.82   |
+--------+------------+------------+----------+
```

## I have consider 60K points for linear SVM and 30K for RBF