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
```

```
C:\ProgramData\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWa
rning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_seria
l")
```

In [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect('C:/Users/Excel/Desktop/vins/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 50
0000 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 Sco
re != 3 LIMIT 500000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score
 != 3 LIMIT 100000""", con)

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

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU

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

In [5]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text	(

	UserId	ProductId	ProfileName	Time	Score	Text
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...

In [6]:

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

Out[6]: 393063

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [7]:

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln

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

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and 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='quicksort', na_position='last')
```

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

Out[9]: (87775, 10)

In [10]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]: 87.775

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:

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

In [12]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [13]:
```
#Before starting the next phase of preprocessing lets see the number of
 entries left
print(final.shape)
```

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

(87773, 10)

Out[13]:
```
1    73592
0    14181
Name: Score, dtype: int64
```

In [14]:
```
final["Time"] = pd.to_datetime(final["Time"], unit = "s")
final = final.sort_values(by = "Time")
```

[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 [15]:
```
# printing some random reviews
```

```python
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)
```

```
I bought a few of these after my apartment was infested with fruit flie
s. After only a few hours, the trap had &quot;attracted&quot; many flie
s and within a few days they were practically gone. This may not be a l
ong term  solution, but if flies are driving you crazy, consider buying
this. One  caution- the surface is very sticky, so try to avoid touchin
g it.
==================================================
I have made these brownies for family and for a den of cub scouts and n
o one would have known they were gluten free and everyone asked for sec
onds!  These brownies have a fudgy texture and have bits of chocolate c
hips in them which are delicious.  I would say the mix is very thick an
d a little difficult to work with.  The cooked brownies are slightly di
fficult to cut into very neat edges as the edges tend to crumble a litt
le and I would also say that they make a slightly thinner layer of brow
nies than most of the store brand gluten containing but they taste just
as good, if not better.  Highly recommended!<br /><br />(For those wond
ering, this mix requires 2 eggs OR 4 egg whites and 7 tbs melted butter
to prepare.  They do have suggestions for lactose free and low fat prep
arations)
==================================================
This gum is my absolute favorite. By purchasing on amazon I can get the
savings of large quanities at a very good price. I highly recommend to
all gum chewers. Plus as you enjoy the peppermint flavor and freshing o
f breath you are whitening your teeth all at the same time.
```

==
This is an excellent product, both tastey and priced right. It's difficult to find this product in regular local grocery stores, so I was thrilled to find it.
==

In [16]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

I bought a few of these after my apartment was infested with fruit flies. After only a few hours, the trap had "attracted" many flies and within a few days they were practically gone. This may not be a long term solution, but if flies are driving you crazy, consider buying this. One caution- the surface is very sticky, so try to avoid touching it.

In [17]:
```
# 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)
```

```
I bought a few of these after my apartment was infested with fruit flie
s. After only a few hours, the trap had "attracted" many flies and with
in a few days they were practically gone. This may not be a long term
solution, but if flies are driving you crazy, consider buying this. One
  caution- the surface is very sticky, so try to avoid touching it.
==================================================
I have made these brownies for family and for a den of cub scouts and n
o one would have known they were gluten free and everyone asked for sec
onds!  These brownies have a fudgy texture and have bits of chocolate c
hips in them which are delicious.  I would say the mix is very thick an
d a little difficult to work with.  The cooked brownies are slightly di
fficult to cut into very neat edges as the edges tend to crumble a litt
le and I would also say that they make a slightly thinner layer of brow
nies than most of the store brand gluten containing but they taste just
as good, if not better.  Highly recommended!(For those wondering, this
mix requires 2 eggs OR 4 egg whites and 7 tbs melted butter to prepare.
  They do have suggestions for lactose free and low fat preparations)
==================================================
This gum is my absolute favorite. By purchasing on amazon I can get the
savings of large quanities at a very good price. I highly recommend to
all gum chewers. Plus as you enjoy the peppermint flavor and freshing o
f breath you are whitening your teeth all at the same time.
==================================================
This is an excellent product, both tastey and priced right. It's diffic
ult to find this product in regular local grocery stores, so I was thri
lled to find it.
```

In [18]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
```

```
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [19]:

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

```
This gum is my absolute favorite. By purchasing on amazon I can get the
savings of large quanities at a very good price. I highly recommend to
all gum chewers. Plus as you enjoy the peppermint flavor and freshing o
f breath you are whitening your teeth all at the same time.
==================================================
```

In [20]:

```
#remove words with numbers python: https://stackoverflow.com/a/1808237
0/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
I bought a few of these after my apartment was infested with fruit flie
s. After only a few hours, the trap had &quot;attracted&quot; many flie
s and within a few days they were practically gone. This may not be a l
ong term  solution, but if flies are driving you crazy, consider buying
this. One  caution- the surface is very sticky, so try to avoid touchin
g it.
```

In [21]:

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

```
This gum is my absolute favorite By purchasing on amazon I can get the
savings of large quanities at a very good price I highly recommend to a
ll gum chewers Plus as you enjoy the peppermint flavor and freshing of
breath you are whitening your teeth all at the same time
```

In [22]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'no
t'
# <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', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', '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', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n'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 [23]:
```python
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower
() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████████████████████████████████| 87773/87773 [00:38<00:00, 230
5.13it/s]
```

In [24]:
```python
preprocessed_reviews[1500]
```

Out[24]:
```
'gum absolute favorite purchasing amazon get savings large quanities go
od price highly recommend gum chewers plus enjoy peppermint flavor fres
hing breath whitening teeth time'
```

[3.2] Preprocessing Review Summary

In [25]:
```python
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS (LINEAR KERNEL)

In [26]:
```
X=preprocessed_reviews
Y=final["Score"]
```

In [27]:
```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3,shuffle=False) # this is time based splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33,shuffle=False)
```

In [28]:
```
# we are converting the into one hot encoding
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,max_features=10000, ngram_range=
(1,2))
vectorizer.fit(X_train) # fit has to happen only on train data

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

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

```
After BOW VEC
(39400, 10000) (39400,)
(19407, 10000) (19407,)
(28966, 10000) (28966,)
```

standardising the data

In [29]:
```
from sklearn.preprocessing import StandardScaler
```

```
standardised=StandardScaler(with_mean=False)
X_train_bow=standardised.fit_transform(X_train_bow)
X_cv_bow=standardised.transform(X_cv_bow)
X_test_bow=standardised.transform(X_test_bow)
```

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

In [30]:

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
```

In [31]:

```
def support(train,cv):
    alphas=[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]
    penalt=["l1","l2"]
    parameter={"alpha":alphas,"penalty":penalt}

    svm=GridSearchCV(SGDClassifier(),parameter,verbose=1,scoring="roc_a
uc")
    svm.fit(train,y_train)

    alpha_opt = svm.best_params_.get('alpha')
    penalty_opt=svm.best_params_.get('penalty')
    print("best optimized alpha:" ,alpha_opt)
```

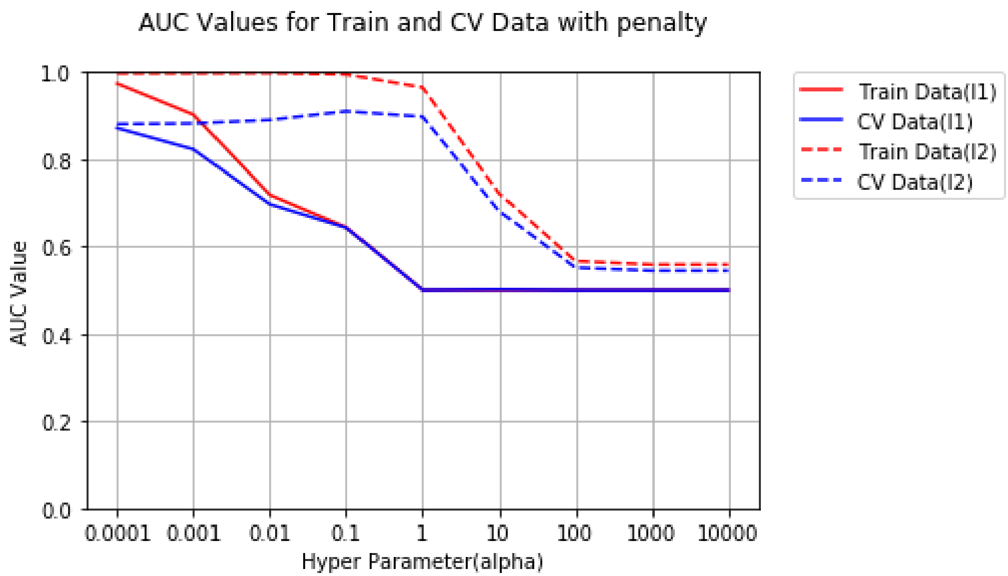
```
    print("best optimized regularization:",penalty_opt)
    train_score = svm.cv_results_.get('mean_train_score')
    test_score = svm.cv_results_.get('mean_test_score')


    plt.plot(np.log10(alphas),train_score[::2],'r', label = 'Train Data
(l1)')
    plt.plot(np.log10(alphas),test_score[::2],'b', label = 'CV Data(l
1)')
    plt.plot(np.log10(alphas),train_score[1::2],'r--', label = 'Train D
ata(l2)')
    plt.plot(np.log10(alphas),test_score[1::2],'b--', label = 'CV Data
(l2)')
    plt.xticks(np.log10(alphas), alphas)
    plt.ylim(0,1)
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespa
d=0.)
    plt.grid(True)
    plt.title("AUC Values for Train and CV Data with penalty\n")
    plt.xlabel("Hyper Parameter(alpha)")
    plt.ylabel("AUC Value")
    plt.show()
```

In [32]:
```
def confusion_matrix(train,test):
    from sklearn.metrics import confusion_matrix
    from sklearn.metrics import confusion_matrix
    Y_test_pred=SGD.predict(test)
    Y_train_pred=SGD.predict(train)
    cm_train=confusion_matrix(y_train,Y_train_pred)
    cm_test=confusion_matrix(y_test,Y_test_pred)
    print(cm_train)
    print(cm_test)
    print("*"*100)
    print("confusion matrix for test data")
    import seaborn as sns
    class_label=["0","1"]
    df_cm=pd.DataFrame(cm_test,index=class_label,columns=class_label)
    sns.heatmap(df_cm,annot=True,fmt="d")
    plt.title("confusion matrix")
```

```
    plt.xlabel("predicted label")
    plt.ylabel("true label")
    plt.show()
```

In [33]:
```
support(X_train_bow,X_cv_bow)
```

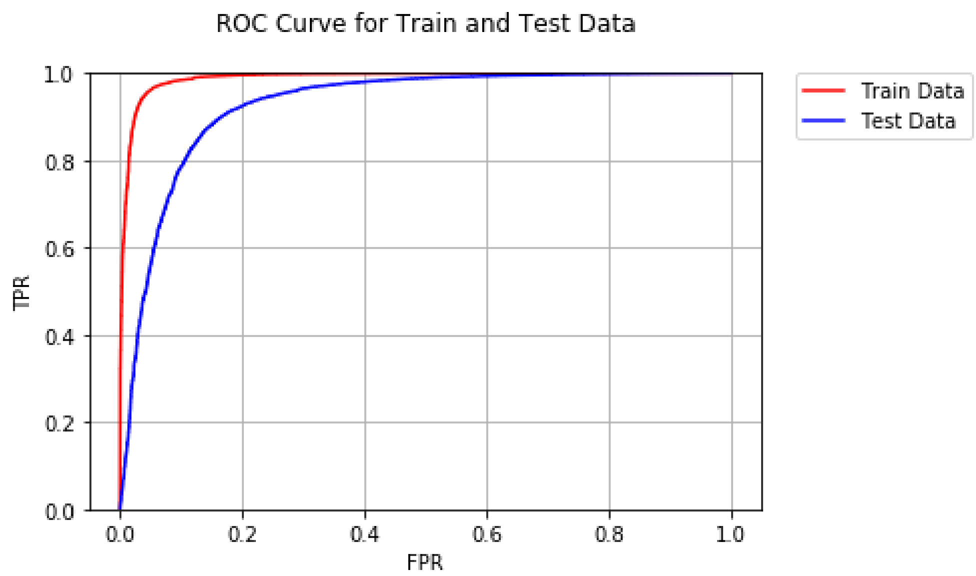
```
Fitting 3 folds for each of 18 candidates, totalling 54 fits
[Parallel(n_jobs=1)]: Done  54 out of  54 | elapsed:    5.4s finished
best optimized alpha: 0.1
best optimized regularization: l2
```



In [34]:
```
SGD=SGDClassifier(penalty="l2",alpha=0.1)
from sklearn.metrics import roc_auc_score
SGD.fit(X_train_bow,y_train)

y_train_predict_proba=SGD.decision_function(X_train_bow)
y_test_predict_proba=SGD.decision_function(X_test_bow)
fpr,tpr,threshold=roc_curve(y_train,y_train_predict_proba[:])
fpr1,tpr1,threshold1=roc_curve(y_test,y_test_predict_proba[:])
```

```
print("The AUC value for test data is ",roc_auc_score( y_test, y_test_p
redict_proba))

plt.plot(fpr,tpr,'r', label = 'Train Data')
plt.plot(fpr1,tpr1,'b', label = 'Test Data')
plt.ylim(0,1)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.
)
plt.grid(True)
plt.title("ROC Curve for Train and Test Data\n")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```

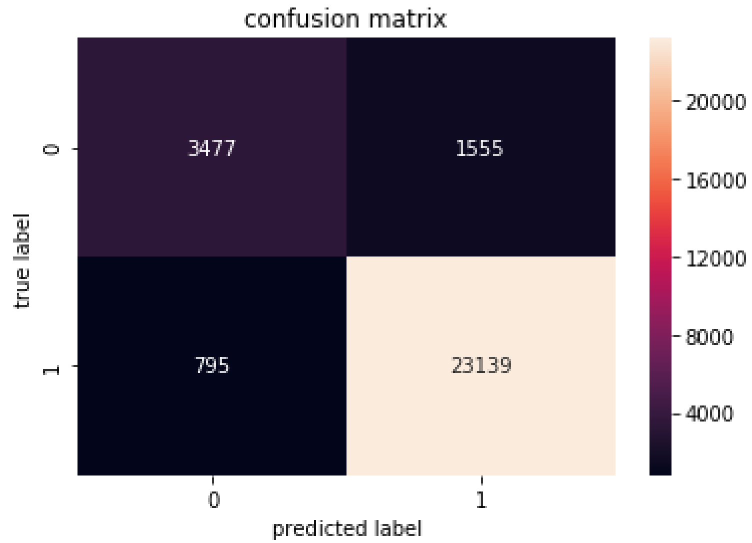
```
The AUC value for test data is  0.926261514341971
```



CONFUSION MATRIX

In [35]:
```
confusion_matrix(X_train_bow,X_test_bow)
```

```
[[ 4957   781]
 [  276 33386]]
[[ 3477  1555]
 [  795 23139]]
************************************************************************
*****************************
confusion matrix for test data
```



top 10 feature of both positive and negative

In [36]:
```
SGD = SGDClassifier(penalty="l2",alpha=0.1)
SGD.fit(X_train_bow,y_train)
feat_log = SGD.coef_

count_vect = CountVectorizer()
s = vectorizer.fit_transform(X_train)
s = pd.DataFrame(feat_log.T,columns=['-ve'])
s['feature'] = vectorizer.get_feature_names()
```

In [37]:
```
v = s.sort_values(by = '-ve',kind = 'quicksort',ascending= False)
```

```
print("Top 10  important features of positive class", np.array(v['featu
re'][:10]))
print("*"*100)
print("Top 10  important features of negative class",np.array(v.tail(10
)['feature']))
```

```
Top 10  important features of positive class ['great' 'good' 'love' 'be
st' 'delicious' 'loves' 'excellent' 'perfect'
 'tasty' 'favorite']
************************************************************************
****************************
Top 10  important features of negative class ['disappointing' 'not reco
mmend' 'not good' 'two stars' 'not buy'
 'terrible' 'awful' 'not worth' 'disappointed' 'worst']
```

TF-IDF (LINEAR KERNEL)

In [38]:

```
X=preprocessed_reviews
Y=final["Score"]
```

In [39]:

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.33,shuff
le=False)
X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.
33,shuffle=False)
```

In [40]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_TF = TfidfVectorizer(min_df=10,max_features=10000,ngram_rang
e=(1,2))
vectorizer_TF.fit(X_train) # fit has to happen only on train data

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

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

```
After TFIDF VEC
(39400, 10000) (39400,)
(19407, 10000) (19407,)
(28966, 10000) (28966,)
```

standardising the data

In [41]:

```
from sklearn.preprocessing import StandardScaler
standardised=StandardScaler(with_mean=False)
X_train_tf=standardised.fit_transform(X_train_tf)
X_cv_tf=standardised.transform(X_cv_tf)
X_test_tf=standardised.transform(X_test_tf)
```

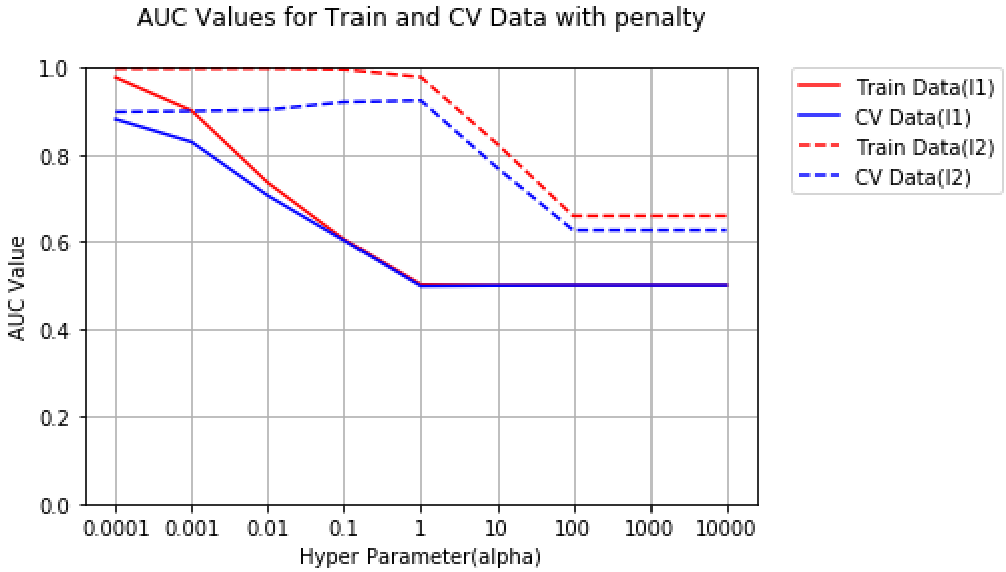
In [42]:

```
support(X_train_tf,X_cv_tf)
```

```
Fitting 3 folds for each of 18 candidates, totalling 54 fits
```

```
[Parallel(n_jobs=1)]: Done  54 out of  54 | elapsed:    5.2s finished
```

```
best optimized alpha: 1
best optimized regularization: l2
```

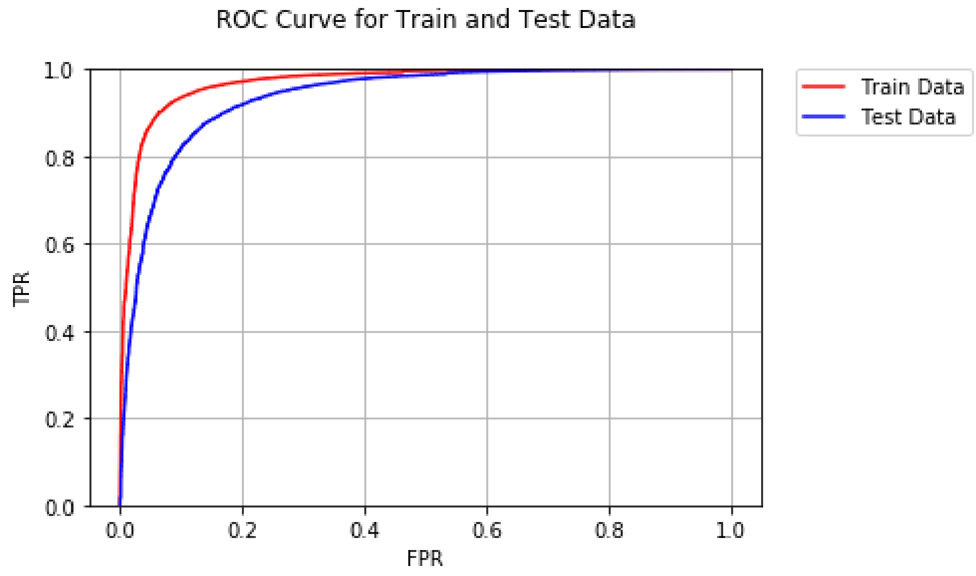


In [43]:

```
SGD=SGDClassifier(penalty="l2",alpha=1)
from sklearn.metrics import roc_auc_score
SGD.fit(X_train_tf,y_train)

y_train_predict_proba=SGD.decision_function(X_train_tf)
y_test_predict_proba=SGD.decision_function(X_test_tf)
fpr,tpr,threshold=roc_curve(y_train,y_train_predict_proba[:])
fpr1,tpr1,threshold1=roc_curve(y_test,y_test_predict_proba[:])

print("The AUC value for test data is ",roc_auc_score( y_test, y_test_p
redict_proba))

plt.plot(fpr,tpr,'r', label = 'Train Data')
plt.plot(fpr1,tpr1,'b', label = 'Test Data')
plt.ylim(0,1)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.
)
plt.grid(True)
plt.title("ROC Curve for Train and Test Data\n")
plt.xlabel("FPR")
```

```
plt.ylabel("TPR")
plt.show()
```

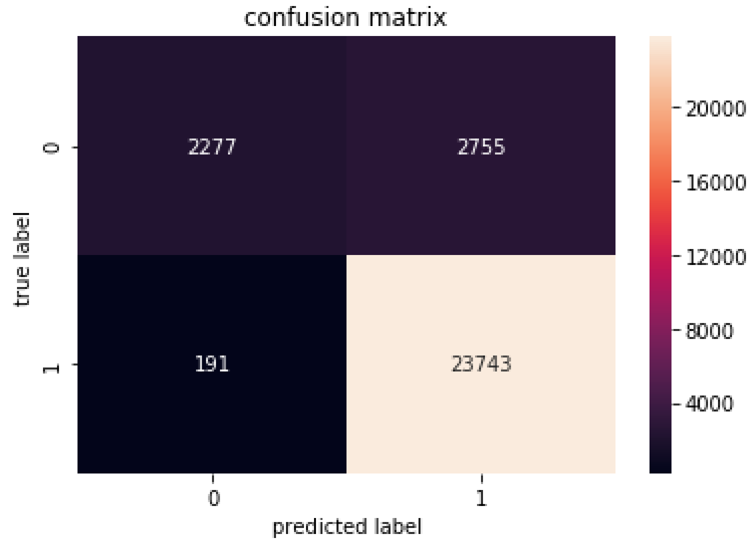
```
The AUC value for test data is  0.9354996577100009
```



CONFUSION MATRIX

In [44]:
```
confusion_matrix(X_train_tf,X_test_tf)
```

```
[[ 3025  2713]
 [  126 33536]]
[[ 2277  2755]
 [  191 23743]]
***********************************************************************
*****************************
confusion matrix for test data
```



TOP 10 MOST IMPORTANCE FEATURES OF POSITIVE AND NEGATIVE

In [45]:
```
SGD = SGDClassifier(penalty="l2",alpha=1)
SGD.fit(X_train_tf,y_train)
feat_log = SGD.coef_

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
s = tf_idf_vect.fit_transform(X_train)
s = pd.DataFrame(feat_log.T,columns=['+ve'])
s['feature'] = vectorizer.get_feature_names()
```

In [46]:
```
v = s.sort_values(by = '+ve',kind = 'quicksort',ascending= False)
print("Top 10  important features of positive class", np.array(v['featu
re'][:10]))
print("*"*100)
print("Top 10  important features of negative class",np.array(v.tail(10
)['feature']))
```

```
Top 10  important features of positive class ['great' 'good' 'love' 'be
st' 'delicious' 'loves' 'excellent' 'perfect'
 'favorite' 'wonderful']
***********************************************************************
*****************************
Top 10  important features of negative class ['not purchase' 'threw' 'd
isappointment' 'not buy' 'disappointed'
 'not worth' 'terrible' 'awful' 'horrible' 'worst']
```

AVG W2V (LINEAR KERNEL)

In [47]:
```
X=preprocessed_reviews
Y=final['Score']
```

In [48]:
```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3,shuffle=False) # this is time based splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33,shuffle=False)
```

In [49]:
```
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [50]:
```
sent_of_train=[]
for sent in X_train:
    sent_of_train.append(sent.split())
```

In [51]:
```
sent_of_cv=[]
for sent in X_cv:
    sent_of_cv.append(sent.split())
```

```
sent_of_test=[]
for sent in X_test:
    sent_of_test.append(sent.split())

# Train your own Word2Vec model using your own train text corpus
# min_count = 5 considers only words that occured atleast 5 times
w2v_model=Word2Vec(sent_of_train,min_count=5,size=50, workers=4)

w2v_words = list(w2v_model.wv.vocab)
```

In [52]:

```
train_vectors = [];
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    train_vectors.append(sent_vec)

cv_vectors = [];
for sent in sent_of_cv:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    cv_vectors.append(sent_vec)
```

```python
# compute average word2vec for each review for X_test .
test_vectors = [];
for sent in sent_of_test:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        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)
```

In [53]:
```python
X_train_wv=train_vectors
X_cv_wv=cv_vectors
X_test_wv=test_vectors
```

STANDARDISING THE DATA

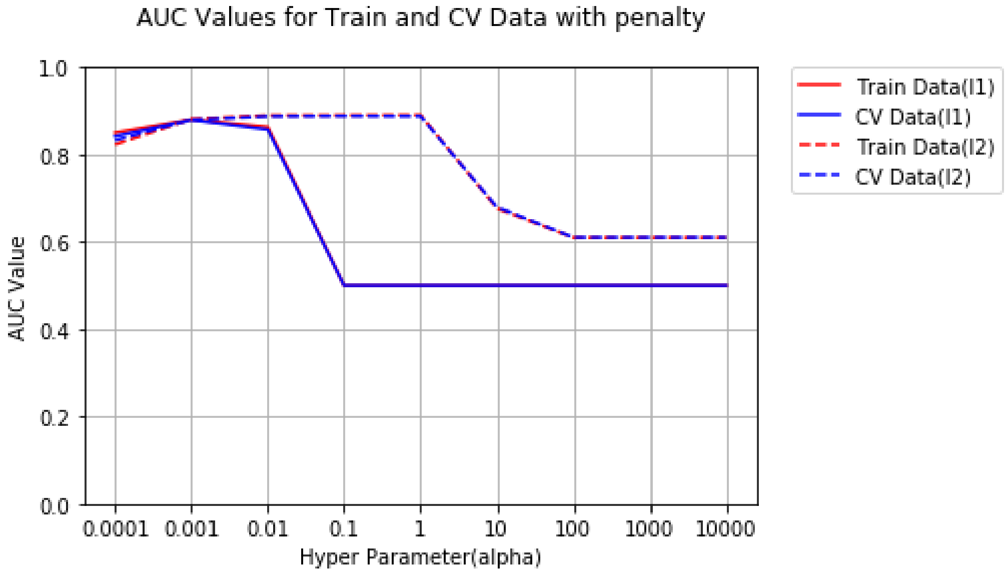
In [54]:
```python
from sklearn.preprocessing import StandardScaler
standardised=StandardScaler(with_mean=False)
X_train_wv=standardised.fit_transform(X_train_wv)
X_cv_wv=standardised.transform(X_cv_wv)
X_test_wv=standardised.transform(X_test_wv)
```

In [55]:
```python
support(X_train_wv,X_cv_wv)
```

```
Fitting 3 folds for each of 18 candidates, totalling 54 fits
```

```
[Parallel(n_jobs=1)]: Done  54 out of  54 | elapsed:    6.1s finished
```

```
best optimized alpha: 0.1
best optimized regularization: l2
```

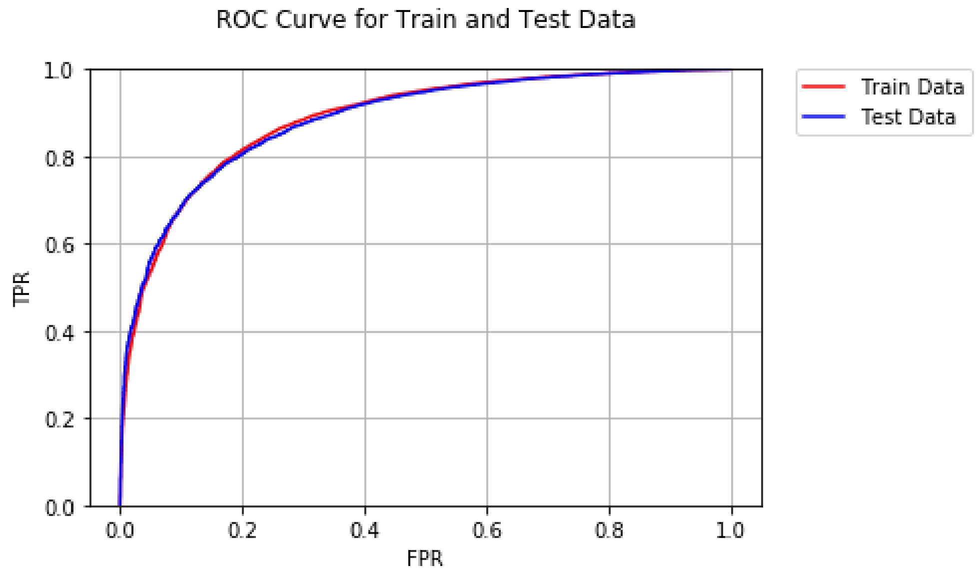


In [56]:

```
SGD=SGDClassifier(penalty="l2",alpha=0.01)
from sklearn.metrics import roc_auc_score
SGD.fit(X_train_wv,y_train)

y_train_predict_proba=SGD.decision_function(X_train_wv)
y_test_predict_proba=SGD.decision_function(X_test_wv)
fpr,tpr,threshold=roc_curve(y_train,y_train_predict_proba[:])
fpr1,tpr1,threshold1=roc_curve(y_test,y_test_predict_proba[:])

print("The AUC value for test data is ",roc_auc_score( y_test, y_test_p
redict_proba))

plt.plot(fpr,tpr,'r', label = 'Train Data')
plt.plot(fpr1,tpr1,'b', label = 'Test Data')
plt.ylim(0,1)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.
)
plt.grid(True)
plt.title("ROC Curve for Train and Test Data\n")
plt.xlabel("FPR")
```

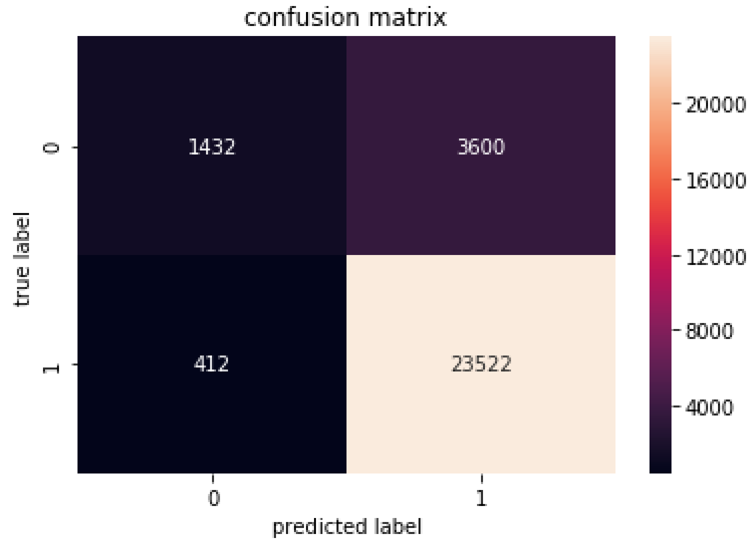
```
plt.ylabel("TPR")
plt.show()
```

```
The AUC value for test data is  0.8860180447210221
```



In [57]:
```
confusion_matrix(X_train_wv,X_test_wv)
```

```
[[ 1609  4129]
 [  523 33139]]
[[ 1432  3600]
 [  412 23522]]
************************************************************************
*****************************
confusion matrix for test data
```



TF-IDF W2V (LINEAR KERNEL)

In [61]:
```
X=preprocessed_reviews
Y=final["Score"]
```

In [62]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3,shuffle=False) # this is random splitting
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_
size=0.33,shuffle=False)
```

In [63]:
```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [64]:
```
tfidf_feat = model.get_feature_names() # tfidf words/col-names
```

```
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

tfidf_train_vectors = []; # the tfidf-w2v for each sentence/review is s
tored in this list
row=0;
for sent in tqdm(sent_of_train): # 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/r
eview
    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_train_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████| 39400/39400 [12:50<00:00, 5
1.15it/s]
```

In [65]:

```
tfidf_cv_vectors = []; # the tfidf-w2v for each sentence/review is stor
ed in this list
row=0;
for sent in tqdm(sent_of_cv): # 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/r
eview
    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_cv_vectors.append(sent_vec)
    row += 1
```

```
100%|████████████████████████████████████| 19407/19407 [06:28<00:00, 4
9.97it/s]
```

In [66]:

```
tfidf_test_vectors = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(sent_of_test): # 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/r
eview
    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_test_vectors.append(sent_vec)
    row += 1
```

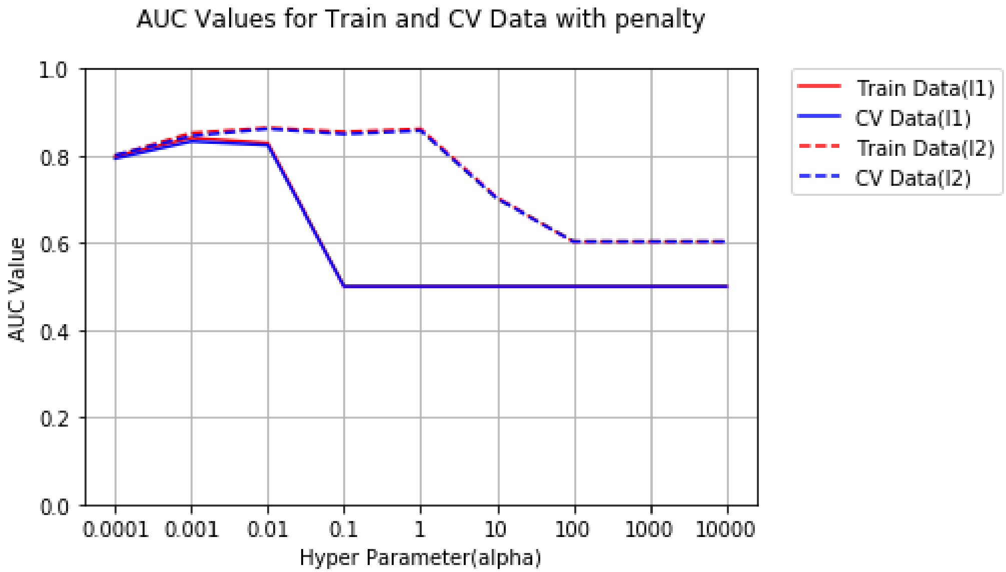
```
100%|████████████████████████████████████| 28966/28966 [09:39<00:00, 4
9.95it/s]
```

In [67]:
```
X_train_tw=tfidf_train_vectors
X_cv_tw=tfidf_cv_vectors
X_test_tw=tfidf_test_vectors
```

In [68]:
```
from sklearn.preprocessing import StandardScaler
standardised=StandardScaler(with_mean=False)
X_train_tw=standardised.fit_transform(X_train_tw)
X_cv_tw=standardised.transform(X_cv_tw)
X_test_tw=standardised.transform(X_test_tw)
```

In [69]:
```
support(X_train_tw,X_cv_tw)
```

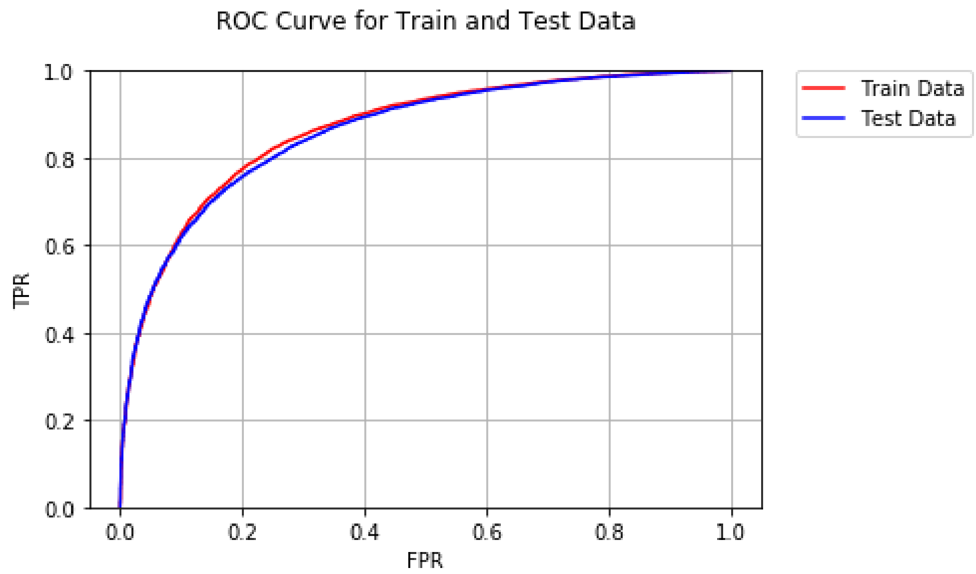
```
Fitting 3 folds for each of 18 candidates, totalling 54 fits
[Parallel(n_jobs=1)]: Done  54 out of  54 | elapsed:    6.2s finished
best optimized alpha: 0.01
best optimized regularization: l2
```



In [70]:
```
SGD=SGDClassifier(penalty="l2",alpha=0.01)
from sklearn.metrics import roc_auc_score
```

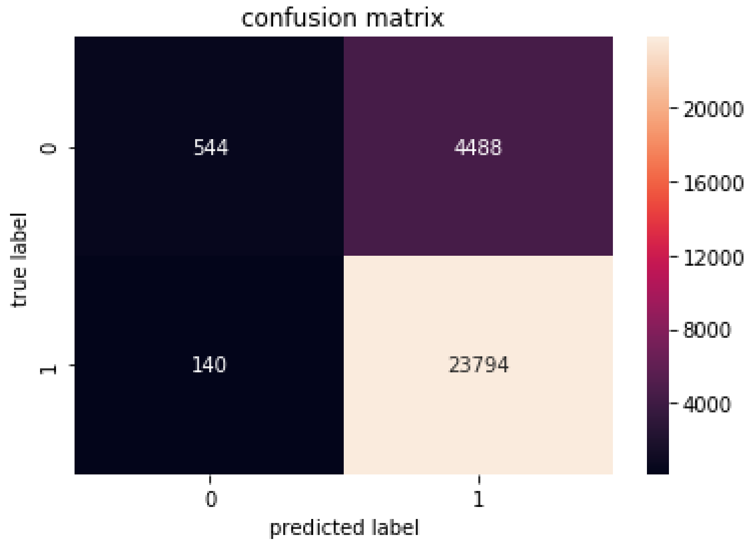
```
SGD.fit(X_train_tw,y_train)

y_train_predict_proba=SGD.decision_function(X_train_tw)
y_test_predict_proba=SGD.decision_function(X_test_tw)
fpr,tpr,threshold=roc_curve(y_train,y_train_predict_proba[:])
fpr1,tpr1,threshold1=roc_curve(y_test,y_test_predict_proba[:])

print("The AUC value for test data is ",roc_auc_score( y_test, y_test_p
redict_proba))

plt.plot(fpr,tpr,'r', label = 'Train Data')
plt.plot(fpr1,tpr1,'b', label = 'Test Data')
plt.ylim(0,1)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.
)
plt.grid(True)
plt.title("ROC Curve for Train and Test Data\n")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```

```
The AUC value for test data is  0.8597853407283385
```



In [71]:
```
confusion_matrix(X_train_tw,X_test_tw)
```

```
[[  561  5177]
 [  149 33513]]
[[  544  4488]
 [  140 23794]]
**********************************************************************
****************************
confusion matrix for test data
```



In [119]:
```
from tabulate import tabulate
print(tabulate ([['BOW(l2)', 0.1, 92],['TF-IDF(l2)',1,93],['AVG-W2V(L
2)',0.01,88.69] , ['TFIDF-W2V(L2)',0.01,85.72]],    headers=['Vectorize
r(BEST_REGULARIZATION)', 'best_ALPHA','AUC_test']))
```

```
Vectorizer(BEST_REGULARIZATION)      best_ALPHA    AUC_test
---------------------------------  ------------  ----------
BOW(l2)                                     0.1       92
TF-IDF(l2)                                  1         93
AVG-W2V(L2)                                 0.01      88.69
TFIDF-W2V(L2)                               0.01      85.72
```

1. cost of computation is very low.
2. In linear SVM the best model is TF-idf with l2 regularization.
3. even we can improve the model by taking more data points and feature engineering.

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**

    - 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

[5.1.1] Applying Linear SVM on BOW, SET 1

In [3]:
```
# Please write all the code with proper documentation
```

[5.1.2] Applying Linear SVM on TFIDF, SET 2

In [3]:
```
# Please write all the code with proper documentation
```

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

In [3]:
```
# Please write all the code with proper documentation
```

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

In [3]:
```
# Please write all the code with proper documentation
```

[5.2] RBF SVM

[5.2.1] Applying RBF SVM on BOW, SET 1

In [3]:
```
# Please write all the code with proper documentation
```

[5.2.2] Applying RBF SVM on TFIDF, SET 2

In [3]:
```
# Please write all the code with proper documentation
```

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

In [3]:
```
# Please write all the code with proper documentation
```

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

In [3]:
```
# Please write all the code with proper documentation
```

[6] Conclusions

In [4]:
```
# Please compare all your models using Prettytable library
```