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
```

```python
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]:

```python
# 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 50000""", 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 (50000, 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]:
```
(46072, 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]:
```
92.144
```

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

In [14]:
```
#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()
```

```
(46071, 10)
```

Out[14]:
```
1    38479
0     7592
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 [15]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
```

```python
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
This was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
==================================================
Speaking as another Texan, I think the first rule for these delicious t
reats is to NOT order them during spring or summer. In fact, your safes
t bet is to ONLY order them in the dead of winter.  LOL!  As long as yo
u do that, be prepared for a truly amazing treat!  This package comes w
ith 12 bite-sized delicacies.  The chocolate is high-quality, the nuts
are crunchy, and the overall taste couldn't be better.  Definitely wort
h the price!
==================================================
Fast, easy and definitely delicious.  Makes a great cup of coffee and v
ery easy to make.  Good purchase.  Will continue to order from here.<br
/>Thanx...
==================================================
Naturally this review is based upon my cat's intake of Petite Cuisine.
She's not a particular picky eater, so I can't say much about that. How
ever, she looks to really enjoy this brand of cat food. I have tried so
me brands of wet food in the past that have made her sick (I know cats
seem to have digestive systems that are prone to upsetting!) Petite Cui
sine did not have any effect there, and she really enjoyed all the flav
ors. I don't feed her wet food often, usually just some tuna fish now a
nd then. So, although this food is expensive if you used it at every me
al, it is priced about the same as tuna, so it fits my needs perfectly.
I'm sure my cat will enjoy the rest of this case, and I can keep the ca
```

```
nned tuna to myself for now :-)
==================================================
```

In [16]:
```python
# remove urls from text python: https://stackoverflow.com/a/40823105/40
84039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
This was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
```

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

```
This was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
==================================================
Speaking as another Texan, I think the first rule for these delicious t
reats is to NOT order them during spring or summer. In fact, your safes
t bet is to ONLY order them in the dead of winter.  LOL!  As long as yo
u do that, be prepared for a truly amazing treat!  This package comes w
ith 12 bite-sized delicacies.  The chocolate is high-quality, the nuts
are crunchy, and the overall taste couldn't be better.  Definitely wort
h the price!
==================================================
Fast, easy and definitely delicious.  Makes a great cup of coffee and v
ery easy to make.  Good purchase.  Will continue to order from here.Tha
nx...
==================================================
Naturally this review is based upon my cat's intake of Petite Cuisine.
She's not a particular picky eater, so I can't say much about that. How
ever, she looks to really enjoy this brand of cat food. I have tried so
me brands of wet food in the past that have made her sick (I know cats
seem to have digestive systems that are prone to upsetting!) Petite Cui
sine did not have any effect there, and she really enjoyed all the flav
ors. I don't feed her wet food often, usually just some tuna fish now a
nd then. So, although this food is expensive if you used it at every me
al, it is priced about the same as tuna, so it fits my needs perfectly.
I'm sure my cat will enjoy the rest of this case, and I can keep the ca
nned tuna to myself for now :-)
```

In [18]:

```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 [19]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
Fast, easy and definitely delicious.  Makes a great cup of coffee and v
ery easy to make.  Good purchase.  Will continue to order from here.<br
/>Thanx...
==================================================
```

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)
```

```
This was a really good idea and the final product is outstanding. I use
the decals on my car window and everybody asks where i bought the decal
s i made.  Two thumbs up!
```

In [21]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
Fast easy and definitely delicious Makes a great cup of coffee and very
easy to make Good purchase Will continue to order from here br Thanx
```

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]:

```
# 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%|██████████████████████████████████████████| 46071/46071 [00:19<00:00, 231
4.76it/s]
```

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

Out[24]:
```
'fast easy definitely delicious makes great cup coffee easy make good p
urchase continue order thanx'
```

[3.2] Preprocessing Review Summary

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

[4] Featurization

Applying Logistic Regression on L1 regularization on BOW

In [29]:
```
X=preprocessed_reviews
np.shape(X)
```

Out[29]:
```
(46071,)
```

In [30]:
```
Y=final["Score"]
Y.shape
```

Out[30]:
```
(46071,)
```

In [36]:
```
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)

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

```
(20680,) (20680,)
(10187,) (10187,)
(15204,) (15204,)
```

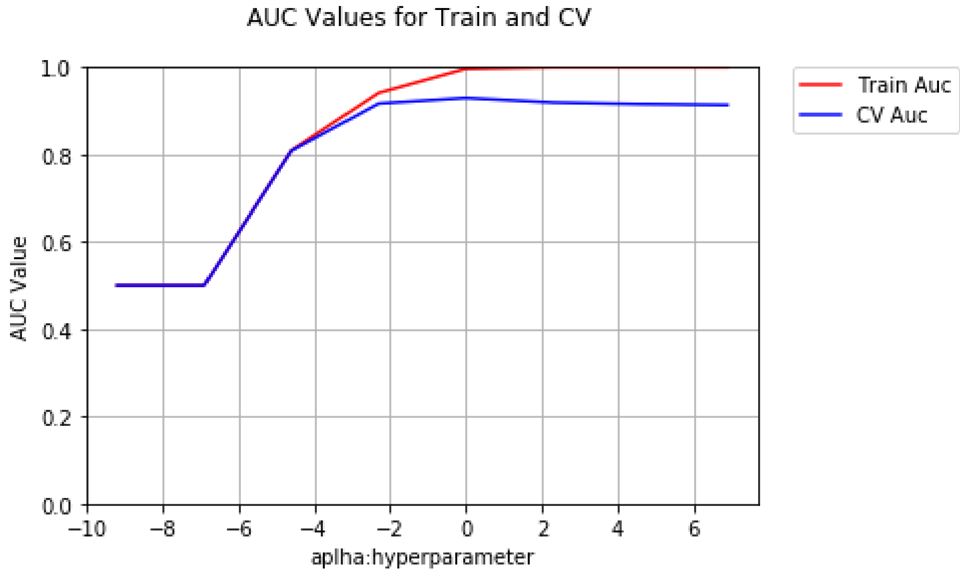
In [39]:
```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer=CountVectorizer(min_df=10,max_features=10000,ngram_range=(1,
2))
vectorizer.fit(X_train)
X_train_bow=vectorizer.transform(X_train)
X_cv_bow=vectorizer.transform(X_cv)
X_test_bow=vectorizer.transform(X_test)
print("*"*100)
print("After Bag of words(one hot encoding)")
print(np.shape(X_train_bow),y_train.shape)
print(np.shape(X_cv_bow),y_cv.shape)
print(np.shape(X_test_bow),y_test.shape)
```

```
**********************************************************************
******************************
After Bag of words(one hot encoding)
(20680, 10000) (20680,)
(10187, 10000) (10187,)
(15204, 10000) (15204,)
```

In [41]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
```

In [54]:
```
def log_opt(train,cv,penalt):
    train_auc = []
    cv_auc = []
    inv_lambda=[0.0001,0.001,0.01,0.1,1,10,100,1000]
    for i in inv_lambda:
        LR=LogisticRegression(penalty=penalt,C=i)
        LR.fit(train,y_train)

        pred_prob = LR.predict_proba(cv)
        pred_prob_train = LR.predict_proba(train)

        cv_auc.append(roc_auc_score(y_cv, pred_prob[:,1]))
        train_auc.append(roc_auc_score(y_train, pred_prob_train[:,1]))
    plt.plot(np.log(inv_lambda),train_auc,'r', label = 'Train Auc')
    plt.plot(np.log(inv_lambda),cv_auc,'b', label = 'CV Auc')
    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 \n")
    plt.xlabel("aplha:hyperparameter")
    plt.ylabel("AUC Value")
    plt.show()



    mx = 0
    for i in range(len(cv_auc)):
        if(cv_auc[i]> cv_auc[mx]):
            mx = i
    opt = inv_lambda[mx]
    print("The optimal value of c = ", opt)
```

In [55]:
```
log_opt(X_train_bow,X_cv_bow,"l1")
```

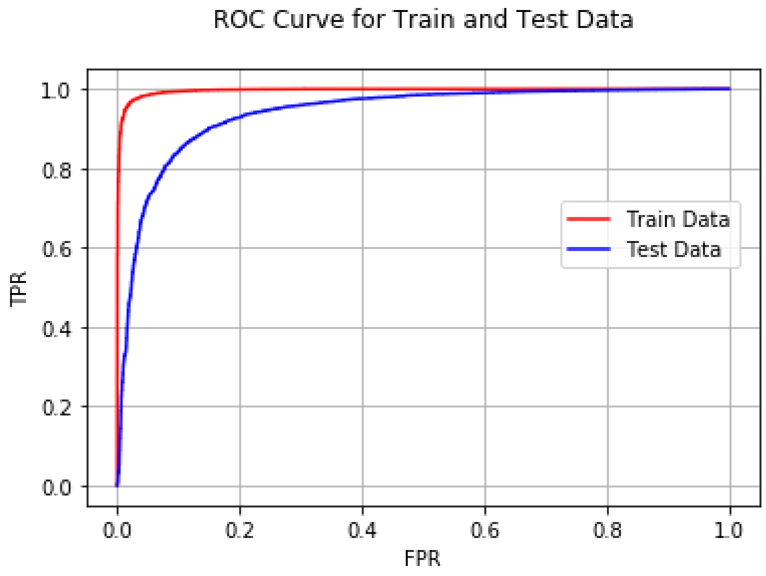



```
The optimal value of c =  1
```

In [98]:
```
LR=LogisticRegression(penalty="l1",C=1)
LR.fit(X_train_bow,y_train)


Y_test_pred_proba=LR.predict_proba(X_test_bow)
Y_train_pred_proba=LR.predict_proba(X_train_bow)

fpr,tpr,threshold=roc_curve(y_train,Y_train_pred_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,Y_test_pred_proba[:,1])
print("AUC value for test data ",roc_auc_score(y_test,Y_test_pred_proba
[:,1]))
print("AUC value for train data ",roc_auc_score(y_train,Y_train_pred_pr
oba[:,1]))
plt.plot(fpr,tpr,'r', label = 'Train Data')
plt.plot(fpr1,tpr1,'b', label = 'Test Data')
plt.legend(bbox_to_anchor=(1, 0.5), loc='lower right', borderaxespad=1)
plt.grid(True)
plt.title("ROC Curve for Train and Test Data\n")
plt.xlabel("FPR")
```

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

```
AUC value for test data  0.9392563415283378
AUC value for train data  0.996375144370256
```



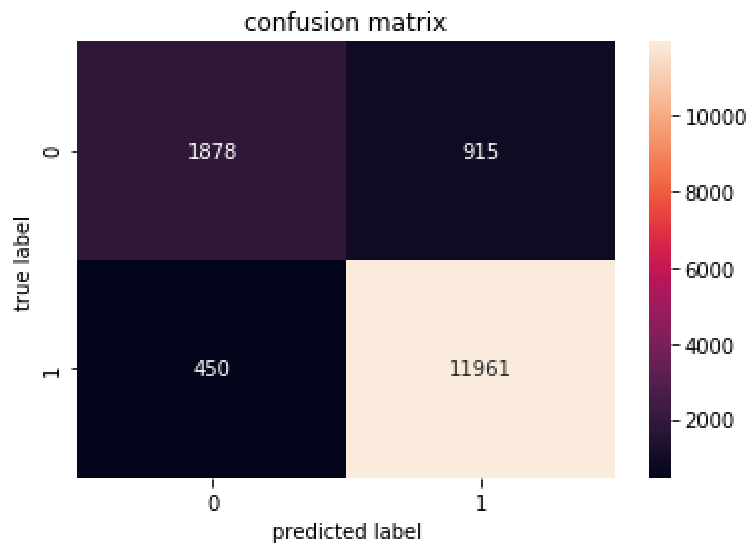
In [99]:

```
def confusion_matrix(train,test):
    from sklearn.metrics import confusion_matrix
    from sklearn.metrics import confusion_matrix
    Y_test_pred=LR.predict(test)
    Y_train_pred=LR.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 [79]:
```
confusion_matrix(X_train_bow,X_test_bow)
```

```
[[ 2642   376]
 [   68 17594]]
[[ 1878   915]
 [  450 11961]]
************************************************************************
*****************************
confusion matrix for test data
```



Sparsity

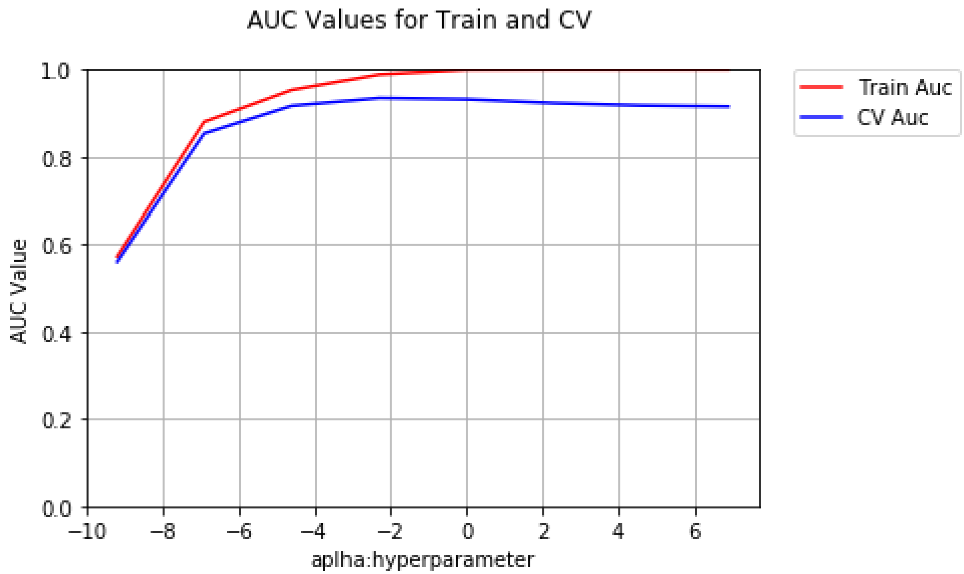
In [104]:
```
w = LR.coef_
a = np.count_nonzero(w)
b = w.size
```

```
print("\n\nSparsity on weight vector obtained using L1 regularization i
s : ",(b-a)/b*100,"percent")
```

```
Sparsity on weight vector obtained using L1 regularization is :  77.8 p
ercent
```

Applying Logistic Regression on L2 regularization on BOW

In [105]:
```
log_opt(X_train_bow,X_cv_bow,"l2")
```

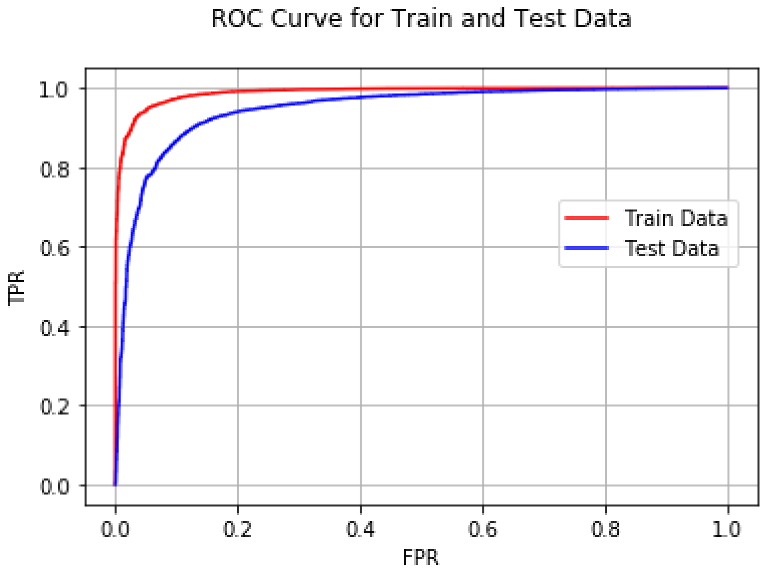



```
The optimal value of c =  0.1
```

In [106]:
```
LR=LogisticRegression(penalty="l2",C=0.1)
LR.fit(X_train_bow,y_train)


Y_test_pred_proba=LR.predict_proba(X_test_bow)
```

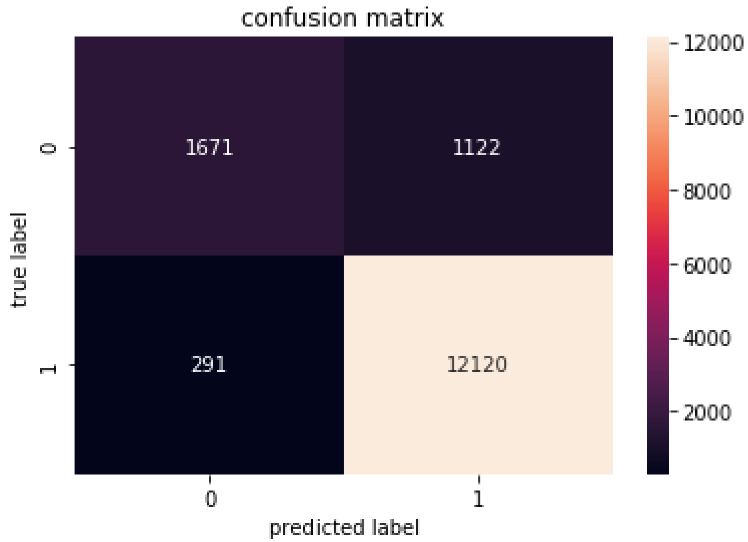
```
Y_train_pred_proba=LR.predict_proba(X_train_bow)

fpr,tpr,threshold=roc_curve(y_train,Y_train_pred_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,Y_test_pred_proba[:,1])
print("AUC value for test data ",roc_auc_score(y_test,Y_test_pred_proba
[:,1]))
print("AUC value for train data ",roc_auc_score(y_train,Y_train_pred_pr
oba[:,1]))
plt.plot(fpr,tpr,'r', label = 'Train Data')
plt.plot(fpr1,tpr1,'b', label = 'Test Data')
plt.legend(bbox_to_anchor=(1, 0.5), loc='lower right', borderaxespad=1)
plt.grid(True)
plt.title("ROC Curve for Train and Test Data\n")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```

```
AUC value for test data  0.9461011957590606
AUC value for train data  0.9887098726480059
```



In [108]:
```
confusion_matrix(X_train_bow,X_test_bow)
```

```
[[ 2241   777]
 [   95 17567]]
[[ 1671  1122]
 [  291 12120]]
*************************************************************************
*****************************
confusion matrix for test data
```



Performing pertubation test (multicollinearity check) on BOW

In [138]:

```
LR=LogisticRegression()
LR.fit(X_train_bow,y_train)
```

Out[138]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=
True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=
1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.00
```

```
01,
          verbose=0, warm_start=False)
```

In [139]:
```
weight1=LR.coef_
weight1
```

Out[139]:
```
array([[0.14859133, 0.62582013, 0.15003421, ..., 0.17049267, 0.0665383
2,
        0.14645865]])
```

In [140]:
```
noisy_train=X_train_bow.astype(float)
noisy_train.data+=np.random.normal(-0.0001,0.0001,1 )
```

In [141]:
```
LR=LogisticRegression()
LR.fit(noisy_train,y_train)
weight2=LR.coef_
```

In [146]:
```
weight1+=10**-6
weight2+=10**-6
```

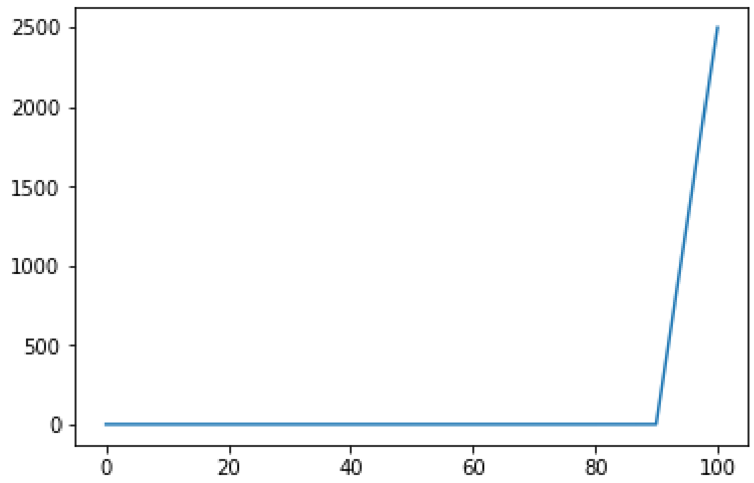
In [147]:
```
percent_change_between_weights=abs((weight1-weight2)/(weight1))*100
```

In [155]:
```
d=range(0,101,10)
for i in d:
    print(i, "th percentile :" ,np.percentile(percent_change_between_we
ights,i))
plt.plot(d,np.percentile(percent_change_between_weights,d))
```

```
0 th percentile : 6.092189333777582e-06
10 th percentile : 0.00642861090485667
20 th percentile : 0.01322183018746459
30 th percentile : 0.020998308940257 46
40 th percentile : 0.02997062709023482
50 th percentile : 0.04078078090654698
60 th percentile : 0.055524588338558056
70 th percentile : 0.0776668203002136
80 th percentile : 0.12411996775846965
```

```
90 th percentile : 0.2569052651162836
100 th percentile : 2498.0723215786006
```

Out[155]: [<matplotlib.lines.Line2D at 0x16122370>]

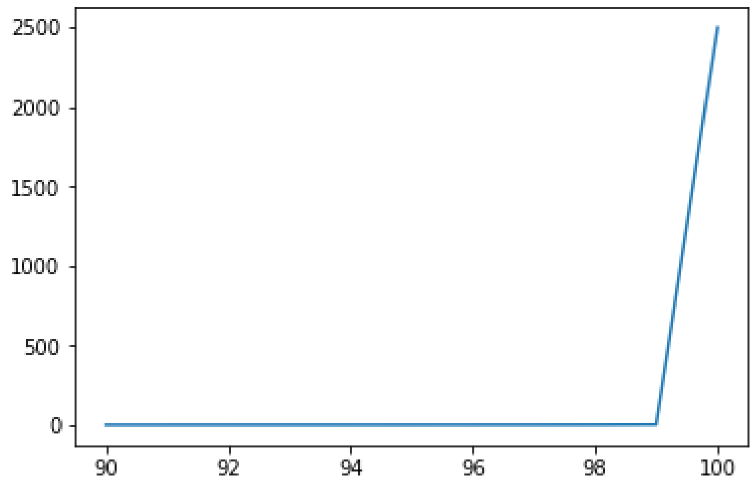


In [161]:

```
d=range(90,101,1)
for i in d:
    print(i, "th percentile :" ,np.percentile(percent_change_between_we
ights,i))
plt.plot(d,np.percentile(percent_change_between_weights,d))
```

```
90 th percentile : 0.2569052651162836
91 th percentile : 0.28611181960159093
92 th percentile : 0.3245273344437252
93 th percentile : 0.3703877629118301
94 th percentile : 0.43665478542229136
95 th percentile : 0.5217222107792607
96 th percentile : 0.6904925607699413
97 th percentile : 0.902788529617884
98 th percentile : 1.3655107162893607
99 th percentile : 2.8895312266560387
100 th percentile : 2498.0723215786006
```

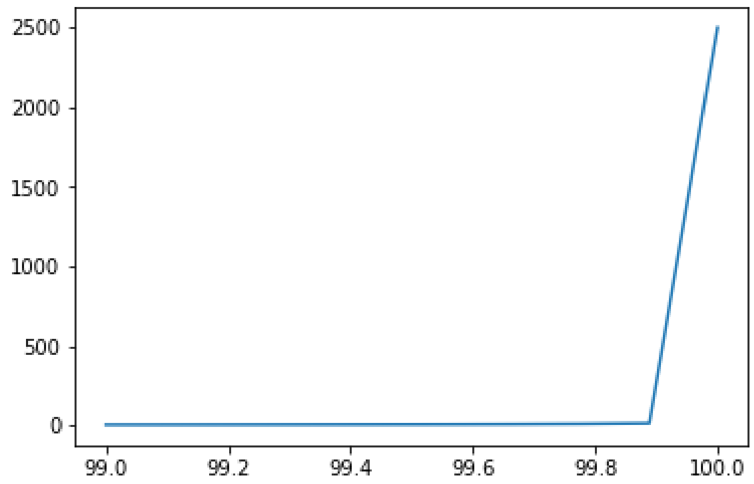
Out[161]: [<matplotlib.lines.Line2D at 0x164466d0>]



In [167]:
```
d=np.linspace(99,100,10)
for i in d:
    print(i, "th percentile :" ,np.percentile(percent_change_between_we
ights,i))
plt.plot(d,np.percentile(percent_change_between_weights,d))
```

```
99.0 th percentile : 2.8895312266560387
99.11111111111111 th percentile : 3.1813830357684516
99.22222222222223 th percentile : 3.5434359227330843
99.33333333333333 th percentile : 4.052278077009432
99.44444444444444 th percentile : 4.621112987221331
99.55555555555556 th percentile : 5.383870477570108
99.66666666666667 th percentile : 6.596806507088486
99.77777777777777 th percentile : 8.741870591402673
99.88888888888889 th percentile : 12.533169609423584
100.0 th percentile : 2498.0723215786006
```

Out[167]:
```
[<matplotlib.lines.Line2D at 0x163a3190>]
```



In [184]:
```
diff = (abs(weight1 - weight2)/weight1) * 100
q = diff[np.where(diff > 10)].size
print("Percentage of features which did not change by more than 10% is
 :",(weight1.size - q)/weight1.size*100)
```

```
Percentage of features which did not change by more than 10% is : 99.92
```

Here only 0.8 percent of values are changed more than 10 percent.hence it is not multicollinearity

Top 10 feature for both negative and positive class

In [219]:
```
#refered from github example for dataframe
LR = LogisticRegression()
LR.fit(X_train_bow,y_train)
feat_log = LR.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 [220]:

```
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 ['not disappointed' 'delic
ious' 'pleased' 'perfect' 'amazing' 'awesome'
 'product great' 'yummy' 'excellent' 'hooked']
**********************************************************************
*****************************
Top 10  important features of negative class ['two stars' 'disgusting'
'threw' 'not happy' 'awful' 'died' 'terrible'
 'disappointing' 'not worth' 'worst']
```

[5.2] Logistic Regression on TFIDF

In [299]:

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

In [300]:

```
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)

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

```
(20680,) (20680,)
(10187,) (10187,)
```

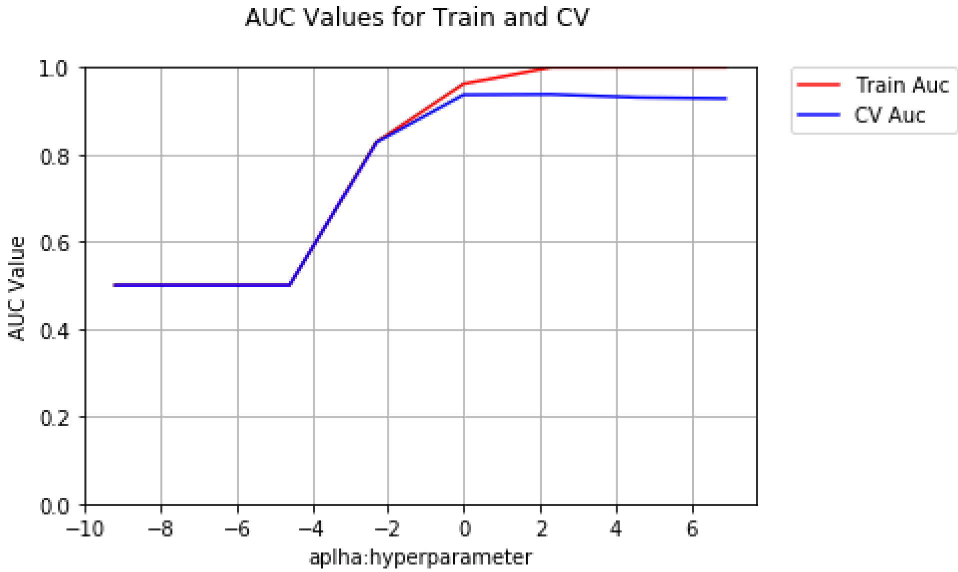
```
(15204,) (15204,)
```

In [301]:
```python
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
(20680, 10000) (20680,)
(10187, 10000) (10187,)
(15204, 10000) (15204,)
```

[5.1.2] Applying Logistic Regression with L1 regularization on TF-IDF

In [302]:
```python
log_opt(X_train_tf,X_cv_tf,"l1")
```



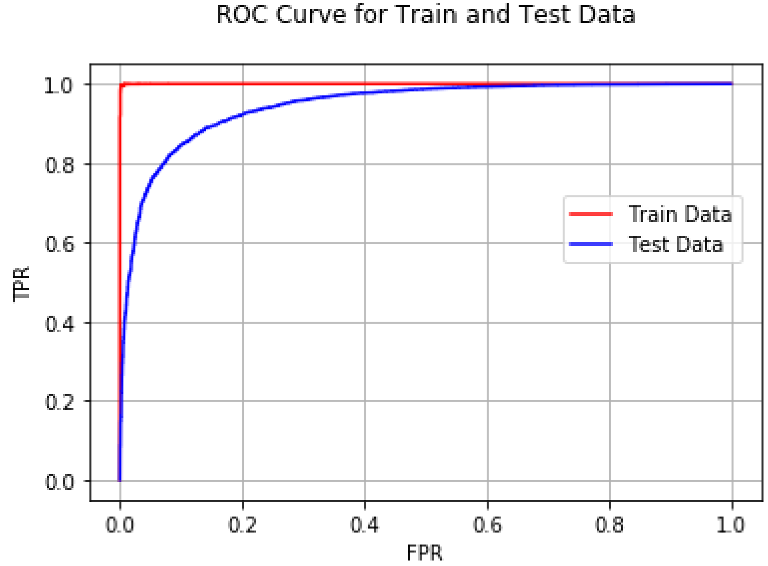

```
The optimal value of c =  10
```

In [303]:

```
LR=LogisticRegression(penalty="l1",C=10)
LR.fit(X_train_tf,y_train)


Y_test_pred_proba=LR.predict_proba(X_test_tf)
Y_train_pred_proba=LR.predict_proba(X_train_tf)

fpr,tpr,threshold=roc_curve(y_train,Y_train_pred_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,Y_test_pred_proba[:,1])
print("AUC value for test data ",roc_auc_score(y_test,Y_test_pred_proba
[:,1]))
print("AUC value for train data ",roc_auc_score(y_train,Y_train_pred_pr
oba[:,1]))
plt.plot(fpr,tpr,'r', label = 'Train Data')
plt.plot(fpr1,tpr1,'b', label = 'Test Data')
plt.legend(bbox_to_anchor=(1, 0.5), loc='lower right', borderaxespad=1)
plt.grid(True)
plt.title("ROC Curve for Train and Test Data\n")
plt.xlabel("FPR")
```

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

```
AUC value for test data  0.9447843223053547
AUC value for train data  0.9998999698258567
```

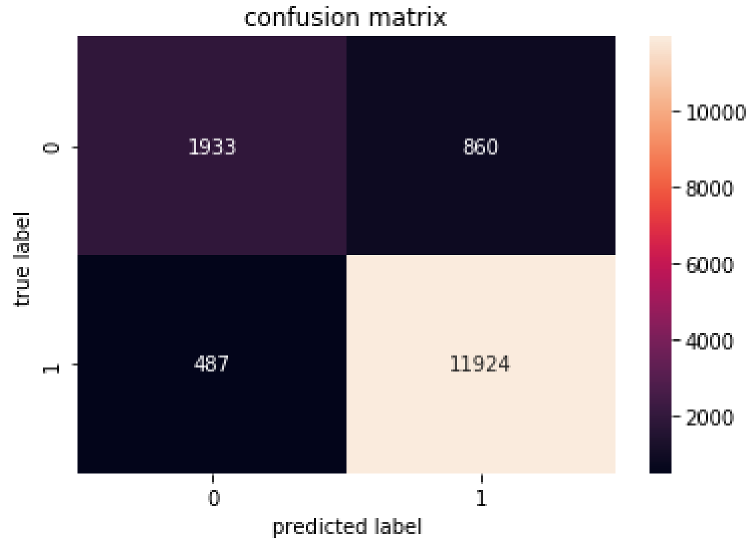


confusion matrix

In [304]:

```
confusion_matrix(X_train_tf,X_test_tf)
```

```
[[ 2974    44]
 [   11 17651]]
[[ 1933   860]
 [  487 11924]]
************************************************************************
*****************************
confusion matrix for test data
```



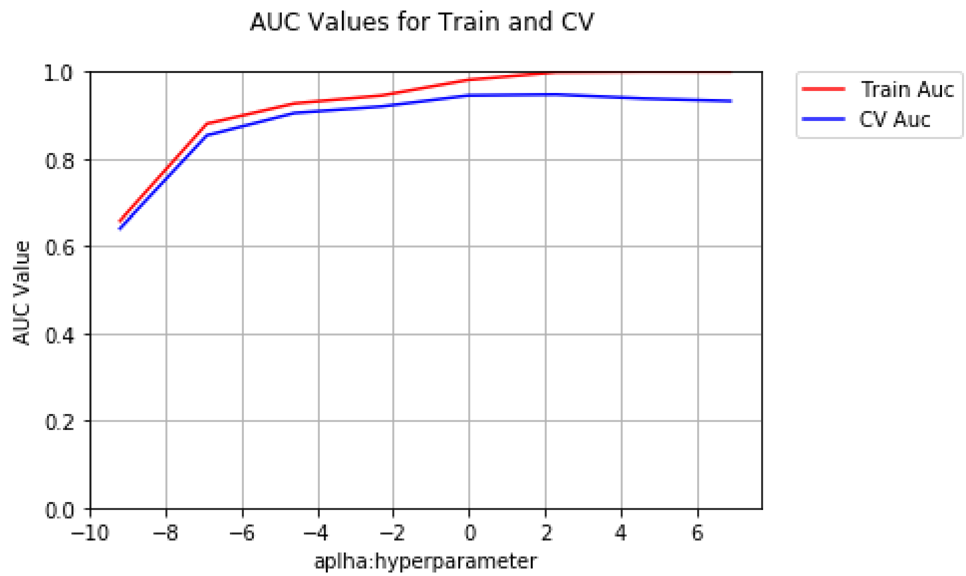
sparsity

In [305]:
```
w = LR.coef_
a = np.count_nonzero(w)
b = w.size
print("\n\nSparsity on weight vector obtained using L1 regularization i
s : ",(b-a)/b*100,"percent")
```

```
Sparsity on weight vector obtained using L1 regularization is :  69.6 p
ercent
```

[5.1.2] Applying Logistic Regression with L2 regularization on TF-IDF

In [306]:
```
log_opt(X_train_tf,X_cv_tf,"l2")
```



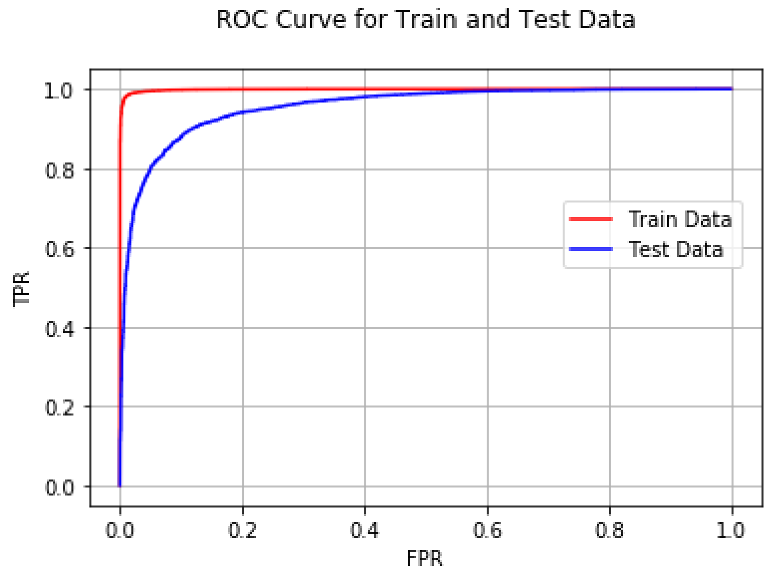

```
The optimal value of c =  10
```

In [307]:

```
LR=LogisticRegression(penalty="l2",C=10)
LR.fit(X_train_tf,y_train)


Y_test_pred_proba=LR.predict_proba(X_test_tf)
Y_train_pred_proba=LR.predict_proba(X_train_tf)

fpr,tpr,threshold=roc_curve(y_train,Y_train_pred_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,Y_test_pred_proba[:,1])
print("AUC value for test data ",roc_auc_score(y_test,Y_test_pred_proba
[:,1]))
print("AUC value for train data ",roc_auc_score(y_train,Y_train_pred_pr
oba[:,1]))
plt.plot(fpr,tpr,'r', label = 'Train Data')
plt.plot(fpr1,tpr1,'b', label = 'Test Data')
plt.legend(bbox_to_anchor=(1, 0.5), loc='lower right', borderaxespad=1)
plt.grid(True)
plt.title("ROC Curve for Train and Test Data\n")
plt.xlabel("FPR")
```

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

```
AUC value for test data  0.9551249580147059
AUC value for train data  0.9987577085330842
```

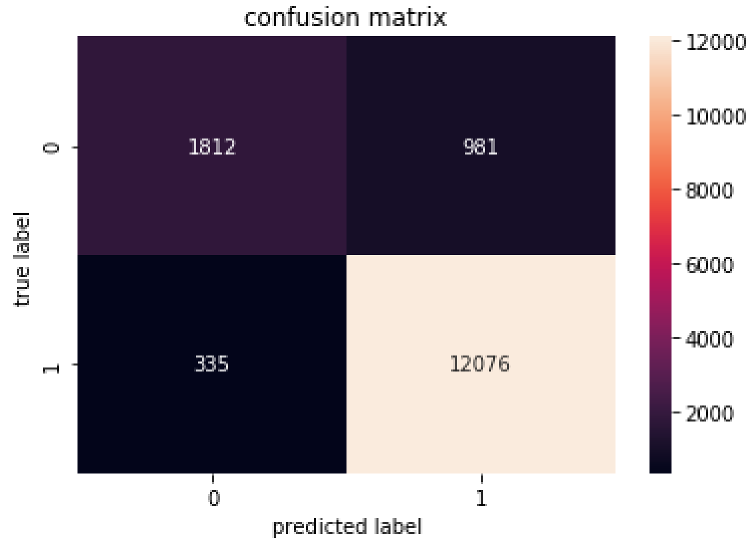


Confusion matrix

In [308]:

```
confusion_matrix(X_train_tf,X_test_tf)
```

```
[[ 2762   256]
 [   43 17619]]
[[ 1812   981]
 [  335 12076]]
***********************************************************************
*****************************
confusion matrix for test data
```



Top 10 important features of both negative and positive class

In [309]:

```
#refered from github examples for dataframes
LR = LogisticRegression()
LR.fit(X_train_tf,y_train)
feat_log = LR.coef_

vectorizer_TF = TfidfVectorizer(min_df=10,max_features=10000,ngram_rang
e=(1,2))
vectorizer_TF.fit(X_train)
s = pd.DataFrame(feat_log.T,columns=['-ve'])
s['feature'] = vectorizer_TF.get_feature_names()
```

In [310]:

```
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' 'best' 'good' 'de
licious' 'love' 'perfect' 'loves' 'excellent'
 'nice' 'wonderful']
***********************************************************************
*****************************
Top 10  important features of negative class ['unfortunately' 'money'
'disappointing' 'awful' 'horrible' 'not worth'
 'terrible' 'disappointed' 'worst' 'not']
```

[5.3] Logistic Regression on AVG W2V

In [311]:
```
X=preprocessed_reviews
```

In [312]:
```
Y=final["Score"]
Y.shape
```

Out[312]:
```
(46071,)
```

In [313]:
```
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)


print(np.shape(X_train), y_train.shape)

print(np.shape(X_cv), y_cv.shape)
print(np.shape(X_test), y_test.shape)
```

```
(20680,) (20680,)
(10187,) (10187,)
```

```
(15204,) (15204,)
```

In [314]:
```
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [315]:
```
sent_of_train=[]
for sent in X_train:
    sent_of_train.append(sent.split())
```

In [316]:
```
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)
```

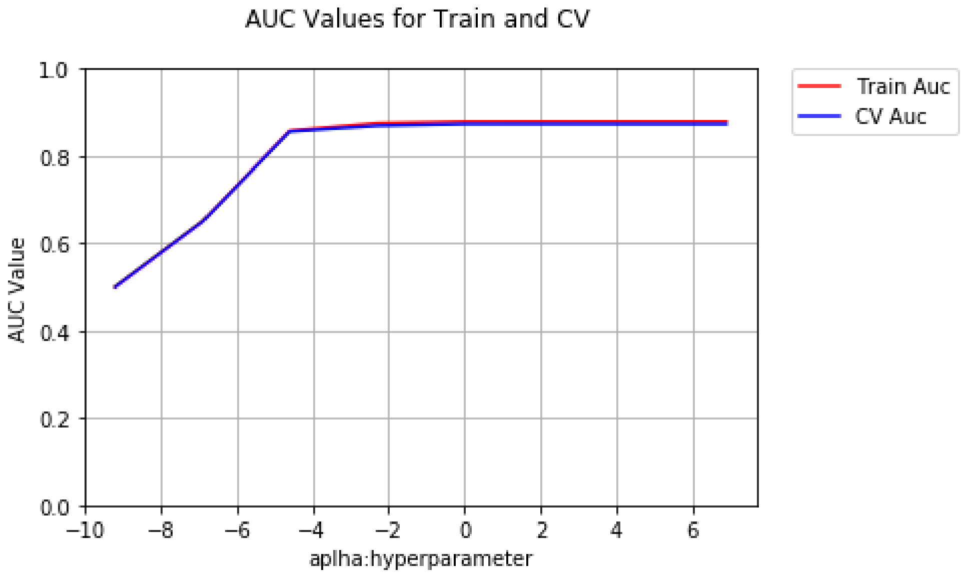
In [317]:
```
train_vectors = [];
for sent in sent_of_train:
    sent_vec = np.zeros(50)
    cnt_words =0;
    for word in sent: #
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    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)




# 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 [348]:
```
X_train_wv=train_vectors
X_cv_wv=cv_vectors
X_test_wv=test_vectors
```

Applying Logistic Regression with L1 regularization on Avg_w2v

In [320]:
```
log_opt(X_train_wv,X_cv_wv,"l1")
```

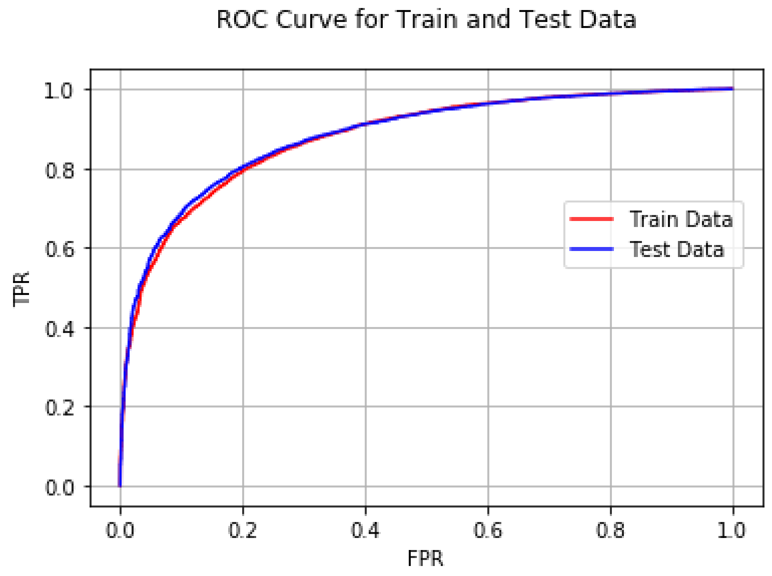



```
The optimal value of c =  10
```

In [349]:
```
LR=LogisticRegression(penalty="l1",C=10)
LR.fit(X_train_wv,y_train)


Y_test_pred_proba=LR.predict_proba(X_test_wv)
Y_train_pred_proba=LR.predict_proba(X_train_wv)

fpr,tpr,threshold=roc_curve(y_train,Y_train_pred_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,Y_test_pred_proba[:,1])
print("AUC value for test data ",roc_auc_score(y_test,Y_test_pred_proba
[:,1]))
print("AUC value for train data ",roc_auc_score(y_train,Y_train_pred_pr
oba[:,1]))
plt.plot(fpr,tpr,'r', label = 'Train Data')
plt.plot(fpr1,tpr1,'b', label = 'Test Data')
plt.legend(bbox_to_anchor=(1, 0.5), loc='lower right', borderaxespad=1)
```

```
plt.grid(True)
plt.title("ROC Curve for Train and Test Data\n")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```

```
AUC value for test data  0.8826927927343942
AUC value for train data  0.8786042473877529
```

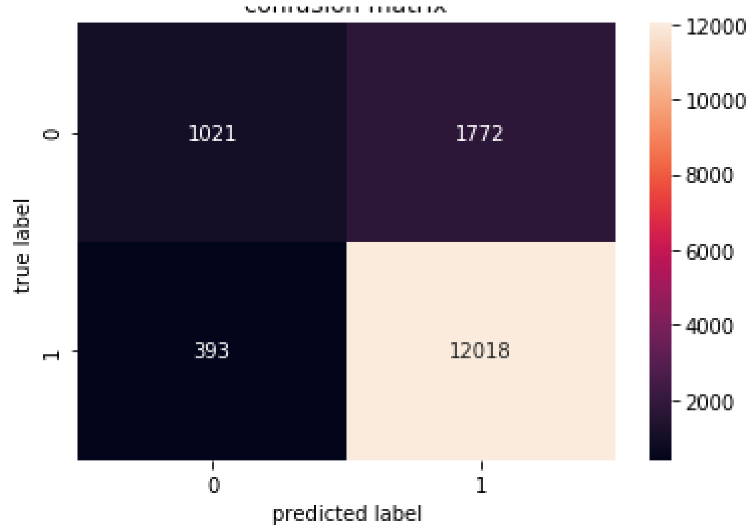


confusion matrix

In [350]:
```
confusion_matrix(X_train_wv,X_test_wv)
```

```
[[ 1043  1975]
 [  499 17163]]
[[ 1021  1772]
 [  393 12018]]
*************************************************************************
*****************************
confusion matrix for test data
```

confusion matrix



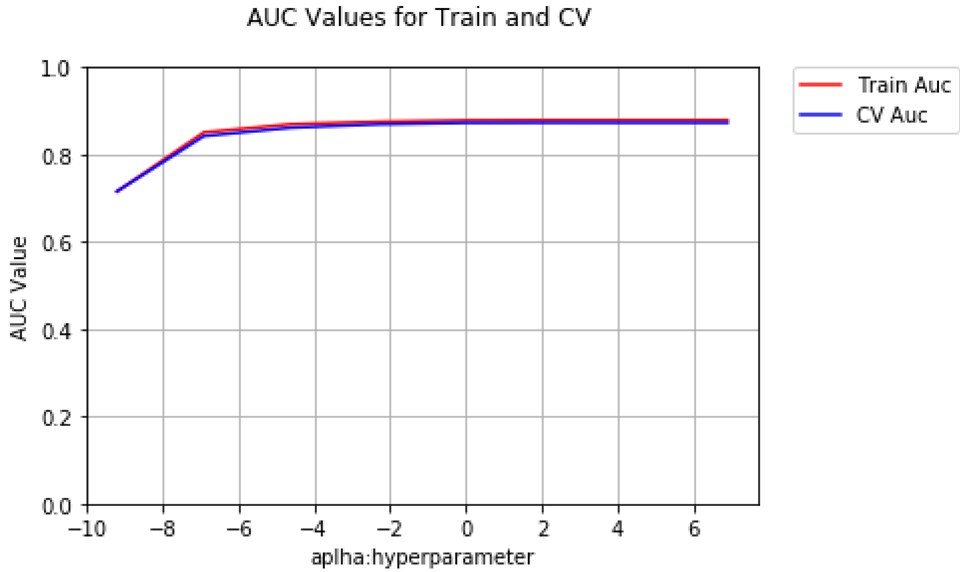
sparsity

In [351]:
```
w = LR.coef_
a = np.count_nonzero(w)
b = w.size
print("\n\nSparsity on weight vector obtained using L1 regularization i
s : ",(b-a)/b*100,"percent")
```

```
Sparsity on weight vector obtained using L1 regularization is :  0.0 pe
rcent
```

Applying Logistic Regression with L2 regularization on AvG_W2v

In [352]:
```
log_opt(X_train_wv,X_cv_wv,"l2")
```



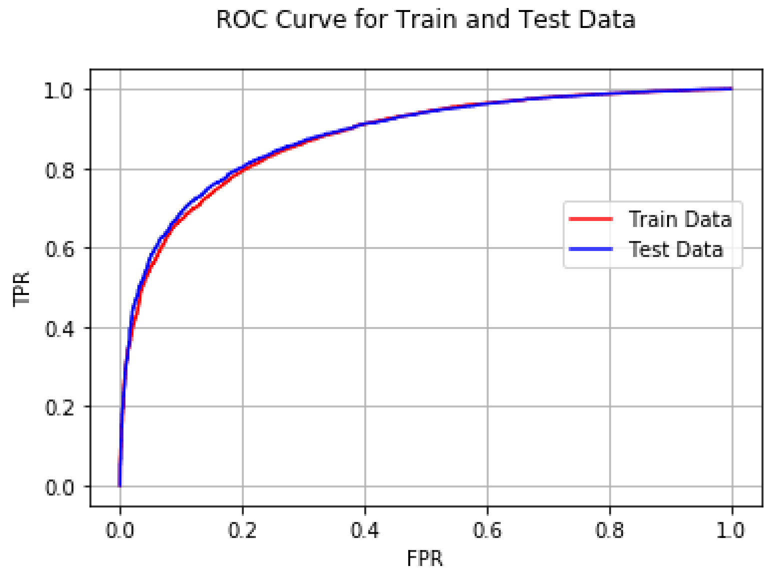

```
The optimal value of c =  10
```

In [354]:

```
LR=LogisticRegression(penalty="l2",C=10)
LR.fit(X_train_wv,y_train)


Y_test_pred_proba=LR.predict_proba(X_test_wv)
Y_train_pred_proba=LR.predict_proba(X_train_wv)

fpr,tpr,threshold=roc_curve(y_train,Y_train_pred_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,Y_test_pred_proba[:,1])
print("AUC value for test data ",roc_auc_score(y_test,Y_test_pred_proba
[:,1]))
print("AUC value for train data ",roc_auc_score(y_train,Y_train_pred_pr
oba[:,1]))
plt.plot(fpr,tpr,'r', label = 'Train Data')
plt.plot(fpr1,tpr1,'b', label = 'Test Data')
plt.legend(bbox_to_anchor=(1, 0.5), loc='lower right', borderaxespad=1)
plt.grid(True)
plt.title("ROC Curve for Train and Test Data\n")
plt.xlabel("FPR")
```

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

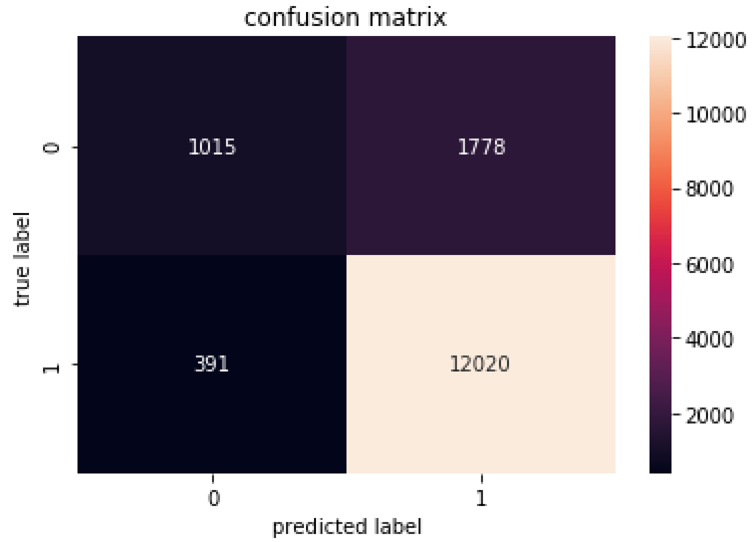
```
AUC value for test data  0.8828344674086658
AUC value for train data  0.8785970246538735
```



confusion matrix on Avg_w2v(L2)

In [355]:
```
confusion_matrix(X_train_wv,X_test_wv)
```

```
[[ 1039  1979]
 [  498 17164]]
[[ 1015  1778]
 [  391 12020]]
*************************************************************************
*****************************
confusion matrix for test data
```



[5.4] Logistic Regression on TFIDF W2V

In [257]:
```
X=preprocessed_reviews
Y=final["Score"]
```

In [258]:
```
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)


print(np.shape(X_train), y_train.shape)

print(np.shape(X_cv), y_cv.shape)
print(np.shape(X_test), y_test.shape)
```

```
(20680,) (20680,)
```

```
(10187,) (10187,)
(15204,) (15204,)
```

In [259]:
```python
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [260]:
```python
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_train_vectors = []; # the tfidf-w2v for each sentence/review is stored 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/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_train_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 20680/20680 [07:14<00:00, 47.58it/s]
```

In [261]:

```
tfidf_cv_vectors = []; # the tfidf-w2v for each sentence/review is stored 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/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_cv_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████| 10187/10187 [03:49<00:00, 35.67it/s]
```

In [262]:

```
tfidf_test_vectors = []; # the tfidf-w2v for each sentence/review is stored 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/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
```

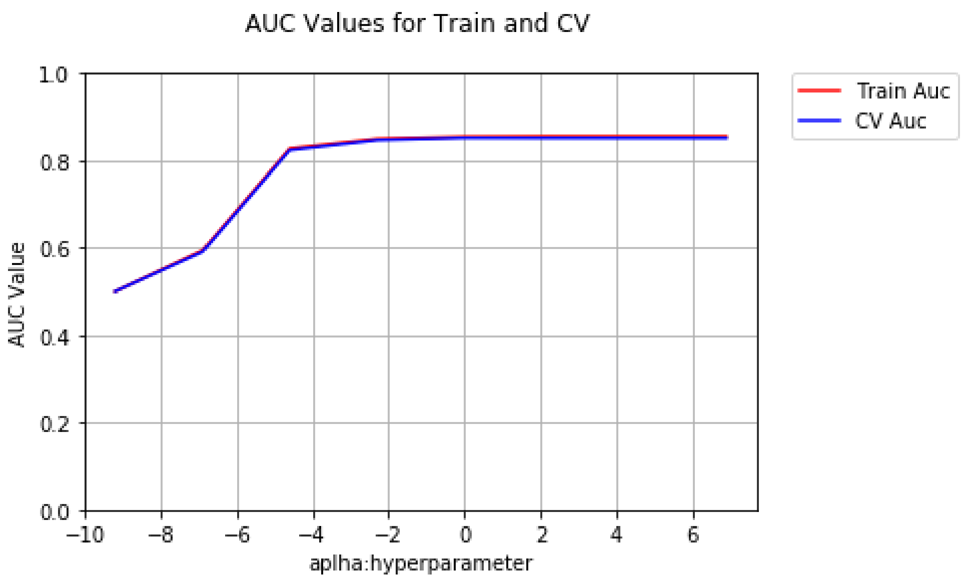
```
                tf_idf = dictionary[word]*(sent.count(word)/len(sent))
                sent_vec += (vec * tf_idf)
                weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_test_vectors.append(sent_vec)
    row += 1
```

```
100%|████████████████████████████████████| 15204/15204 [05:52<00:00, 4
3.10it/s]
```

In [340]:
```
X_train_tw=tfidf_train_vectors
X_cv_tw=tfidf_cv_vectors
X_test_tw=tfidf_test_vectors
```

Applying Logistic Regression with L1 regularization on TF_IDFW2V

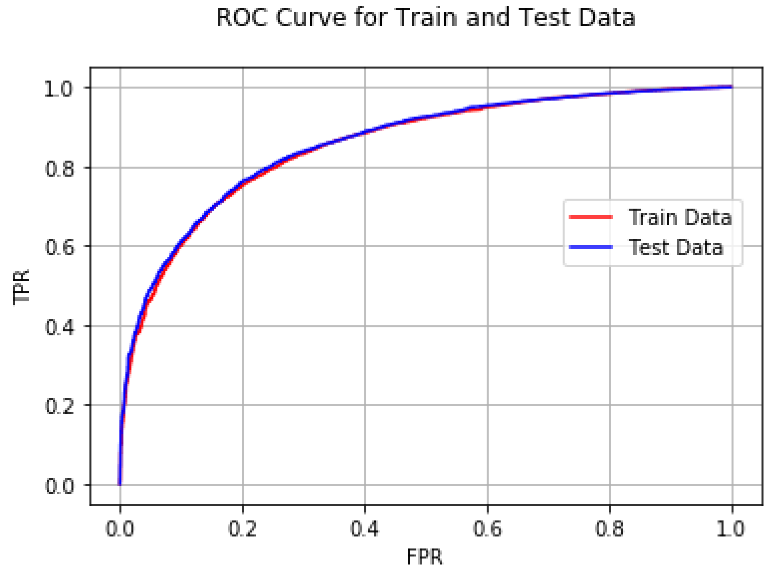
In [341]:
```
log_opt(X_train_tw,X_cv_tw,"l1")
```



```
The optimal value of c =  1
```

In [342]:
```python
LR=LogisticRegression(penalty="l1",C=1)
LR.fit(X_train_tw,y_train)


Y_test_pred_proba=LR.predict_proba(X_test_tw)
Y_train_pred_proba=LR.predict_proba(X_train_tw)

fpr,tpr,threshold=roc_curve(y_train,Y_train_pred_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,Y_test_pred_proba[:,1])
print("AUC value for test data ",roc_auc_score(y_test,Y_test_pred_proba
[:,1]))
print("AUC value for train data ",roc_auc_score(y_train,Y_train_pred_pr
oba[:,1]))
plt.plot(fpr,tpr,'r', label = 'Train Data')
plt.plot(fpr1,tpr1,'b', label = 'Test Data')
plt.legend(bbox_to_anchor=(1, 0.5), loc='lower right', borderaxespad=1)
plt.grid(True)
plt.title("ROC Curve for Train and Test Data\n")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.show()
```

```
AUC value for test data  0.8580005788727376
AUC value for train data  0.8537520207708567
```



sparsity

In [343]:
```
w = LR.coef_
a = np.count_nonzero(w)
b = w.size
print("\n\nSparsity on weight vector obtained using L1 regularization i
s : ",(b-a)/b*100,"percent")
```

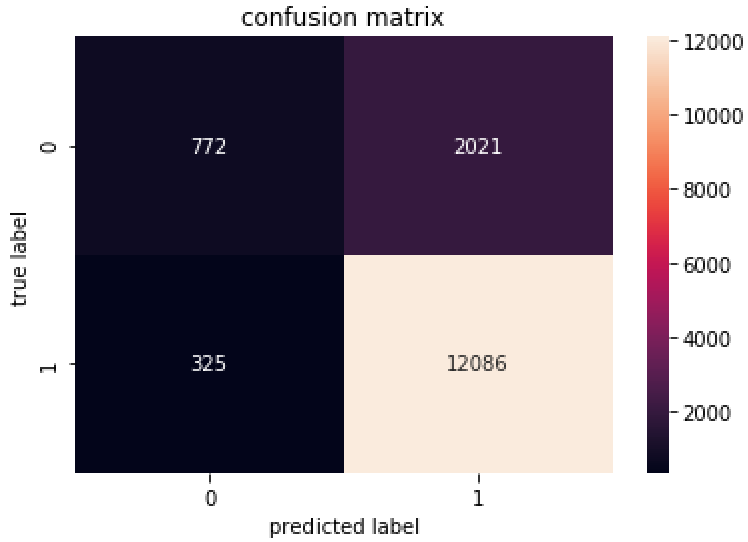
```
Sparsity on weight vector obtained using L1 regularization is :  8.0 pe
rcent
```

confusion matrix

In [344]:
```
confusion_matrix(X_train_tw,X_test_tw)
```

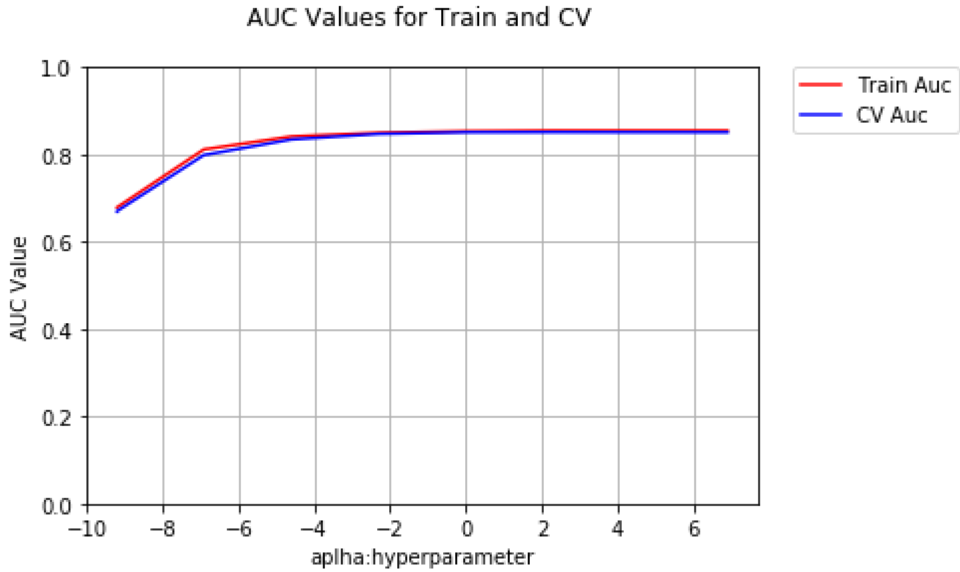
```
[[  795  2223]
 [  430 17232]]
```

```
[[  772  2021]
 [  325 12086]]
************************************************************************
*****************************
confusion matrix for test data
```



Applying Logistic Regression with L2 regularization on TF_IDFW2V

In [345]:
```
log_opt(X_train_tw,X_cv_tw,"l2")
```

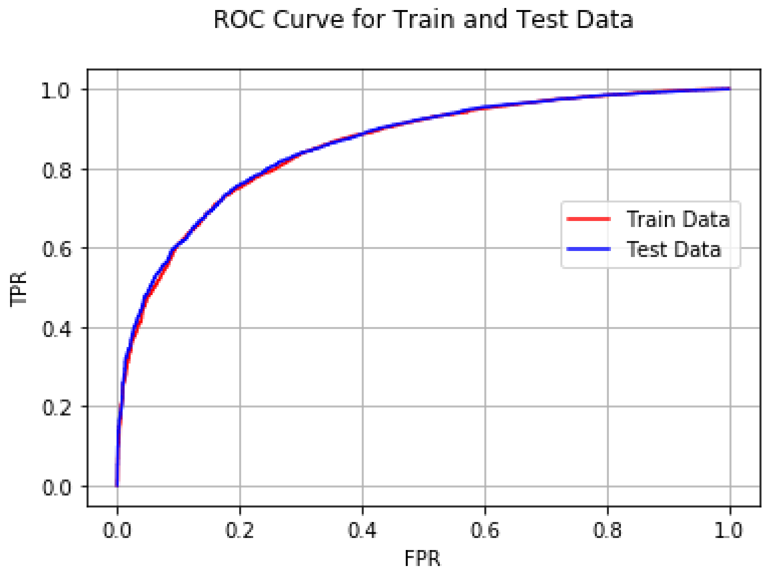



```
The optimal value of c =  10
```

In [346]:
```
LR=LogisticRegression(penalty="l2",C=10)
LR.fit(X_train_tw,y_train)


Y_test_pred_proba=LR.predict_proba(X_test_tw)
Y_train_pred_proba=LR.predict_proba(X_train_tw)

fpr,tpr,threshold=roc_curve(y_train,Y_train_pred_proba[:,1])
fpr1,tpr1,threshold1=roc_curve(y_test,Y_test_pred_proba[:,1])
print("AUC value for test data ",roc_auc_score(y_test,Y_test_pred_proba
[:,1]))
print("AUC value for train data ",roc_auc_score(y_train,Y_train_pred_pr
oba[:,1]))
plt.plot(fpr,tpr,'r', label = 'Train Data')
plt.plot(fpr1,tpr1,'b', label = 'Test Data')
plt.legend(bbox_to_anchor=(1, 0.5), loc='lower right', borderaxespad=1)
plt.grid(True)
plt.title("ROC Curve for Train and Test Data\n")
plt.xlabel("FPR")
```

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

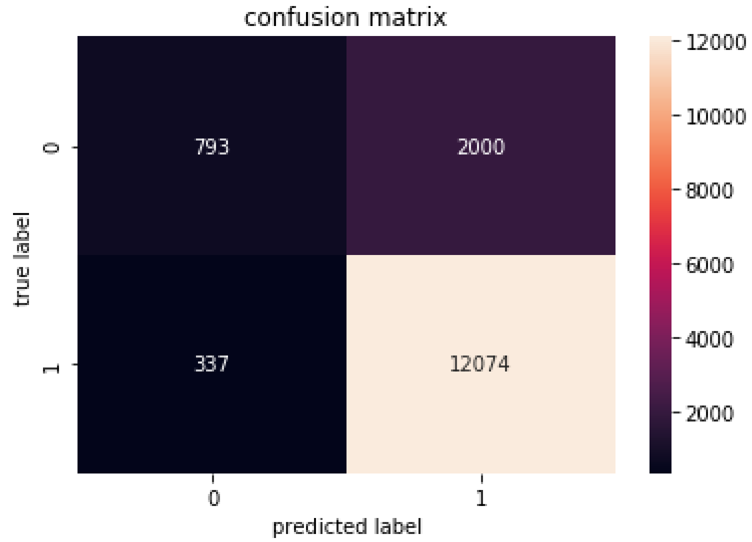
```
AUC value for test data  0.8572815027312403
AUC value for train data  0.8545176305620772
```



confusion matrix

In [347]:
```
confusion_matrix(X_train_tw,X_test_tw)
```

```
[[  803  2215]
 [  441 17221]]
[[  793  2000]
 [  337 12074]]
*************************************************************************
*****************************
confusion matrix for test data
```



CONCLUSION

In [339]:
```
print("l1 REGULARIZATION ")
from tabulate import tabulate
print(tabulate ([['BOW(l1)', 1, 93,77.8],['TF-IDF(l1)',10,94,69.55],['A
VG-W2V(L1)',10,88,0] , ['TFIDF-W2V(L1)',1,85,8]],    headers=['Vectoriz
er(regularization)', 'best_c','AUC_test','sparsity']))
```

```
l1 REGULARIZATION
Vectorizer(regularization)      best_c    AUC_test    sparsity
----------------------------  --------  ----------  ----------
BOW(l1)                              1          93       77.8
TF-IDF(l1)                          10          94       69.55
AVG-W2V(L1)                         10          88        0
TFIDF-W2V(L1)                        1          85        8
```

In [338]:
```
print("L2 REGULARIZATION ")
from tabulate import tabulate
print(tabulate ([['BOW(l2)', 0.1, 94],['TF-IDF(l2)',10,95],['AVG-W2V(L
```

```
2)',10,88] , ['TFIDF-W2V(L2)',10,85.72]],    headers=['Vectorizer(regul
arization)', 'best_c','AUC_test']))
```

```
L2 REGULARIZATION
Vectorizer(regularization)      best_c    AUC_test
----------------------------  --------  ----------
BOW(l2)                            0.1       94
TF-IDF(l2)                        10         95
AVG-W2V(L2)                       10         88
TFIDF-W2V(L2)                     10         85.72
```

1. logistic regression is faster than naive bayes and knn model.
2. If we compared all other model means TF-IDF is best model.
3. I only consider only 50k points and taking more datapoints we can improve the accuracy .