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 [0]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

In [78]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remoun
t, call drive.mount("/content/drive", force_remount=True).
```

In [79]:

```
# using SQLite Table to read data.
con = sqlite3.connect('drive/My Drive/Colab Notebooks/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[79]:

	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 [0]:

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

In [81]:

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

```
(80668, 7)
```

Out[81]:

	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 [82]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[82]:

	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 [83]:
```
display['COUNT(*)'].sum()
```

Out[83]: 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 [84]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[84]:

	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 [0]:
```
#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 [86]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape
```

Out[86]:
```
(87775, 10)
```

In [87]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[87]:
```
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 [88]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[88]:

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

In [0]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [90]:
```
#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[90]:
```
1    73592
0    14181
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 [91]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)
```

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the
candy has little taste to it.  Very little of the 2 lbs that I bought w
ere eaten and I threw the rest away.  I would not buy the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil sme
ll. So if you are afraid of the fishy smell, don't get it. But I think
my dog likes it because of the smell. These treats are really small in
size. They are great for training. You can give your dog several of the
se without worrying about him over eating. Amazon's price was much more
reasonable than any other retailer. You can buy a 1 pound bag on Amazon
for almost the same price as a 6 ounce bag at other retailers. It's def
initely worth it to buy a big bag if your dog eats them a lot.
==================================================
```

In [92]:

```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)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [93]:

```
# 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)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the
```

```
candy has little taste to it.  Very little of the 2 lbs that I bought w
ere eaten and I threw the rest away.  I would not buy the candy again.
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil sme
ll. So if you are afraid of the fishy smell, don't get it. But I think
my dog likes it because of the smell. These treats are really small in
size. They are great for training. You can give your dog several of the
se without worrying about him over eating. Amazon's price was much more
reasonable than any other retailer. You can buy a 1 pound bag on Amazon
for almost the same price as a 6 ounce bag at other retailers. It's def
initely worth it to buy a big bag if your dog eats them a lot.
```

In [0]:

```
# 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 [95]:

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

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

In [96]:
```
#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)
```

```
My dogs loves this chicken but its a product from China, so we wont be
buying it anymore.  Its very hard to find any chicken products made in
the USA but they are out there, but this one isnt.  Its too bad too bec
ause its a good product but I wont take any chances till they know what
is going on with the china imports.
```

In [97]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

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

In [0]:
```
# 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 [99]:

```
# 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())
```

```
  0%|          | 0/87773 [00:00<?, ?it/s]
```

```
0%|          | 245/87773 [00:00<00:35, 2449.75it/s]

1%|          | 449/87773 [00:00<00:37, 2309.36it/s]

1%|          | 701/87773 [00:00<00:36, 2368.75it/s]

1%|          | 949/87773 [00:00<00:36, 2400.83it/s]

1%|▏         | 1199/87773 [00:00<00:35, 2428.91it/s]

2%|▏         | 1445/87773 [00:00<00:35, 2436.94it/s]

2%|▏         | 1697/87773 [00:00<00:35, 2459.07it/s]

2%|▏         | 1957/87773 [00:00<00:34, 2480.67it/s]

3%|▎         | 2219/87773 [00:00<00:33, 2519.00it/s]

3%|▎         | 2490/87773 [00:01<00:33, 2573.16it/s]
```

```
3%|▎         | 2755/87773 [00:01<00:32, 2594.24it/s]

3%|▎         | 3011/87773 [00:01<00:33, 2561.85it/s]

4%|▎         | 3280/87773 [00:01<00:32, 2597.81it/s]

4%|▍         | 3538/87773 [00:01<00:33, 2523.11it/s]

4%|▍         | 3793/87773 [00:01<00:33, 2529.06it/s]

5%|▍         | 4046/87773 [00:01<00:33, 2468.21it/s]

5%|▍         | 4293/87773 [00:01<00:33, 2466.65it/s]

5%|▌         | 4554/87773 [00:01<00:33, 2506.38it/s]

5%|▌         | 4821/87773 [00:01<00:32, 2552.57it/s]

6%|▌         | 5077/87773 [00:02<00:32, 2545.68it/s]
```

```
6%|▌         | 5339/87773 [00:02<00:32, 2565.04it/s]
6%|▋         | 5611/87773 [00:02<00:31, 2608.67it/s]
7%|▋         | 5873/87773 [00:02<00:31, 2577.34it/s]
7%|▋         | 6132/87773 [00:02<00:32, 2516.30it/s]
7%|▋         | 6385/87773 [00:02<00:33, 2411.04it/s]
8%|▊         | 6639/87773 [00:02<00:33, 2444.80it/s]
8%|▊         | 6885/87773 [00:02<00:33, 2433.42it/s]
8%|▊         | 7139/87773 [00:02<00:32, 2463.59it/s]
8%|▊         | 7399/87773 [00:02<00:32, 2502.57it/s]
9%|▊         | 7668/87773 [00:03<00:31, 2553.58it/s]
```

```
  9%|▉         | 7932/87773 [00:03<00:30, 2577.25it/s]

  9%|▉         | 8191/87773 [00:03<00:31, 2554.80it/s]

 10%|▉         | 8447/87773 [00:03<00:31, 2545.45it/s]

 10%|▉         | 8702/87773 [00:03<00:31, 2536.57it/s]

 10%|█         | 8956/87773 [00:03<00:31, 2501.94it/s]

 10%|█         | 9214/87773 [00:03<00:31, 2523.82it/s]

 11%|█         | 9467/87773 [00:03<00:31, 2489.85it/s]

 11%|█         | 9721/87773 [00:03<00:31, 2502.13it/s]

 11%|█▏        | 9972/87773 [00:03<00:31, 2484.92it/s]
```

```
12%|█         | 10226/87773 [00:04<00:31, 2498.75it/s]

12%|█         | 10482/87773 [00:04<00:30, 2514.68it/s]

12%|█         | 10734/87773 [00:04<00:30, 2501.25it/s]

13%|█         | 10985/87773 [00:04<00:31, 2476.81it/s]

13%|█         | 11247/87773 [00:04<00:30, 2517.33it/s]

13%|█         | 11516/87773 [00:04<00:29, 2566.43it/s]

13%|█         | 11782/87773 [00:04<00:29, 2592.70it/s]

14%|█         | 12042/87773 [00:04<00:29, 2581.16it/s]

14%|█         | 12301/87773 [00:04<00:29, 2542.74it/s]

14%|█         | 12556/87773 [00:04<00:29, 2514.42it/s]
```

```
15%|█         | 12808/87773 [00:05<00:29, 2502.69it/s]

15%|█         | 13059/87773 [00:05<00:29, 2497.46it/s]

15%|█         | 13309/87773 [00:05<00:30, 2459.93it/s]

15%|█         | 13556/87773 [00:05<00:31, 2350.04it/s]

16%|█         | 13805/87773 [00:05<00:30, 2389.66it/s]

16%|█         | 14061/87773 [00:05<00:30, 2437.88it/s]

16%|█         | 14325/87773 [00:05<00:29, 2494.57it/s]

17%|█         | 14576/87773 [00:05<00:29, 2495.31it/s]

17%|█         | 14845/87773 [00:05<00:28, 2549.84it/s]

17%|█         | 15101/87773 [00:06<00:28, 2547.46it/s]
```

```
17%|█         | 15357/87773 [00:06<00:28, 2512.30it/s]

18%|█         | 15609/87773 [00:06<00:29, 2485.48it/s]

18%|█         | 15858/87773 [00:06<00:29, 2470.99it/s]

18%|█         | 16111/87773 [00:06<00:28, 2488.38it/s]

19%|█         | 16376/87773 [00:06<00:28, 2532.75it/s]

19%|█         | 16630/87773 [00:06<00:28, 2518.54it/s]

19%|█         | 16886/87773 [00:06<00:28, 2528.70it/s]

20%|█         | 17140/87773 [00:06<00:28, 2506.98it/s]

20%|█         | 17401/87773 [00:06<00:27, 2535.79it/s]

20%|█         | 17659/87773 [00:07<00:27, 2548.45it/s]
```

```
20%|██        | 17924/87773 [00:07<00:27, 2574.89it/s]

21%|██        | 18183/87773 [00:07<00:26, 2579.06it/s]

21%|██        | 18446/87773 [00:07<00:26, 2593.77it/s]

21%|██        | 18721/87773 [00:07<00:26, 2636.91it/s]

22%|██        | 18995/87773 [00:07<00:25, 2666.39it/s]

22%|██        | 19262/87773 [00:07<00:26, 2617.96it/s]

22%|██        | 19525/87773 [00:07<00:26, 2599.21it/s]

23%|██        | 19787/87773 [00:07<00:26, 2603.13it/s]

23%|██        | 20048/87773 [00:07<00:26, 2568.60it/s]
```

```
23%|██         | 20306/87773 [00:08<00:27, 2481.06it/s]

23%|██         | 20555/87773 [00:08<00:27, 2407.13it/s]

24%|██         | 20797/87773 [00:08<00:27, 2395.27it/s]

24%|██         | 21038/87773 [00:08<00:28, 2380.05it/s]

24%|██         | 21277/87773 [00:08<00:28, 2366.85it/s]

25%|██         | 21520/87773 [00:08<00:27, 2384.72it/s]

25%|██         | 21765/87773 [00:08<00:27, 2402.18it/s]

25%|██         | 22020/87773 [00:08<00:26, 2441.61it/s]

25%|██         | 22265/87773 [00:08<00:26, 2440.02it/s]

26%|██         | 22523/87773 [00:08<00:26, 2479.19it/s]
```

```
26%|██▌       | 22787/87773 [00:09<00:25, 2524.25it/s]

26%|██▋       | 23043/87773 [00:09<00:25, 2534.23it/s]

27%|██▋       | 23297/87773 [00:09<00:26, 2454.40it/s]

27%|██▋       | 23558/87773 [00:09<00:25, 2497.74it/s]

27%|██▋       | 23813/87773 [00:09<00:25, 2508.52it/s]

27%|██▋       | 24065/87773 [00:09<00:25, 2510.26it/s]

28%|██▊       | 24319/87773 [00:09<00:25, 2516.38it/s]

28%|██▊       | 24572/87773 [00:09<00:25, 2519.37it/s]

28%|██▊       | 24825/87773 [00:09<00:24, 2519.85it/s]

29%|██▊       | 25078/87773 [00:10<00:24, 2518.50it/s]
```

```
29%|███       | 25340/87773 [00:10<00:24, 2547.70it/s]

29%|███       | 25603/87773 [00:10<00:24, 2571.36it/s]

29%|███       | 25861/87773 [00:10<00:24, 2524.43it/s]

30%|███       | 26114/87773 [00:10<00:24, 2519.38it/s]

30%|███       | 26367/87773 [00:10<00:24, 2471.82it/s]

30%|███       | 26615/87773 [00:10<00:25, 2358.49it/s]

31%|███       | 26853/87773 [00:10<00:25, 2358.52it/s]

31%|███       | 27114/87773 [00:10<00:24, 2427.66it/s]

31%|███       | 27360/87773 [00:10<00:24, 2437.26it/s]

31%|███▏      | 27610/87773 [00:11<00:24, 2455.62it/s]
```

```
32%|███      | 27865/87773 [00:11<00:24, 2480.80it/s]
32%|███      | 28114/87773 [00:11<00:24, 2455.94it/s]
32%|███      | 28360/87773 [00:11<00:24, 2403.11it/s]
33%|███      | 28617/87773 [00:11<00:24, 2448.84it/s]
33%|███      | 28873/87773 [00:11<00:23, 2480.75it/s]
33%|███      | 29141/87773 [00:11<00:23, 2535.23it/s]
33%|███      | 29396/87773 [00:11<00:23, 2466.64it/s]
34%|███      | 29662/87773 [00:11<00:23, 2519.24it/s]
34%|███      | 29919/87773 [00:11<00:22, 2532.99it/s]
```

```
34%|███         | 30173/87773 [00:12<00:22, 2507.56it/s]

35%|███         | 30425/87773 [00:12<00:23, 2467.26it/s]

35%|███         | 30677/87773 [00:12<00:22, 2482.58it/s]

35%|███         | 30926/87773 [00:12<00:23, 2388.37it/s]

36%|███         | 31197/87773 [00:12<00:22, 2476.34it/s]

36%|███         | 31447/87773 [00:12<00:22, 2473.78it/s]

36%|███         | 31700/87773 [00:12<00:22, 2488.59it/s]

36%|███         | 31950/87773 [00:12<00:22, 2490.84it/s]

37%|███         | 32202/87773 [00:12<00:22, 2498.57it/s]

37%|███         | 32455/87773 [00:12<00:22, 2506.14it/s]
```

```
37%|███▋      | 32706/87773 [00:13<00:22, 2468.08it/s]

38%|███▊      | 32954/87773 [00:13<00:22, 2415.96it/s]

38%|███▊      | 33197/87773 [00:13<00:23, 2364.07it/s]

38%|███▊      | 33441/87773 [00:13<00:22, 2384.91it/s]

38%|███▊      | 33686/87773 [00:13<00:22, 2403.05it/s]

39%|███▊      | 33946/87773 [00:13<00:21, 2458.40it/s]

39%|███▉      | 34193/87773 [00:13<00:22, 2398.86it/s]

39%|███▉      | 34434/87773 [00:13<00:23, 2293.06it/s]

40%|███▉      | 34688/87773 [00:13<00:22, 2359.95it/s]

40%|███▉      | 34942/87773 [00:14<00:21, 2409.17it/s]
```

```
 40%|████      | 35189/87773 [00:14<00:21, 2426.24it/s]

 40%|████      | 35433/87773 [00:14<00:21, 2428.63it/s]

 41%|████      | 35702/87773 [00:14<00:20, 2501.45it/s]

 41%|████      | 35956/87773 [00:14<00:20, 2511.35it/s]

 41%|████▏     | 36222/87773 [00:14<00:20, 2553.96it/s]

 42%|████▏     | 36495/87773 [00:14<00:19, 2603.56it/s]

 42%|████▏     | 36757/87773 [00:14<00:19, 2563.25it/s]

 42%|████▏     | 37014/87773 [00:14<00:19, 2552.98it/s]

 42%|████▏     | 37270/87773 [00:14<00:20, 2477.13it/s]

 43%|████▎     | 37519/87773 [00:15<00:20, 2470.14it/s]
```

```
43%|████▎     | 37793/87773 [00:15<00:19, 2544.82it/s]

43%|████▎     | 38049/87773 [00:15<00:19, 2547.27it/s]

44%|████▍     | 38311/87773 [00:15<00:19, 2567.82it/s]

44%|████▍     | 38569/87773 [00:15<00:19, 2521.82it/s]

44%|████▍     | 38831/87773 [00:15<00:19, 2548.32it/s]

45%|████▍     | 39087/87773 [00:15<00:19, 2511.95it/s]

45%|████▍     | 39345/87773 [00:15<00:19, 2531.17it/s]

45%|████▌     | 39600/87773 [00:15<00:19, 2534.19it/s]

45%|████▌     | 39859/87773 [00:15<00:18, 2548.28it/s]
```

```
46%|████▌     | 40124/87773 [00:16<00:18, 2576.24it/s]

46%|████▌     | 40382/87773 [00:16<00:18, 2562.29it/s]

46%|████▋     | 40639/87773 [00:16<00:18, 2524.74it/s]

47%|████▋     | 40892/87773 [00:16<00:18, 2502.65it/s]

47%|████▋     | 41143/87773 [00:16<00:18, 2501.58it/s]

47%|████▋     | 41394/87773 [00:16<00:19, 2402.77it/s]

47%|████▋     | 41636/87773 [00:16<00:19, 2377.18it/s]

48%|████▊     | 41893/87773 [00:16<00:18, 2431.74it/s]

48%|████▊     | 42145/87773 [00:16<00:18, 2455.71it/s]

48%|████▊     | 42392/87773 [00:16<00:18, 2449.63it/s]
```

```
49%|████▉     | 42638/87773 [00:17<00:18, 2449.16it/s]

49%|████▉     | 42901/87773 [00:17<00:17, 2500.46it/s]

49%|████▉     | 43161/87773 [00:17<00:17, 2529.20it/s]

49%|████▉     | 43421/87773 [00:17<00:17, 2548.47it/s]

50%|████▉     | 43677/87773 [00:17<00:17, 2547.73it/s]

50%|█████     | 43933/87773 [00:17<00:17, 2471.03it/s]

50%|█████     | 44200/87773 [00:17<00:17, 2526.52it/s]

51%|█████     | 44454/87773 [00:17<00:17, 2520.36it/s]

51%|█████     | 44723/87773 [00:17<00:16, 2567.65it/s]

51%|█████     | 44981/87773 [00:18<00:16, 2533.13it/s]
```

```
52%|█████▏    | 45235/87773 [00:18<00:16, 2507.74it/s]

52%|█████▏    | 45487/87773 [00:18<00:18, 2327.14it/s]

52%|█████▏    | 45723/87773 [00:18<00:18, 2266.36it/s]

52%|█████▏    | 45953/87773 [00:18<00:18, 2222.30it/s]

53%|█████▎    | 46187/87773 [00:18<00:18, 2256.21it/s]

53%|█████▎    | 46431/87773 [00:18<00:17, 2306.44it/s]

53%|█████▎    | 46697/87773 [00:18<00:17, 2400.59it/s]

53%|█████▎    | 46939/87773 [00:18<00:17, 2316.57it/s]

54%|█████▍    | 47182/87773 [00:18<00:17, 2348.13it/s]

54%|█████▍    | 47446/87773 [00:19<00:16, 2428.01it/s]
```

```
54%|█████     | 47699/87773 [00:19<00:16, 2457.42it/s]

55%|█████     | 47960/87773 [00:19<00:15, 2497.34it/s]

55%|█████     | 48226/87773 [00:19<00:15, 2540.56it/s]

55%|█████     | 48481/87773 [00:19<00:16, 2379.98it/s]

56%|█████     | 48732/87773 [00:19<00:16, 2414.52it/s]

56%|█████     | 48976/87773 [00:19<00:16, 2361.05it/s]

56%|█████     | 49216/87773 [00:19<00:16, 2370.64it/s]

56%|█████     | 49463/87773 [00:19<00:15, 2395.90it/s]

57%|█████     | 49712/87773 [00:20<00:15, 2423.17it/s]
```

```
57%|█████▋    | 49975/87773 [00:20<00:15, 2478.51it/s]

57%|█████▋    | 50229/87773 [00:20<00:15, 2495.52it/s]

58%|█████▊    | 50486/87773 [00:20<00:14, 2516.72it/s]

58%|█████▊    | 50772/87773 [00:20<00:14, 2608.83it/s]

58%|█████▊    | 51047/87773 [00:20<00:13, 2647.75it/s]

58%|█████▊    | 51313/87773 [00:20<00:13, 2627.11it/s]

59%|█████▉    | 51583/87773 [00:20<00:13, 2646.22it/s]

59%|█████▉    | 51849/87773 [00:20<00:14, 2561.94it/s]

59%|█████▉    | 52107/87773 [00:20<00:13, 2565.72it/s]

60%|█████▉    | 52365/87773 [00:21<00:13, 2540.26it/s]
```

```
60%|██████    | 52620/87773 [00:21<00:14, 2474.30it/s]
60%|██████    | 52879/87773 [00:21<00:13, 2506.35it/s]
61%|██████    | 53131/87773 [00:21<00:14, 2467.03it/s]
61%|██████    | 53379/87773 [00:21<00:13, 2461.29it/s]
61%|██████    | 53628/87773 [00:21<00:13, 2469.16it/s]
61%|██████▏   | 53878/87773 [00:21<00:13, 2477.91it/s]
62%|██████▏   | 54130/87773 [00:21<00:13, 2488.83it/s]
62%|██████▏   | 54380/87773 [00:21<00:13, 2462.78it/s]
62%|██████▏   | 54637/87773 [00:21<00:13, 2493.94it/s]
63%|██████▎   | 54887/87773 [00:22<00:14, 2346.23it/s]
```

```
63%|██████▎   | 55124/87773 [00:22<00:14, 2265.26it/s]

63%|██████▎   | 55364/87773 [00:22<00:14, 2303.48it/s]

63%|██████▎   | 55596/87773 [00:22<00:14, 2196.57it/s]

64%|██████▎   | 55818/87773 [00:22<00:15, 2087.53it/s]

64%|██████▍   | 56053/87773 [00:22<00:14, 2159.14it/s]

64%|██████▍   | 56272/87773 [00:22<00:14, 2143.71it/s]

64%|██████▍   | 56500/87773 [00:22<00:14, 2179.81it/s]

65%|██████▍   | 56720/87773 [00:22<00:14, 2165.02it/s]

65%|██████▍   | 56975/87773 [00:23<00:13, 2266.06it/s]

65%|██████▌   | 57216/87773 [00:23<00:13, 2306.06it/s]
```

```
65%|██████   | 57449/87773 [00:23<00:13, 2252.73it/s]

66%|██████   | 57689/87773 [00:23<00:13, 2293.80it/s]

66%|██████   | 57951/87773 [00:23<00:12, 2382.69it/s]

66%|██████   | 58191/87773 [00:23<00:12, 2368.24it/s]

67%|██████   | 58436/87773 [00:23<00:12, 2391.60it/s]

67%|██████   | 58688/87773 [00:23<00:11, 2426.72it/s]

67%|██████   | 58955/87773 [00:23<00:11, 2493.65it/s]

67%|██████   | 59206/87773 [00:23<00:11, 2489.22it/s]

68%|██████   | 59458/87773 [00:24<00:11, 2496.62it/s]
```

```
68%|███████   | 59714/87773 [00:24<00:11, 2514.86it/s]

68%|███████   | 59966/87773 [00:24<00:11, 2497.21it/s]

69%|███████   | 60227/87773 [00:24<00:10, 2529.84it/s]

69%|███████   | 60481/87773 [00:24<00:10, 2511.67it/s]

69%|███████   | 60733/87773 [00:24<00:10, 2460.34it/s]

69%|███████   | 60995/87773 [00:24<00:10, 2504.07it/s]

70%|███████   | 61246/87773 [00:24<00:10, 2504.47it/s]

70%|███████   | 61514/87773 [00:24<00:10, 2554.61it/s]

70%|███████   | 61773/87773 [00:24<00:10, 2563.49it/s]

71%|███████   | 62031/87773 [00:25<00:10, 2567.80it/s]
```

```
71%|███████   | 62291/87773 [00:25<00:09, 2577.24it/s]

71%|███████   | 62549/87773 [00:25<00:09, 2554.17it/s]

72%|███████   | 62805/87773 [00:25<00:09, 2542.16it/s]

72%|███████   | 63060/87773 [00:25<00:09, 2520.44it/s]

72%|███████   | 63317/87773 [00:25<00:09, 2532.10it/s]

72%|███████   | 63571/87773 [00:25<00:09, 2504.63it/s]

73%|███████   | 63822/87773 [00:25<00:10, 2394.93it/s]

73%|███████   | 64063/87773 [00:25<00:09, 2387.56it/s]

73%|███████   | 64303/87773 [00:25<00:10, 2343.20it/s]

74%|███████   | 64562/87773 [00:26<00:09, 2411.20it/s]
```

```
74%|███████▍  | 64824/87773 [00:26<00:09, 2469.83it/s]
74%|███████▍  | 65082/87773 [00:26<00:09, 2499.87it/s]
74%|███████▍  | 65333/87773 [00:26<00:09, 2452.42it/s]
75%|███████▌  | 65597/87773 [00:26<00:08, 2504.13it/s]
75%|███████▌  | 65864/87773 [00:26<00:08, 2546.59it/s]
75%|███████▌  | 66120/87773 [00:26<00:08, 2511.57it/s]
76%|███████▌  | 66379/87773 [00:26<00:08, 2532.43it/s]
76%|███████▌  | 66644/87773 [00:26<00:08, 2565.26it/s]
76%|███████▌  | 66907/87773 [00:27<00:08, 2578.24it/s]
77%|███████▋  | 67166/87773 [00:27<00:08, 2431.00it/s]
```

```
77%|███████▋  | 67412/87773 [00:27<00:08, 2382.98it/s]

77%|███████▋  | 67661/87773 [00:27<00:08, 2410.51it/s]

77%|███████▋  | 67912/87773 [00:27<00:08, 2433.83it/s]

78%|███████▊  | 68164/87773 [00:27<00:07, 2456.32it/s]

78%|███████▊  | 68434/87773 [00:27<00:07, 2522.80it/s]

78%|███████▊  | 68688/87773 [00:27<00:08, 2343.83it/s]

79%|███████▊  | 68934/87773 [00:27<00:07, 2376.37it/s]

79%|███████▉  | 69192/87773 [00:27<00:07, 2430.63it/s]

79%|███████▉  | 69447/87773 [00:28<00:07, 2462.34it/s]
```

```
79%|████████  | 69695/87773 [00:28<00:07, 2448.64it/s]

80%|████████  | 69969/87773 [00:28<00:07, 2529.01it/s]

80%|████████  | 70230/87773 [00:28<00:06, 2552.08it/s]

80%|████████  | 70487/87773 [00:28<00:07, 2445.59it/s]

81%|████████  | 70734/87773 [00:28<00:07, 2368.08it/s]

81%|████████  | 70973/87773 [00:28<00:07, 2347.64it/s]

81%|████████  | 71220/87773 [00:28<00:06, 2380.34it/s]

81%|████████  | 71476/87773 [00:28<00:06, 2430.83it/s]

82%|████████  | 71721/87773 [00:29<00:06, 2375.75it/s]

82%|████████  | 71960/87773 [00:29<00:06, 2334.10it/s]
```

```
82%|████████▏ | 72195/87773 [00:29<00:06, 2279.50it/s]

83%|████████▎ | 72456/87773 [00:29<00:06, 2369.39it/s]

83%|████████▎ | 72714/87773 [00:29<00:06, 2427.24it/s]

83%|████████▎ | 72959/87773 [00:29<00:06, 2371.90it/s]

83%|████████▎ | 73204/87773 [00:29<00:06, 2391.69it/s]

84%|████████▎ | 73445/87773 [00:29<00:06, 2379.99it/s]

84%|████████▍ | 73695/87773 [00:29<00:05, 2412.24it/s]

84%|████████▍ | 73937/87773 [00:29<00:06, 2208.74it/s]

85%|████████▍ | 74194/87773 [00:30<00:05, 2304.19it/s]

85%|████████▍ | 74429/87773 [00:30<00:05, 2305.61it/s]
```

```
85%|████████▌ | 74679/87773 [00:30<00:05, 2360.22it/s]
85%|████████▌ | 74937/87773 [00:30<00:05, 2418.98it/s]
86%|████████▌ | 75181/87773 [00:30<00:05, 2407.64it/s]
86%|████████▌ | 75424/87773 [00:30<00:05, 2377.33it/s]
86%|████████▌ | 75663/87773 [00:30<00:05, 2267.13it/s]
86%|████████▋ | 75892/87773 [00:30<00:05, 2185.80it/s]
87%|████████▋ | 76113/87773 [00:30<00:05, 2139.49it/s]
87%|████████▋ | 76329/87773 [00:31<00:05, 2144.40it/s]
87%|████████▋ | 76564/87773 [00:31<00:05, 2201.31it/s]
88%|████████▊ | 76812/87773 [00:31<00:04, 2277.23it/s]
```

```
88%|████████▊ | 77066/87773 [00:31<00:04, 2347.71it/s]

88%|████████▊ | 77303/87773 [00:31<00:04, 2250.61it/s]

88%|████████▊ | 77553/87773 [00:31<00:04, 2318.76it/s]

89%|████████▊ | 77787/87773 [00:31<00:06, 1636.94it/s]

89%|████████▉ | 78017/87773 [00:31<00:05, 1790.66it/s]

89%|████████▉ | 78258/87773 [00:31<00:04, 1939.85it/s]

89%|████████▉ | 78519/87773 [00:32<00:04, 2101.70it/s]

90%|████████▉ | 78784/87773 [00:32<00:04, 2240.35it/s]

90%|█████████ | 79024/87773 [00:32<00:03, 2280.24it/s]
```

```
90%|█████████ | 79272/87773 [00:32<00:03, 2335.05it/s]

91%|█████████ | 79514/87773 [00:32<00:03, 2292.25it/s]

91%|█████████ | 79753/87773 [00:32<00:03, 2319.01it/s]

91%|█████████ | 79990/87773 [00:32<00:03, 2312.59it/s]

91%|█████████▏| 80236/87773 [00:32<00:03, 2353.92it/s]

92%|█████████▏| 80491/87773 [00:32<00:03, 2408.31it/s]

92%|█████████▏| 80734/87773 [00:32<00:02, 2409.55it/s]

92%|█████████▏| 80977/87773 [00:33<00:02, 2412.64it/s]

93%|█████████▎| 81220/87773 [00:33<00:02, 2306.08it/s]

93%|█████████▎| 81489/87773 [00:33<00:02, 2407.66it/s]
```

```
93%|█████████ | 81733/87773 [00:33<00:02, 2319.39it/s]

93%|█████████ | 81968/87773 [00:33<00:02, 2249.68it/s]

94%|█████████ | 82230/87773 [00:33<00:02, 2345.61it/s]

94%|█████████ | 82468/87773 [00:33<00:02, 2243.40it/s]

94%|█████████ | 82714/87773 [00:33<00:02, 2301.27it/s]

95%|█████████ | 82959/87773 [00:33<00:02, 2342.97it/s]

95%|█████████ | 83213/87773 [00:34<00:01, 2396.66it/s]

95%|█████████ | 83455/87773 [00:34<00:01, 2233.29it/s]

95%|█████████ | 83682/87773 [00:34<00:01, 2191.41it/s]

96%|█████████ | 83904/87773 [00:34<00:01, 2150.39it/s]
```

```
96%|█████████▌| 84164/87773 [00:34<00:01, 2267.07it/s]

96%|█████████▌| 84394/87773 [00:34<00:01, 2159.52it/s]

96%|█████████▋| 84621/87773 [00:34<00:01, 2186.40it/s]

97%|█████████▋| 84881/87773 [00:34<00:01, 2294.84it/s]

97%|█████████▋| 85128/87773 [00:34<00:01, 2343.12it/s]

97%|█████████▋| 85397/87773 [00:35<00:00, 2437.16it/s]

98%|█████████▊| 85677/87773 [00:35<00:00, 2534.15it/s]

98%|█████████▊| 85962/87773 [00:35<00:00, 2615.26it/s]

98%|█████████▊| 86227/87773 [00:35<00:00, 2573.66it/s]

99%|█████████▊| 86487/87773 [00:35<00:00, 2498.70it/s]
```

```
 99%|██████████| 86739/87773 [00:35<00:00, 2432.52it/s]

 99%|██████████| 86991/87773 [00:35<00:00, 2456.83it/s]

 99%|██████████| 87246/87773 [00:35<00:00, 2483.45it/s]

100%|██████████| 87496/87773 [00:35<00:00, 2409.70it/s]

100%|██████████| 87744/87773 [00:35<00:00, 2428.41it/s]

100%|██████████| 87773/87773 [00:35<00:00, 2440.00it/s]
```

In [100]:
```
preprocessed_reviews[1500]
```

Out[100]:
```
'way hot blood took bite jig lol'
```

[3.2] Preprocessing Review Summary

In [0]:
```
## Similartly you can do preprocessing for review summary also.
```

In [101]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
```

```
for sentance in tqdm(final['Summary'].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_summary.append(sentance.strip())
```

```
  0%|          | 0/87773 [00:00<?, ?it/s]

  0%|          | 364/87773 [00:00<00:24, 3639.96it/s]

  1%|          | 737/87773 [00:00<00:23, 3662.71it/s]

  1%|▏         | 1107/87773 [00:00<00:23, 3673.23it/s]

  2%|▏         | 1482/87773 [00:00<00:23, 3694.45it/s]

  2%|▏         | 1856/87773 [00:00<00:23, 3706.20it/s]

  3%|▎         | 2236/87773 [00:00<00:22, 3733.15it/s]
```

```
  3%|▎         | 2622/87773 [00:00<00:22, 3769.40it/s]
  3%|▎         | 2988/87773 [00:00<00:22, 3735.61it/s]
  4%|▍         | 3366/87773 [00:00<00:22, 3748.47it/s]
  4%|▍         | 3727/87773 [00:01<00:22, 3702.08it/s]
  5%|▍         | 4102/87773 [00:01<00:22, 3713.31it/s]
  5%|▌         | 4483/87773 [00:01<00:22, 3739.43it/s]
  6%|▌         | 4858/87773 [00:01<00:22, 3740.60it/s]
  6%|▌         | 5240/87773 [00:01<00:21, 3762.31it/s]
  6%|▋         | 5614/87773 [00:01<00:22, 3728.31it/s]
  7%|▋         | 5991/87773 [00:01<00:21, 3739.34it/s]
```

```
  7%|▋         | 6369/87773 [00:01<00:21, 3750.14it/s]
  8%|▊         | 6744/87773 [00:01<00:21, 3712.95it/s]
  8%|▊         | 7125/87773 [00:01<00:21, 3739.95it/s]
  9%|▊         | 7500/87773 [00:02<00:21, 3742.64it/s]
  9%|▉         | 7878/87773 [00:02<00:21, 3752.11it/s]
  9%|▉         | 8254/87773 [00:02<00:21, 3744.57it/s]
 10%|▉         | 8629/87773 [00:02<00:21, 3741.40it/s]
 10%|█         | 9004/87773 [00:02<00:21, 3742.72it/s]
 11%|█         | 9382/87773 [00:02<00:20, 3753.34it/s]
```

```
11%|█         | 9758/87773 [00:02<00:20, 3737.79it/s]

12%|█         | 10136/87773 [00:02<00:20, 3748.17it/s]

12%|█         | 10511/87773 [00:02<00:20, 3737.97it/s]

12%|█         | 10889/87773 [00:02<00:20, 3747.87it/s]

13%|█         | 11264/87773 [00:03<00:20, 3703.43it/s]

13%|█         | 11642/87773 [00:03<00:20, 3725.31it/s]

14%|█         | 12023/87773 [00:03<00:20, 3749.94it/s]

14%|█         | 12402/87773 [00:03<00:20, 3760.15it/s]

15%|█         | 12779/87773 [00:03<00:20, 3717.34it/s]

15%|█         | 13151/87773 [00:03<00:20, 3703.99it/s]
```

```
15%|█         | 13522/87773 [00:03<00:20, 3676.86it/s]

16%|█         | 13904/87773 [00:03<00:19, 3718.47it/s]

16%|█         | 14277/87773 [00:03<00:19, 3687.23it/s]

17%|█         | 14650/87773 [00:03<00:19, 3696.77it/s]

17%|█         | 15020/87773 [00:04<00:19, 3661.75it/s]

18%|█         | 15402/87773 [00:04<00:19, 3707.58it/s]

18%|█         | 15777/87773 [00:04<00:19, 3718.85it/s]

18%|█         | 16160/87773 [00:04<00:19, 3749.67it/s]

19%|█         | 16539/87773 [00:04<00:18, 3759.71it/s]

19%|█         | 16918/87773 [00:04<00:18, 3767.34it/s]
```

```
20%|██         | 17295/87773 [00:04<00:18, 3752.44it/s]
20%|██         | 17678/87773 [00:04<00:18, 3773.41it/s]
21%|██         | 18063/87773 [00:04<00:18, 3794.89it/s]
21%|██         | 18446/87773 [00:04<00:18, 3803.18it/s]
21%|██▏        | 18827/87773 [00:05<00:18, 3772.59it/s]
22%|██▏        | 19210/87773 [00:05<00:18, 3788.97it/s]
22%|██▏        | 19595/87773 [00:05<00:17, 3806.62it/s]
23%|██▎        | 19976/87773 [00:05<00:17, 3784.28it/s]
23%|██▎        | 20355/87773 [00:05<00:17, 3758.47it/s]
24%|██▎        | 20731/87773 [00:05<00:17, 3757.24it/s]
```

```
24%|██        | 21107/87773 [00:05<00:17, 3756.90it/s]

24%|██        | 21483/87773 [00:05<00:17, 3751.17it/s]

25%|██        | 21865/87773 [00:05<00:17, 3770.67it/s]

25%|██        | 22246/87773 [00:05<00:17, 3781.79it/s]

26%|██        | 22625/87773 [00:06<00:17, 3760.84it/s]

26%|██        | 23002/87773 [00:06<00:17, 3742.40it/s]

27%|██        | 23380/87773 [00:06<00:17, 3749.45it/s]

27%|██        | 23762/87773 [00:06<00:16, 3769.82it/s]

28%|██        | 24140/87773 [00:06<00:16, 3769.94it/s]
```

```
28%|███       | 24518/87773 [00:06<00:16, 3742.48it/s]

28%|███       | 24899/87773 [00:06<00:16, 3760.65it/s]

29%|███       | 25283/87773 [00:06<00:16, 3782.72it/s]

29%|███       | 25663/87773 [00:06<00:16, 3786.39it/s]

30%|███       | 26042/87773 [00:06<00:16, 3760.39it/s]

30%|███       | 26423/87773 [00:07<00:16, 3774.26it/s]

31%|███       | 26811/87773 [00:07<00:16, 3802.82it/s]

31%|███       | 27194/87773 [00:07<00:15, 3810.77it/s]

31%|███       | 27578/87773 [00:07<00:15, 3818.70it/s]

32%|███       | 27960/87773 [00:07<00:15, 3809.17it/s]
```

```
32%|███          | 28341/87773 [00:07<00:15, 3760.18it/s]

33%|███          | 28719/87773 [00:07<00:15, 3765.40it/s]

33%|███          | 29099/87773 [00:07<00:15, 3773.38it/s]

34%|███          | 29477/87773 [00:07<00:15, 3754.15it/s]

34%|███          | 29853/87773 [00:07<00:15, 3742.15it/s]

34%|███          | 30230/87773 [00:08<00:15, 3749.45it/s]

35%|███          | 30608/87773 [00:08<00:15, 3757.72it/s]

35%|███          | 30989/87773 [00:08<00:15, 3772.27it/s]

36%|███          | 31367/87773 [00:08<00:15, 3759.32it/s]

36%|███          | 31745/87773 [00:08<00:14, 3764.09it/s]
```

```
37%|███▋      | 32122/87773 [00:08<00:15, 3696.25it/s]
37%|███▋      | 32501/87773 [00:08<00:14, 3722.10it/s]
37%|███▋      | 32877/87773 [00:08<00:14, 3733.06it/s]
38%|███▊      | 33260/87773 [00:08<00:14, 3760.34it/s]
38%|███▊      | 33639/87773 [00:08<00:14, 3764.74it/s]
39%|███▉      | 34022/87773 [00:09<00:14, 3783.01it/s]
39%|███▉      | 34401/87773 [00:09<00:14, 3746.79it/s]
40%|███▉      | 34778/87773 [00:09<00:14, 3753.66it/s]
40%|████      | 35165/87773 [00:09<00:13, 3786.83it/s]
40%|████      | 35544/87773 [00:09<00:13, 3761.84it/s]
```

```
41%|████      | 35921/87773 [00:09<00:13, 3756.31it/s]

41%|████      | 36297/87773 [00:09<00:13, 3720.16it/s]

42%|████      | 36670/87773 [00:09<00:14, 3628.87it/s]

42%|████      | 37037/87773 [00:09<00:13, 3638.90it/s]

43%|████      | 37414/87773 [00:09<00:13, 3675.66it/s]

43%|████      | 37793/87773 [00:10<00:13, 3708.63it/s]

43%|████      | 38177/87773 [00:10<00:13, 3745.70it/s]

44%|████      | 38552/87773 [00:10<00:13, 3742.23it/s]

44%|████      | 38940/87773 [00:10<00:12, 3781.86it/s]
```

```
45%|████▌     | 39324/87773 [00:10<00:12, 3797.65it/s]

45%|████▌     | 39715/87773 [00:10<00:12, 3828.35it/s]

46%|████▌     | 40099/87773 [00:10<00:12, 3798.13it/s]

46%|████▌     | 40480/87773 [00:10<00:12, 3784.25it/s]

47%|████▋     | 40860/87773 [00:10<00:12, 3788.86it/s]

47%|████▋     | 41239/87773 [00:10<00:12, 3749.44it/s]

47%|████▋     | 41615/87773 [00:11<00:12, 3748.49it/s]

48%|████▊     | 41999/87773 [00:11<00:12, 3775.18it/s]

48%|████▊     | 42383/87773 [00:11<00:11, 3791.72it/s]

49%|████▊     | 42768/87773 [00:11<00:11, 3807.29it/s]
```

```
49%|█████     | 43150/87773 [00:11<00:11, 3810.80it/s]

50%|█████     | 43532/87773 [00:11<00:11, 3787.54it/s]

50%|█████     | 43914/87773 [00:11<00:11, 3796.94it/s]

50%|█████     | 44294/87773 [00:11<00:11, 3748.11it/s]

51%|█████     | 44669/87773 [00:11<00:11, 3741.42it/s]

51%|█████     | 45044/87773 [00:12<00:11, 3720.84it/s]

52%|█████     | 45423/87773 [00:12<00:11, 3739.13it/s]

52%|█████     | 45805/87773 [00:12<00:11, 3760.41it/s]

53%|█████     | 46182/87773 [00:12<00:11, 3757.36it/s]

53%|█████     | 46567/87773 [00:12<00:10, 3784.65it/s]
```

```
53%|█████▎    | 46946/87773 [00:12<00:10, 3768.57it/s]

54%|█████▍    | 47332/87773 [00:12<00:10, 3794.39it/s]

54%|█████▍    | 47712/87773 [00:12<00:10, 3746.33it/s]

55%|█████▍    | 48088/87773 [00:12<00:10, 3749.36it/s]

55%|█████▌    | 48474/87773 [00:12<00:10, 3779.34it/s]

56%|█████▌    | 48853/87773 [00:13<00:10, 3769.58it/s]

56%|█████▌    | 49231/87773 [00:13<00:10, 3739.68it/s]

57%|█████▋    | 49613/87773 [00:13<00:10, 3761.46it/s]

57%|█████▋    | 49998/87773 [00:13<00:09, 3786.59it/s]

57%|█████▋    | 50378/87773 [00:13<00:09, 3789.37it/s]
```

```
58%|█████▊    | 50758/87773 [00:13<00:09, 3706.54it/s]

58%|█████▊    | 51138/87773 [00:13<00:09, 3730.26it/s]

59%|█████▊    | 51523/87773 [00:13<00:09, 3764.39it/s]

59%|█████▉    | 51900/87773 [00:13<00:09, 3763.41it/s]

60%|█████▉    | 52277/87773 [00:13<00:09, 3743.38it/s]

60%|█████▉    | 52662/87773 [00:14<00:09, 3772.39it/s]

60%|██████    | 53053/87773 [00:14<00:09, 3810.73it/s]

61%|██████    | 53435/87773 [00:14<00:09, 3803.75it/s]

61%|██████▏   | 53816/87773 [00:14<00:08, 3783.88it/s]
```

```
62%|██████    | 54203/87773 [00:14<00:08, 3804.30it/s]

62%|██████    | 54587/87773 [00:14<00:08, 3814.57it/s]

63%|██████    | 54969/87773 [00:14<00:08, 3787.46it/s]

63%|██████    | 55348/87773 [00:14<00:08, 3775.84it/s]

63%|██████    | 55728/87773 [00:14<00:08, 3782.30it/s]

64%|██████    | 56112/87773 [00:14<00:08, 3797.62it/s]

64%|██████    | 56492/87773 [00:15<00:08, 3770.07it/s]

65%|██████    | 56879/87773 [00:15<00:08, 3798.81it/s]

65%|██████    | 57266/87773 [00:15<00:07, 3817.71it/s]

66%|██████    | 57649/87773 [00:15<00:07, 3820.66it/s]
```

```
66%|██████▋   | 58036/87773 [00:15<00:07, 3833.42it/s]

67%|██████▋   | 58420/87773 [00:15<00:07, 3771.14it/s]

67%|██████▋   | 58799/87773 [00:15<00:07, 3776.19it/s]

67%|██████▋   | 59181/87773 [00:15<00:07, 3788.36it/s]

68%|██████▊   | 59560/87773 [00:15<00:07, 3787.03it/s]

68%|██████▊   | 59939/87773 [00:15<00:07, 3777.14it/s]

69%|██████▊   | 60320/87773 [00:16<00:07, 3786.34it/s]

69%|██████▉   | 60699/87773 [00:16<00:07, 3774.52it/s]

70%|██████▉   | 61082/87773 [00:16<00:07, 3789.61it/s]

70%|███████   | 61464/87773 [00:16<00:06, 3796.63it/s]
```

```
70%|███████   | 61846/87773 [00:16<00:06, 3802.68it/s]
71%|███████   | 62227/87773 [00:16<00:06, 3785.96it/s]
71%|███████   | 62606/87773 [00:16<00:06, 3785.27it/s]
72%|███████   | 62985/87773 [00:16<00:06, 3777.40it/s]
72%|███████   | 63365/87773 [00:16<00:06, 3781.81it/s]
73%|███████   | 63748/87773 [00:16<00:06, 3793.85it/s]
73%|███████   | 64128/87773 [00:17<00:06, 3789.17it/s]
73%|███████   | 64507/87773 [00:17<00:06, 3785.02it/s]
74%|███████   | 64890/87773 [00:17<00:06, 3796.72it/s]
74%|███████   | 65274/87773 [00:17<00:05, 3807.08it/s]
```

```
75%|████████▌   | 65655/87773 [00:17<00:05, 3807.28it/s]

75%|████████▋   | 66036/87773 [00:17<00:05, 3793.80it/s]

76%|████████▋   | 66420/87773 [00:17<00:05, 3802.09it/s]

76%|████████▋   | 66802/87773 [00:17<00:05, 3805.47it/s]

77%|████████▊   | 67183/87773 [00:17<00:05, 3803.53it/s]

77%|████████▊   | 67564/87773 [00:17<00:05, 3753.28it/s]

77%|████████▊   | 67945/87773 [00:18<00:05, 3769.09it/s]

78%|████████▉   | 68332/87773 [00:18<00:05, 3797.39it/s]

78%|████████▉   | 68718/87773 [00:18<00:04, 3814.73it/s]
```

```
79%|███████▊  | 69100/87773 [00:18<00:04, 3797.53it/s]

79%|███████▉  | 69480/87773 [00:18<00:04, 3790.43it/s]

80%|███████▉  | 69869/87773 [00:18<00:04, 3817.10it/s]

80%|████████  | 70256/87773 [00:18<00:04, 3831.85it/s]

80%|████████  | 70640/87773 [00:18<00:04, 3799.36it/s]

81%|████████  | 71021/87773 [00:18<00:04, 3778.62it/s]

81%|████████▏ | 71399/87773 [00:18<00:04, 3771.24it/s]

82%|████████▏ | 71782/87773 [00:19<00:04, 3787.13it/s]

82%|████████▏ | 72161/87773 [00:19<00:04, 3764.29it/s]

83%|████████▎ | 72539/87773 [00:19<00:04, 3767.53it/s]
```

```
83%|████████▎ | 72926/87773 [00:19<00:03, 3796.40it/s]

84%|████████▎ | 73309/87773 [00:19<00:03, 3802.96it/s]

84%|████████▍ | 73690/87773 [00:19<00:03, 3797.51it/s]

84%|████████▍ | 74076/87773 [00:19<00:03, 3813.66it/s]

85%|████████▍ | 74458/87773 [00:19<00:03, 3813.15it/s]

85%|████████▌ | 74841/87773 [00:19<00:03, 3814.58it/s]

86%|████████▌ | 75223/87773 [00:19<00:03, 3752.14it/s]

86%|████████▌ | 75606/87773 [00:20<00:03, 3772.22it/s]

87%|████████▋ | 75993/87773 [00:20<00:03, 3798.70it/s]

87%|████████▋ | 76374/87773 [00:20<00:03, 3779.86it/s]
```

```
87%|████████▋ | 76756/87773 [00:20<00:02, 3789.82it/s]

88%|████████▊ | 77138/87773 [00:20<00:02, 3797.66it/s]

88%|████████▊ | 77523/87773 [00:20<00:02, 3812.61it/s]

89%|████████▉ | 77905/87773 [00:20<00:02, 3799.85it/s]

89%|████████▉ | 78291/87773 [00:20<00:02, 3815.23it/s]

90%|████████▉ | 78679/87773 [00:20<00:02, 3834.09it/s]

90%|█████████ | 79063/87773 [00:20<00:02, 3804.72it/s]

91%|█████████ | 79444/87773 [00:21<00:02, 3793.72it/s]

91%|█████████ | 79832/87773 [00:21<00:02, 3818.28it/s]

91%|█████████▏| 80221/87773 [00:21<00:01, 3837.98it/s]
```

```
92%|█████████▏| 80608/87773 [00:21<00:01, 3846.01it/s]

92%|█████████▏| 80993/87773 [00:21<00:01, 3846.18it/s]

93%|█████████▎| 81378/87773 [00:21<00:01, 3794.28it/s]

93%|█████████▎| 81761/87773 [00:21<00:01, 3803.36it/s]

94%|█████████▎| 82142/87773 [00:21<00:01, 3800.20it/s]

94%|█████████▍| 82528/87773 [00:21<00:01, 3815.99it/s]

94%|█████████▍| 82910/87773 [00:22<00:01, 3762.07it/s]

95%|█████████▍| 83295/87773 [00:22<00:01, 3786.27it/s]

95%|█████████▌| 83674/87773 [00:22<00:01, 3763.25it/s]
```

```
 96%|█████████▋| 84060/87773 [00:22<00:00, 3789.54it/s]

 96%|█████████▋| 84443/87773 [00:22<00:00, 3799.09it/s]

 97%|█████████▋| 84824/87773 [00:22<00:00, 3797.25it/s]

 97%|█████████▋| 85204/87773 [00:22<00:00, 3793.43it/s]

 98%|█████████▊| 85584/87773 [00:22<00:00, 3707.13it/s]

 98%|█████████▊| 85956/87773 [00:22<00:00, 3710.53it/s]

 98%|█████████▊| 86337/87773 [00:22<00:00, 3738.68it/s]

 99%|█████████▉| 86719/87773 [00:23<00:00, 3761.04it/s]

 99%|█████████▉| 87101/87773 [00:23<00:00, 3776.22it/s]

100%|█████████▉| 87488/87773 [00:23<00:00, 3802.72it/s]
```

```
100%|██████████| 87773/87773 [00:23<00:00, 3768.75it/s]
```

[5] Assignment 5: Apply Logistic Regression

1. **Apply Logistic Regression on these feature sets**
    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
    - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**
    - Find the best hyper parameter which will give the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Pertubation Test**
    - Get the weights W after fit your model with the data X i.e Train data.
    - Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e)
    - Fit the model again on data X' and get the weights W'
    - Add a small eps value(to eliminate the divisible by zero error) to W and W’ i.e W=W+10^-6 and W’ = W’+10^-6
    - Now find the % change between W and W' (| (W-W') / (W) |)*100)
    - Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in the values of percentage_change_vector

- Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is sudden rise from 1.3 to 34.6, now calculate the 99.1, 99.2, 99.3,..., 100th percentile values and get the proper value after which there is sudden rise the values, assume it is 2.5
- Print the feature names whose % change is more than a threshold x(in our example it's 2.5)

4. **Sparsity**

   - Calculate sparsity on weight vector obtained after using L1 regularization

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

5. **Feature importance**

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

6. **Feature engineering**

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

7. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
   Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please

visualize your confusion matrices using seaborn heatmaps.

8. **Conclusion**

   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

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

Applying Logistic Regression

[5.1] Logistic Regression on BOW, SET 1

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

In [0]:
```
# Please write all the code with proper documentation
```

In [102]:
```
final.shape
final.head()
```

Out[102]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	He
22620	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1	1
22621	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0	0
70677	76870	B00002N8SM	A19Q006CSFT011	Arlielle	0	0
70676	76869	B00002N8SM	A1FYH4S02BW7FN	wonderer	0	0
70675	76868	B00002N8SM	AUE8TB5VHS6ZV	eyeofthestorm	0	0

In [103]:

```
final['PreprocessedText'] = preprocessed_reviews
final['PreprocessedSummary'] = preprocessed_summary

final['Final_Text'] = final['PreprocessedText'] + final['PreprocessedSu
mmary']
final.head(3)
```

Out[103]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Help
22620	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1	1
22621	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0	0
70677	76870	B00002N8SM	A19Q006CSFT011	Arlielle	0	0

In [0]:

```
#sorting based on time

final['Time'] = pd.to_datetime(final['Time'])
final = final.sort_values(by = 'Time')
```

In [0]:
```
#Spliting data for train , test
from sklearn.model_selection import train_test_split

X_train , X_test ,y_train , y_test = train_test_split(final['Final_Text'] , final['Score'] , test_size = 0.3,random_state=42)
```

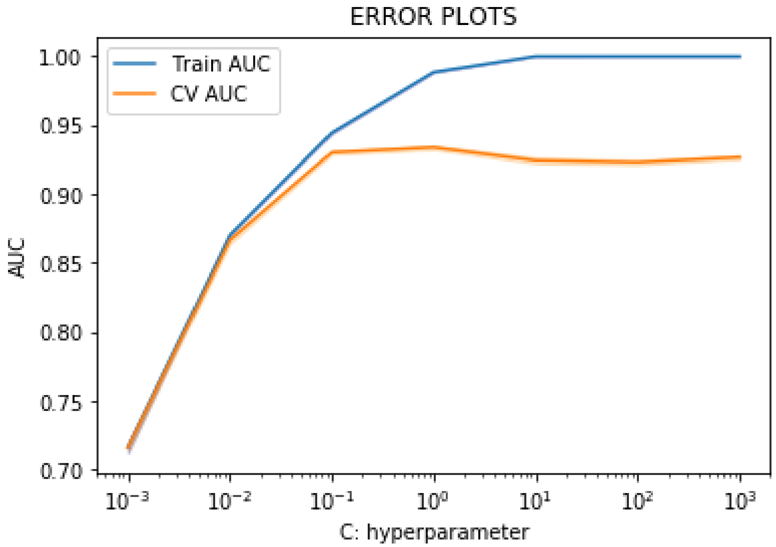
In [0]:
```
#using BOW vectorization
vectorizer = CountVectorizer()
x_train_bow = vectorizer.fit_transform(X_train)
x_test_bow = vectorizer.transform(X_test)
```

In [106]:
```
type(x_train_bow)
#x_train_bow.shape
```

Out[106]: scipy.sparse.csr.csr_matrix

In [107]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

C_Values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

parameters = {'C': C_Values }

clf = GridSearchCV(LogisticRegression(penalty="l1"), parameters, cv=3, scoring='roc_auc',n_jobs=-1)
clf.fit(x_train_bow,y_train)

train_auc = clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std = clf.cv_results_['std_test_score']

plt.plot(C_Values, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(C_Values,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')
```

```python
plt.plot(C_Values, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4
084039
plt.gca().fill_between(C_Values,cv_auc - cv_auc_std,cv_auc + cv_auc_std
,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.xscale("log")
plt.title("ERROR PLOTS")
plt.show()
```

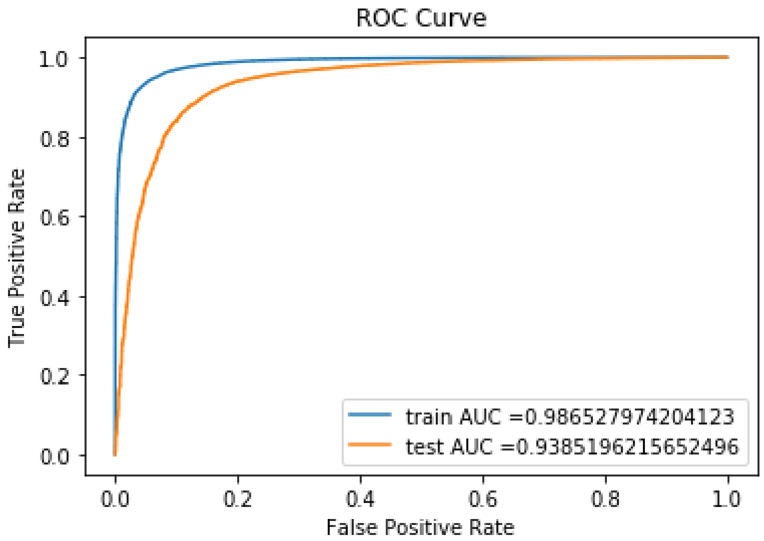


In [108]:
```python
Best_AUC_L1_BOW = clf.best_score_
optimal_C_Value = clf.best_params_['C']
print(Best_AUC_L1_BOW)
print(optimal_C_Value)
```

```
0.93382064517835
1
```

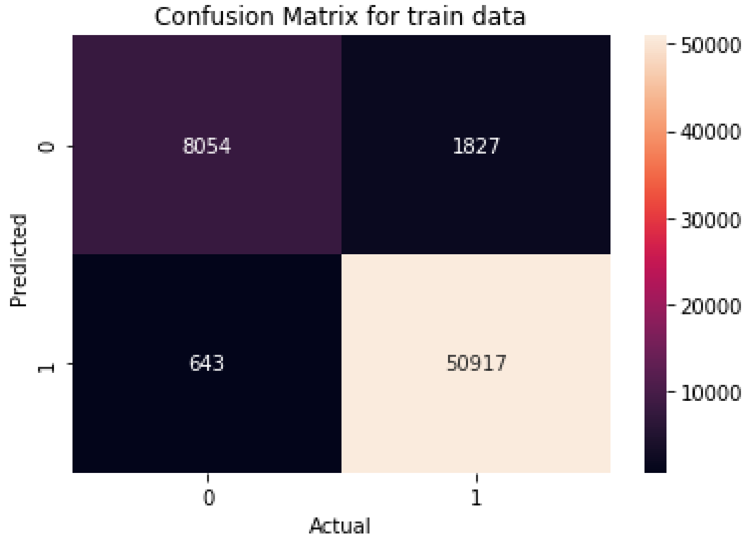
In [109]:
```python
Model_L1_BOW = LogisticRegression(C = optimal_C_Value,penalty='l1')
Model_L1_BOW.fit(x_train_bow,y_train)
```

```
train_fpr, train_tpr, thresholds = roc_curve(y_train,Model_L1_BOW.predi
ct_proba(x_train_bow)[:,1])
test_fpr, test_tpr , thresholds = roc_curve(y_test,Model_L1_BOW.predict
_proba(x_test_bow)[:,1])

plt.plot(train_fpr,train_tpr,label="train AUC ="+str(auc(train_fpr, tra
in_tpr)))
plt.plot(test_fpr,test_tpr,label="test AUC ="+str(auc(test_fpr,test_tpr
)))
plt.xlabel("False Positive Rate")
plt.ylabel('True Positive Rate')
plt.title("ROC Curve")
plt.legend()
plt.show()
```

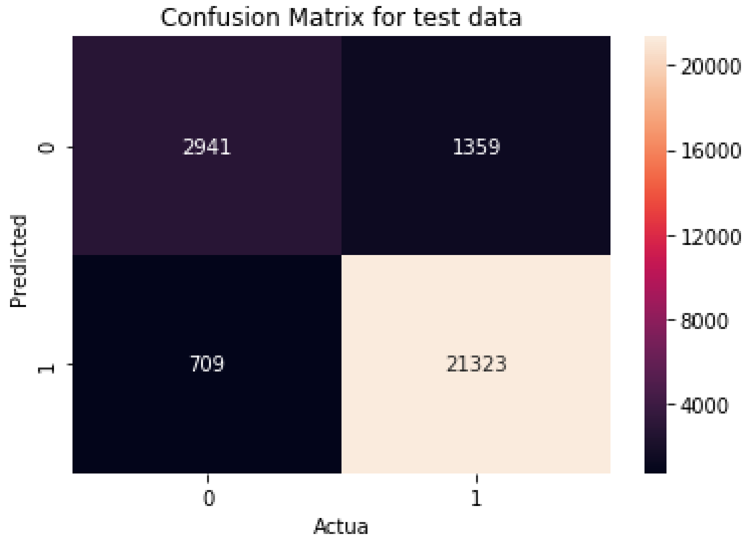


In [110]:
```
conf_matrix = confusion_matrix(y_train, Model_L1_BOW.predict(x_train_bo
w))
class_label = [0, 1]
df_conf_matrix = pd.DataFrame(
conf_matrix, index=class_label, columns=class_label)
```

```
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix for train data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



In [111]:
```
conf_matrix = confusion_matrix(y_test,Model_L1_BOW.predict(x_test_bow))
class_label = [0,1]
df_conf_matrix = pd.DataFrame(conf_matrix,index=class_label,columns=class_label)
sns.heatmap(df_conf_matrix,annot = True ,fmt = 'd')
plt.title("Confusion Matrix for test data")
plt.xlabel("Actua")
plt.ylabel("Predicted")
plt.show()
```



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

In [0]:
```
# Please write all the code with proper documentation
```

In [112]:
```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lam
bda (decreasing C)
clf = LogisticRegression(C=100, penalty='l1');
clf.fit(x_train_bow, y_train);
w = clf.coef_
print(np.count_nonzero(w))
```

11951

In [113]:
```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lam
bda (decreasing C)
clf = LogisticRegression(C=10, penalty='l1');
clf.fit(x_train_bow, y_train);
w = clf.coef_
print(np.count_nonzero(w))
```

10107

In [114]:
```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lam
bda (decreasing C)
clf = LogisticRegression(C=0.1, penalty='l1');
clf.fit(x_train_bow, y_train);
w = clf.coef_
print(np.count_nonzero(w))
```

906

In [115]:
```
# More Sparsity (Fewer elements of W* being non-zero) by increasing Lam
bda (decreasing C)
clf = LogisticRegression(C=0.01, penalty='l1');
clf.fit(x_train_bow, y_train);
w = clf.coef_
print(np.count_nonzero(w))
```

135

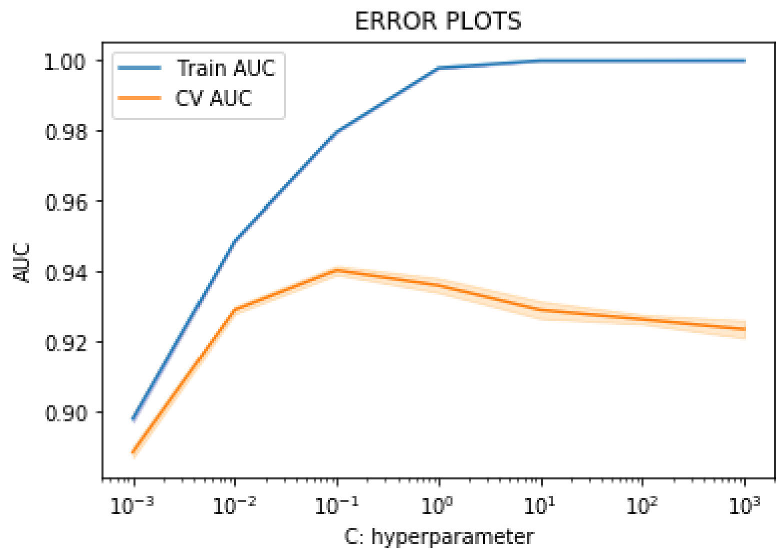
So here as the C value decreases nothing but Lamda value increases the number of weight vectors becomes non-zeros.

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

In [0]:
```
# Please write all the code with proper documentation
```

In [116]:
```
from sklearn.model_selection import  GridSearchCV
from sklearn.linear_model import LogisticRegression

C_Values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

parameters = {'C': C_Values }

clf = GridSearchCV(LogisticRegression(penalty='l2'),parameters,cv=3, sc
```

```
oring='roc_auc',n_jobs=-1)
clf.fit(x_train_bow,y_train)

train_auc = clf.cv_results_["mean_train_score"]
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_["mean_test_score"]
cv_auc_std= clf.cv_results_['std_test_score']


plt.plot(C_Values,train_auc, label= 'Train AUC')
plt.gca().fill_between(C_Values,train_auc - train_auc_std,train_auc + t
rain_auc_std,alpha=0.2,color='darkblue')

plt.plot(C_Values, cv_auc, label='CV AUC')
plt.gca().fill_between(C_Values,cv_auc - cv_auc_std,cv_auc + cv_auc_std
,alpha=0.2,color='darkorange')

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

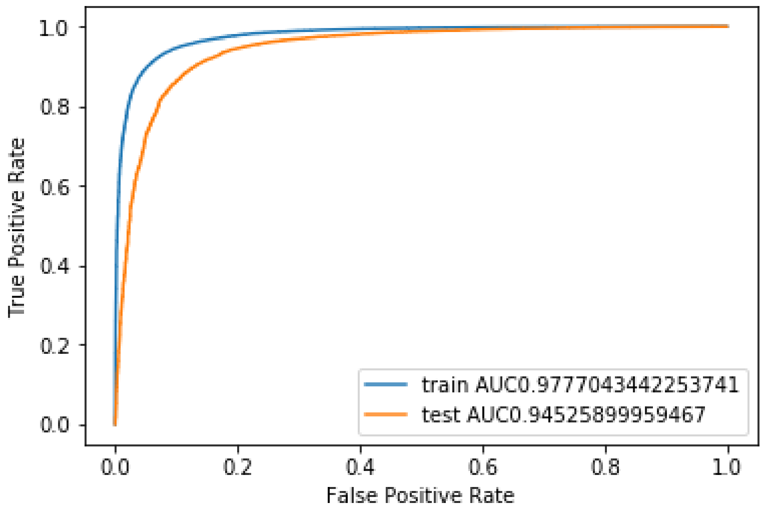


In [117]:
```
Best_AUC_L2_BOW = clf.best_score_
optimal_C_l2_Value = clf.best_params_['C']
print(Best_AUC_L2_BOW)
print(optimal_C_l2_Value)
```

```
0.9403322775359962
0.1
```

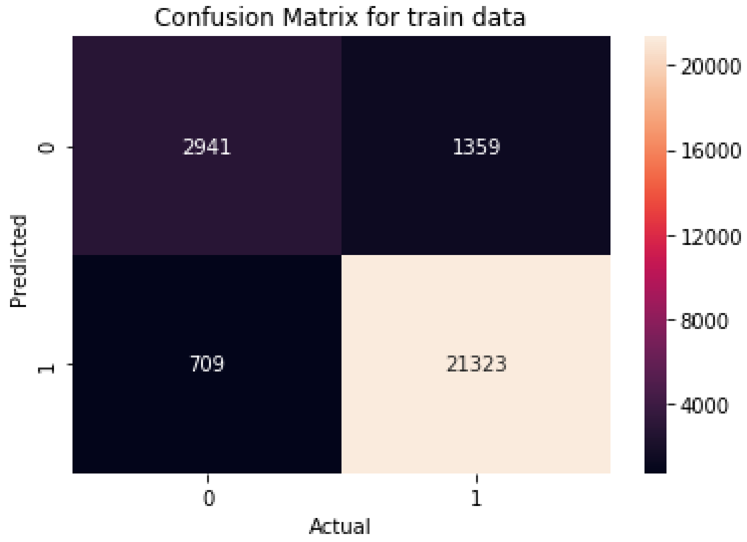
In [118]:
```
Model_L2_BOW = LogisticRegression(C= optimal_C_l2_Value,penalty = "l2")
Model_L2_BOW.fit(x_train_bow,y_train)

train_fpr,train_tpr,thresholds = roc_curve(y_train,Model_L2_BOW.predict
_proba(x_train_bow)[:,1])
test_fpr,test_tpr,thresholds = roc_curve(y_test,Model_L2_BOW.predict_pr
oba(x_test_bow)[:,1])

plt.plot(train_fpr,train_tpr,label = "train AUC"+str(auc(train_fpr,trai
n_tpr)))
plt.plot(test_fpr,test_tpr,label = "test AUC"+str(auc(test_fpr,test_tpr
)))
plt.legend()
plt.xlabel("False Positive Rate")
```

```
plt.ylabel("True Positive Rate")
plt.show()
```

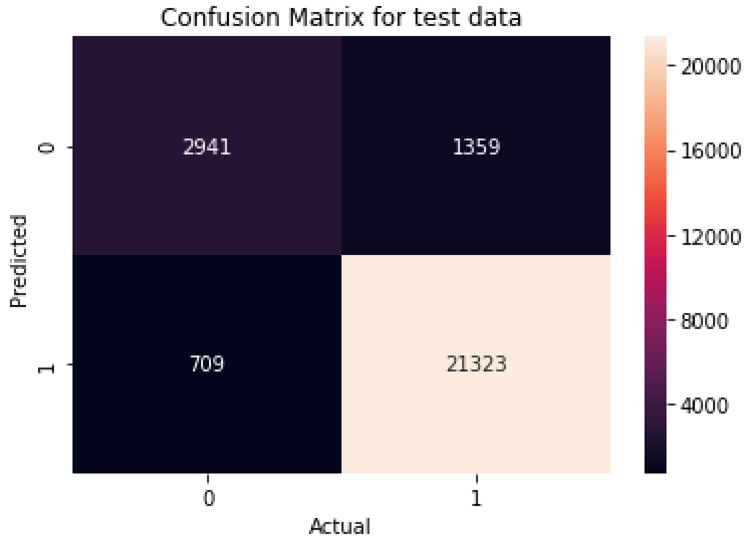


In [119]:

```
conf_matr = confusion_matrix(y_train,Model_L2_BOW.predict(x_train_bow))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')

plt.title("Confusion Matrix for train data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



In [120]:
```
conf_matr = confusion_matrix(y_test,Model_L2_BOW.predict(x_test_bow))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')

plt.title("Confusion Matrix for test data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



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

In [0]:
```
# Please write all the code with proper documentation
```

In [121]:
```
w_old = Model_L2_BOW.coef_
print(w_old)
```

```
[[ 4.89504736e-03  8.92745749e-02  4.27597586e-03 ... -3.52937805e-02
   5.93905558e-05  1.23914793e-02]]
```

In [0]:
```
## Firstly we will get the weights resulted from our model
#getting coefficients
w_old = (Model_L2_BOW.coef_).ravel()

#adding noise to data

noise = np.random.normal(loc=0.0, scale=0.000001)

x_train_bow.data = x_train_bow.data + noise
```

```
#Now fitting the model with noise data

noise_model = LogisticRegression(C=optimal_C_l2_Value,penalty="l2")
noise_model.fit(x_train_bow,y_train)

#getting coefficients from new noise added model
w_new = (noise_model.coef_).ravel()
```

In [0]:

```
#Adding small value of epsilon
epsilon = 1e-6

w_old = w_old + epsilon
w_new = w_new + epsilon
```

In [126]:

```
chng = (w_old-w_new)
perct_chng = list (abs ((chng/w_old) * 100))
print(len(perct_chng))
```

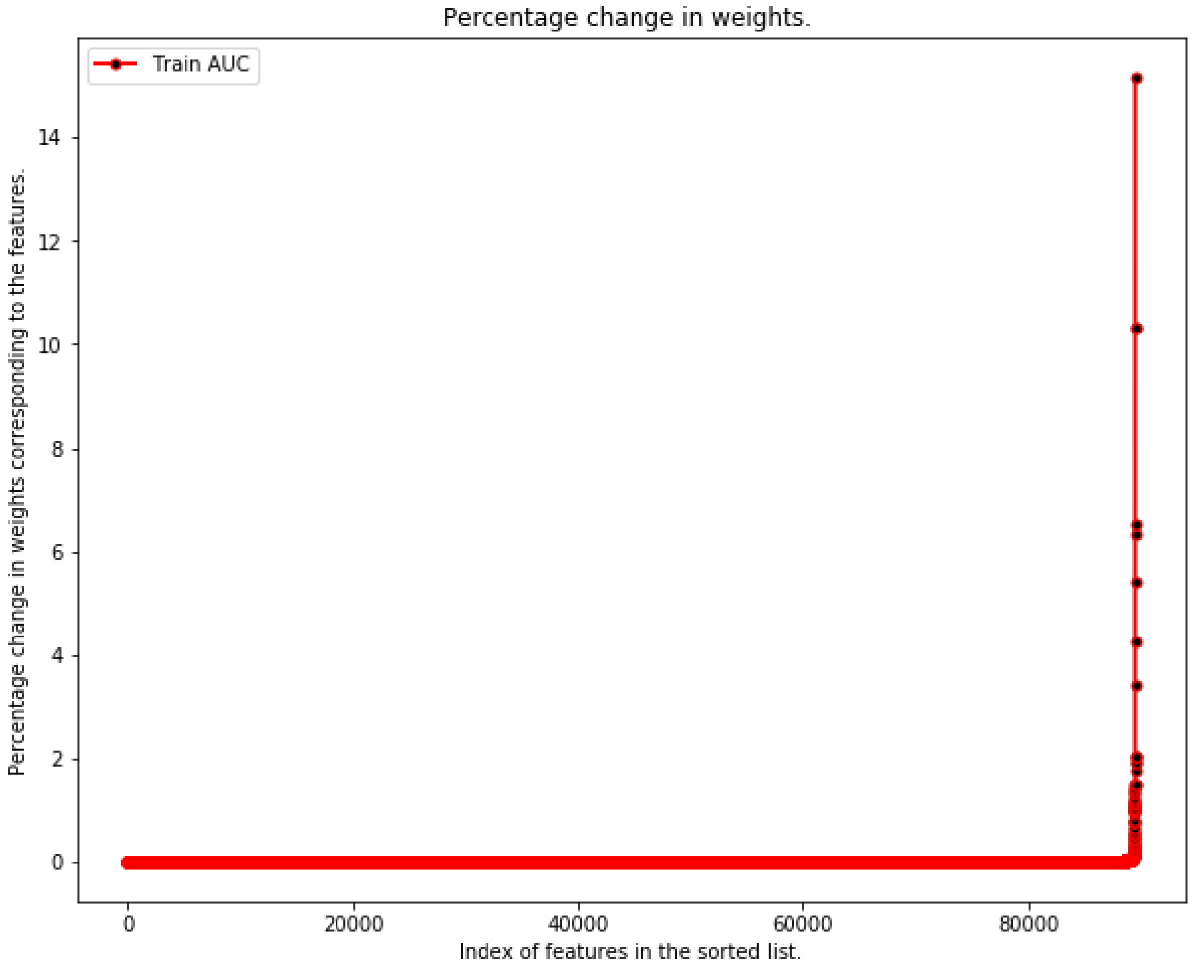
```
89667
```

In [0]:

```
#calculating % change in w_old and w_new

chng = (w_old-w_new)
perct_chng = list (abs ((chng/w_old) * 100))
perct_chng.sort()
```

In [0]:

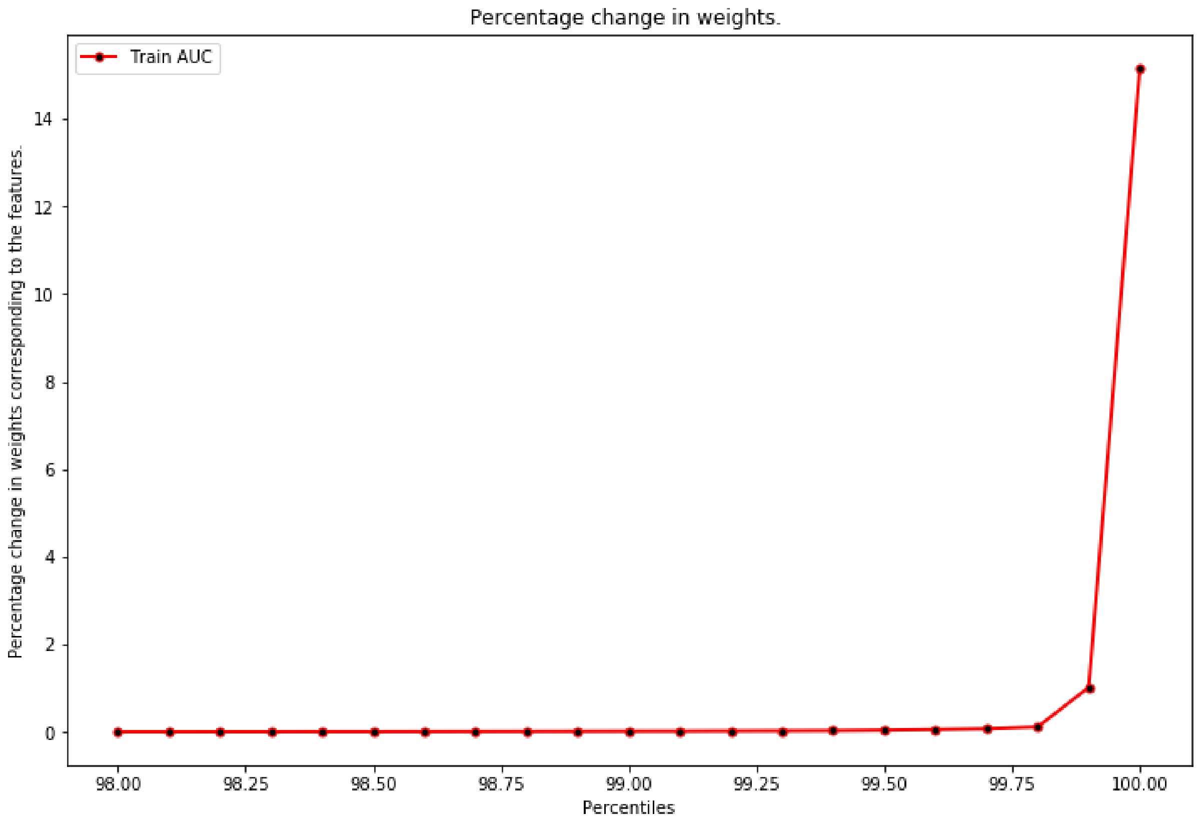
```
plt.figure(figsize=(10,8))
plt.plot(np.arange(0,len(perct_chng)) , perct_chng, color='red', linestyle='-', linewidth=2, marker='.', markerfacecolor='black', markersize=10, label='Train AUC')
plt.title('Percentage change in weights. ')
plt.xlabel('Index of features in the sorted list.')
plt.ylabel('Percentage change in weights corresponding to the features.')
plt.legend()
plt.show()
```



if we consider total x - axis as 100 percentile then from the last 98 to 100 there is a sudden rise.

In [0]:
```
percentiles = [i for i in np.arange(98.0,100.1,0.1)]
percentile_val = [np.percentile(perct_chng, i) for i in percentiles]
plt.figure(figsize=(12,8))
plt.plot(np.arange(98.0,100.1,0.1) , percentile_val, color='red', linestyle='-', linewidth=2, marker='.', markerfacecolor='black', markersize=10, label='Train AUC')
```

```
plt.title('Percentage change in weights. ')
plt.xlabel('Percentiles')
plt.ylabel('Percentage change in weights corresponding to the feature
s.')
plt.legend()
plt.show()
```



so from this plot we can see that at 99.8 there is a sudden raise in the plot.

In [0]:
```
raise_value = np.percentile(perct_chng,99.8)

Multi_Colnr_Features = []
```

```
for i in perct_chng:
  if(i>raise_value):
    Multi_Colnr_Features.append(i)
```

In [0]:
```
feature_index = []
for i in range(0,len(Multi_Colnr_Features)):
  ind = perct_chng.index(Multi_Colnr_Features[i])
  feature_index.append(ind)
```

In [0]:
```
feat_names = vectorizer.get_feature_names()

mul_feat = []
for index in feature_index:
  mul_feat.append(feat_names[index])
```

In [0]:
```
print("Features that are colinear ")
print(mul_feat.hea)
```

```
Features that are colinear
['zareba', 'zareba', 'zatarain', 'zatarain', 'zatarains', 'zataran', 'z
atarans', 'zatarian', 'zaxby', 'zazu', 'zazu', 'zazu', 'zeal', 'zealan
d', 'zealand', 'zealand', 'zealand', 'zealand', 'zealand', 'zede', 'zel
lies', 'zen', 'zeor', 'zero', 'zerocholesterol', 'zerodieter', 'zerodie
ter', 'zerodieter', 'zerosodium', 'zerosooooo', 'zerotwinnings', 'zes
t', 'zested', 'zester', 'zestier', 'zeststandard', 'zesty', 'zetia', 'z
etia', 'zeus', 'zevia', 'zeviafence', 'zeviagood', 'zevias', 'zevias',
'zevias', 'zevias', 'zevias', 'zevias', 'zevias', 'zi', 'zica', 'zica',
'zicoactually', 'zicobetter', 'zicobetter', 'zicokeeps', 'zicon', 'zico
not', 'zicoone', 'zicoone', 'zicorefreshingly', 'zicos', 'zicothumbs',
'zicozico', 'ziegelmair', 'ziegelmair', 'ziegelmair', 'ziggiesi', 'zigg
le', 'zilch', 'zilchamazon', 'zillion', 'zillions', 'zimmern', 'zimmer
n', 'zinc', 'zinco', 'zinfandel', 'zinfandels', 'zing', 'zingbest', 'zi
ngbest', 'zingers', 'zinggreat', 'zinggreat', 'zinggreat', 'zinging',
'zinging', 'zinging', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo',
'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo',
'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo',
'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo',
```

```
'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo',
'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo',
'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo', 'zioo',
'zioo', 'zucini', 'zuckerman', 'zuk', 'zuk', 'zukebailey', 'zukebelieve
r', 'zukedog', 'zukegreat', 'zukelove', 'zukes', 'zukesgreat', 'zukeslo
ve', 'zukesloved', 'zukesloved', 'zukesloved', 'zuma', 'zumba', 'zumb
a', 'zupreem', 'zupreem', 'zupreem', 'zyliss', 'zylitol', 'zymox', 'zyr
up', 'zzzzzchamomile', 'zzzzzzz', 'zzzzzzz', 'zzzzzzzzzzznot']
```

[5.1.3] Feature Importance on BOW, SET 1

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

In [0]:
```
# Please write all the code with proper documentation
```

In [0]:
```
#taking features from 0 to last in descending order and then taking top
 10 features and here we are getting high values are positive and low va
lues are negative.

top_positive_features = (-Model_L1_BOW.coef_[0, :]).argsort()

top_positive_features = np.take(vectorizer.get_feature_names(), top_pos
itive_features[:10])
print(top_positive_features)
```

```
['emeraldforest' 'ordergreat' 'pleasantly' 'friday' 'location'
 'likebetter' 'yea' 'compares' 'baggreat' 'worried']
```

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

In [0]:
```
# Please write all the code with proper documentation
```

In [0]:
```
top_negative_features = (Model_L1_BOW.coef_[0, :]).argsort()
```

```
top_negative_features = np.take(vectorizer.get_feature_names(), top_neg
ative_features[:10])
print(top_negative_features)
```

```
['stassen' 'disappointednot' 'disappointmentnot' 'recommendnot' 'weanin
g'
 'notnot' 'fixens' 'purchasenot' 'moneynot' 'rangekind']
```

[5.2] Logistic Regression on TFIDF, SET 2

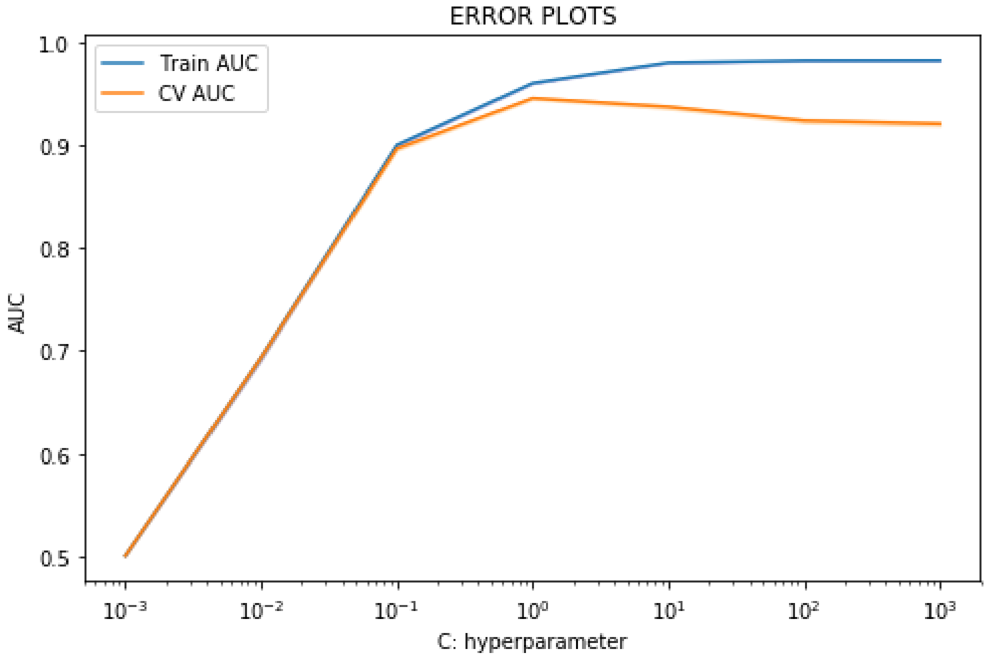
[5.2.1] Applying Logistic Regression with L1 regularization on TFIDF, SET 2

In [0]:
```
# Please write all the code with proper documentation
```

In [0]:
```
tfidf_vect = TfidfVectorizer(min_df=50)
X_train_tfidf = tfidf_vect.fit_transform(X_train)
X_test_tfidf = tfidf_vect.transform(X_test)
```

In [0]:
```
# Please write all the code with proper documentation

c_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

model = LogisticRegression(penalty="l1")
parameters = {'C':c_values}
clf = GridSearchCV(model, parameters, cv=3, scoring='roc_auc',n_jobs=-1
)
clf.fit(X_train_tfidf, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(8,5))
```

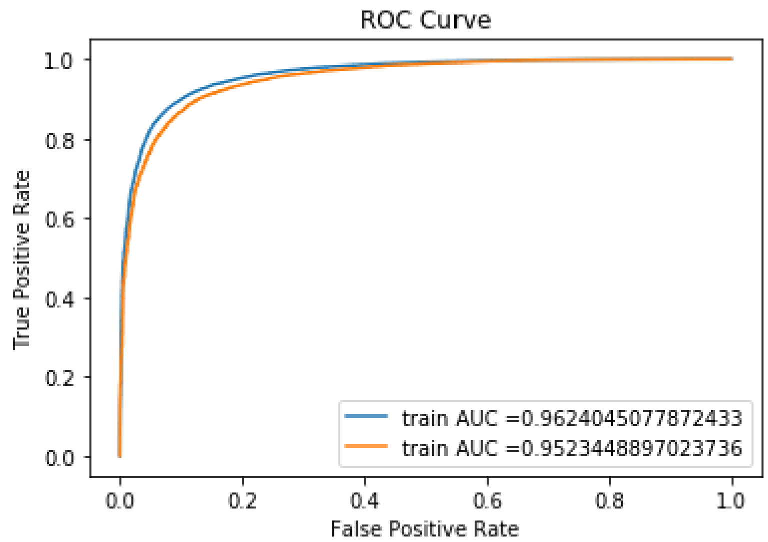
```
plt.plot(c_values, train_auc, label='Train AUC')

plt.gca().fill_between(c_values,train_auc - train_auc_std,train_auc + t
rain_auc_std,alpha=0.2,color='darkblue')

plt.plot(c_values, cv_auc, label='CV AUC')

plt.gca().fill_between(c_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std
,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.xscale("log")
plt.title("ERROR PLOTS")
plt.show()
```



In [0]:
```
auc_l1_tfidf = clf.best_score_
optimal_c_l1_tfidf = clf.best_params_["C"]
print(optimal_c_l1_tfidf)
```

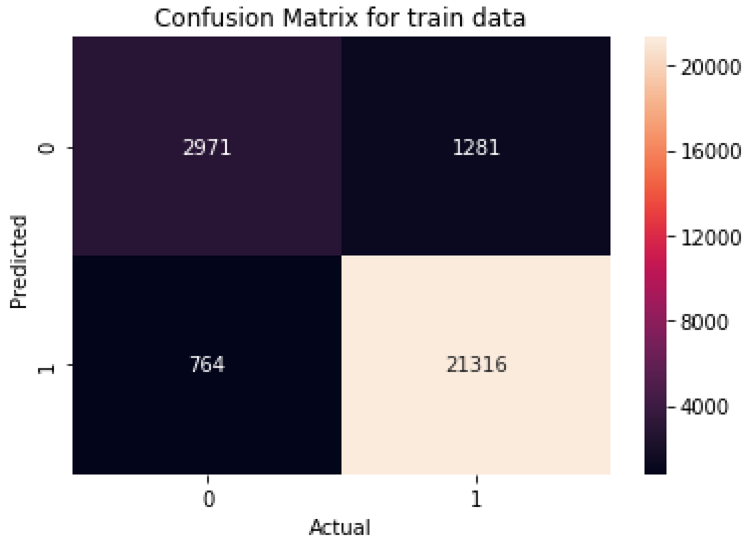
1

In [0]:
```
model_l1_tfidf = LogisticRegression(C=optimal_c_l1_tfidf,penalty="l1")
model_l1_tfidf.fit(X_train_tfidf,y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, model_l1_tfidf.pr
edict_proba(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, model_l1_tfidf.predi
ct_proba(X_test_tfidf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test
_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()
```



In [0]:
```
conf_matr = confusion_matrix(y_train,model_l1_tfidf.predict(X_train_tfi
```

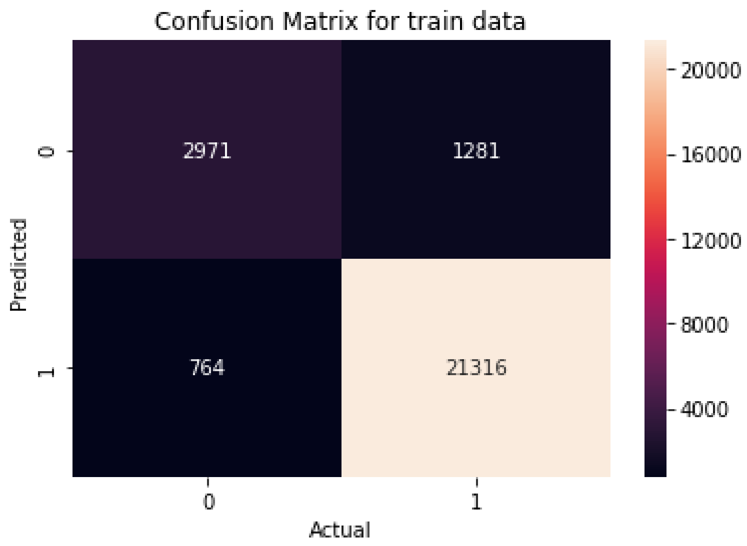
```
df))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')

plt.title("Confusion Matrix for train data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



In [0]:

```
conf_matr = confusion_matrix(y_test,model_l1_tfidf.predict(X_test_tfidf
))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')
```

```
plt.title("Confusion Matrix for train data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```

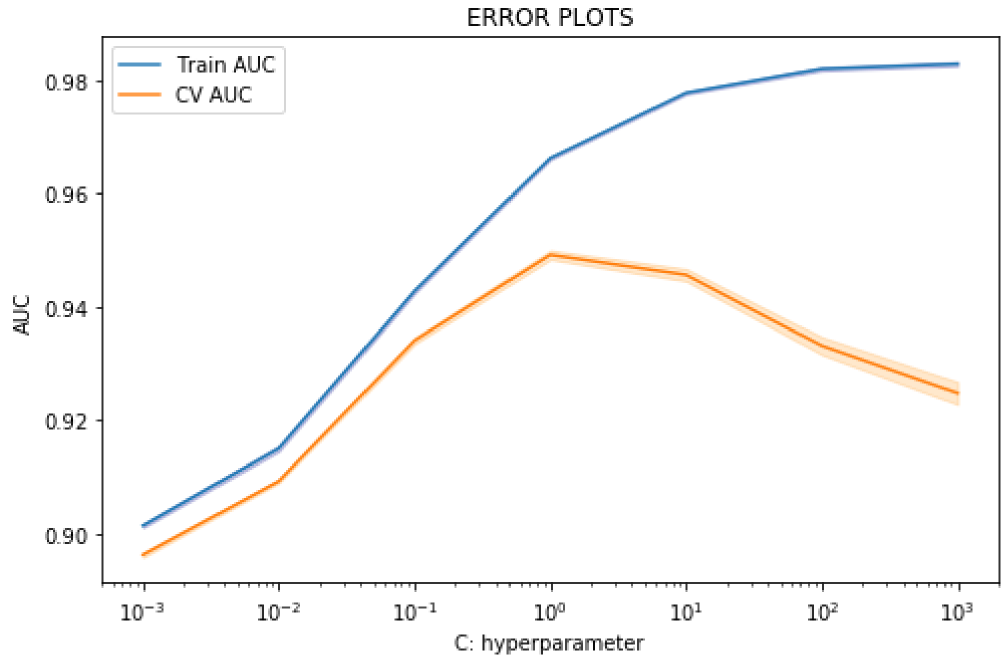


[5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

In [0]:
```
# Please write all the code with proper documentation
```

In [0]:
```
# Please write all the code with proper documentation

c_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

model = LogisticRegression(penalty="l2")
parameters = {'C':c_values}
clf = GridSearchCV(model, parameters, cv=3, scoring='roc_auc',n_jobs=-1
)
clf.fit(X_train_tfidf, y_train)
```

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(8,5))
plt.plot(c_values, train_auc, label='Train AUC')

plt.gca().fill_between(c_values,train_auc - train_auc_std,train_auc + t
rain_auc_std,alpha=0.2,color='darkblue')

plt.plot(c_values, cv_auc, label='CV AUC')

plt.gca().fill_between(c_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std
,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.xscale("log")
plt.title("ERROR PLOTS")
plt.show()
```



In [0]:

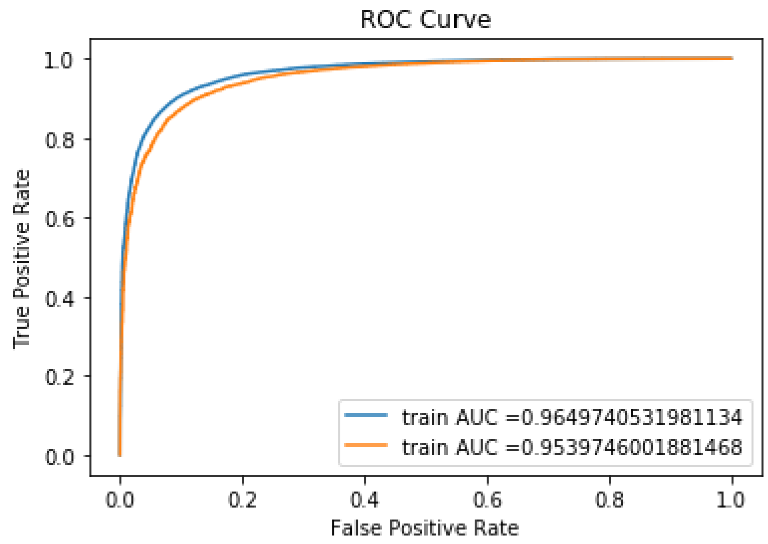
```
auc_l2_tfidf = clf.best_score_
optimal_c_l2_tfidf = clf.best_params_["C"]
print(optimal_c_l2_tfidf)
```

1

In [0]:

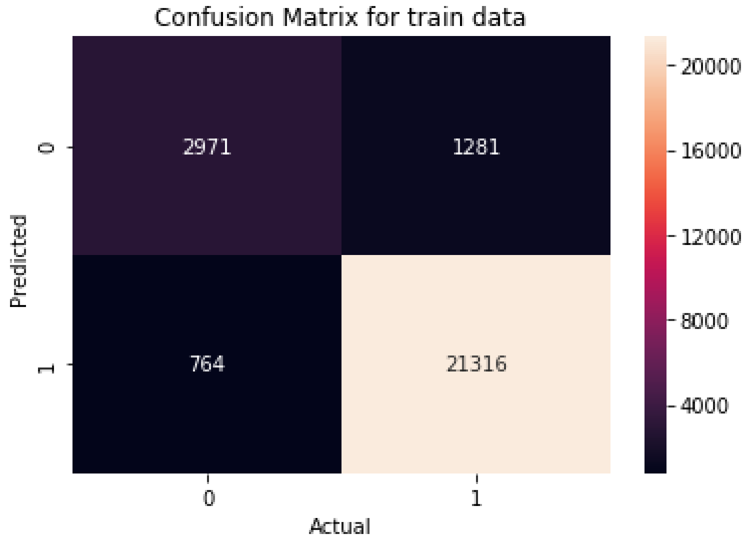
```
model_l2_tfidf = LogisticRegression(C=optimal_c_l2_tfidf,penalty="l2")
model_l2_tfidf.fit(X_train_tfidf,y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, model_l2_tfidf.pr
edict_proba(X_train_tfidf)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, model_l2_tfidf.predi
ct_proba(X_test_tfidf)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test
_tpr)))
```

```
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()
```

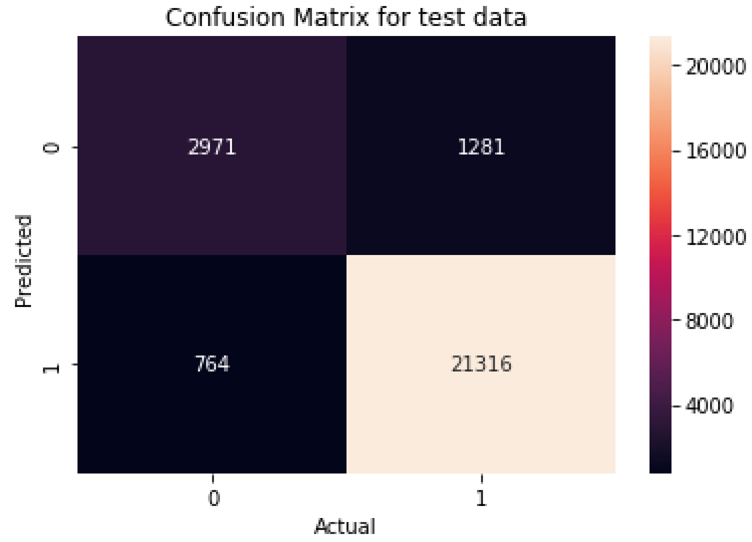


In [0]:

```
conf_matr = confusion_matrix(y_train,model_l2_tfidf.predict(X_train_tfi
df))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')

plt.title("Confusion Matrix for train data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



In [0]:
```
conf_matr = confusion_matrix(y_test,model_l2_tfidf.predict(X_test_tfidf
))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')

plt.title("Confusion Matrix for test data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



[5.2.3] Feature Importance on TFIDF, SET 2

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

In [0]:
```
# Please write all the code with proper documentation
```

In [0]:
```
#taking features from 0 to last in descending order and then taking top
 10 features and here we are getting high values are positve and low va
lues are negative.

top_positive_features = (-model_l2_tfidf.coef_[0, :]).argsort()

top_positive_features = np.take(tfidf_vect.get_feature_names(), top_pos
itive_features[:10])
print(top_positive_features)
```

```
['great' 'delicious' 'best' 'perfect' 'loves' 'excellent' 'wonderful'
 'love' 'nice' 'amazing']
```

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

In [0]:
```
# Please write all the code with proper documentation
```

In [0]:
```
top_positive_features = (model_l2_tfidf.coef_[0, :]).argsort()

top_positive_features = np.take(tfidf_vect.get_feature_names(), top_pos
itive_features[:10])
print(top_positive_features)
```

```
['worst' 'not' 'disappointed' 'awful' 'terrible' 'horrible'
 'disappointing' 'disappointment' 'threw' 'unfortunately']
```

[5.3] Logistic Regression on AVG W2V, SET 3

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

In [0]:
```
# Please write all the code with proper documentation
```

In [0]:
```
# Train your own Word2Vec model using your own text corpus

X_train_sentance=[]
for sentance in X_train:
    X_train_sentance.append(sentance.split())

X_test_sentance=[]
for sentance in X_test:
    X_test_sentance.append(sentance.split())

w2v_model=Word2Vec(X_train_sentance,min_count=5,size=100, workers=4)


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

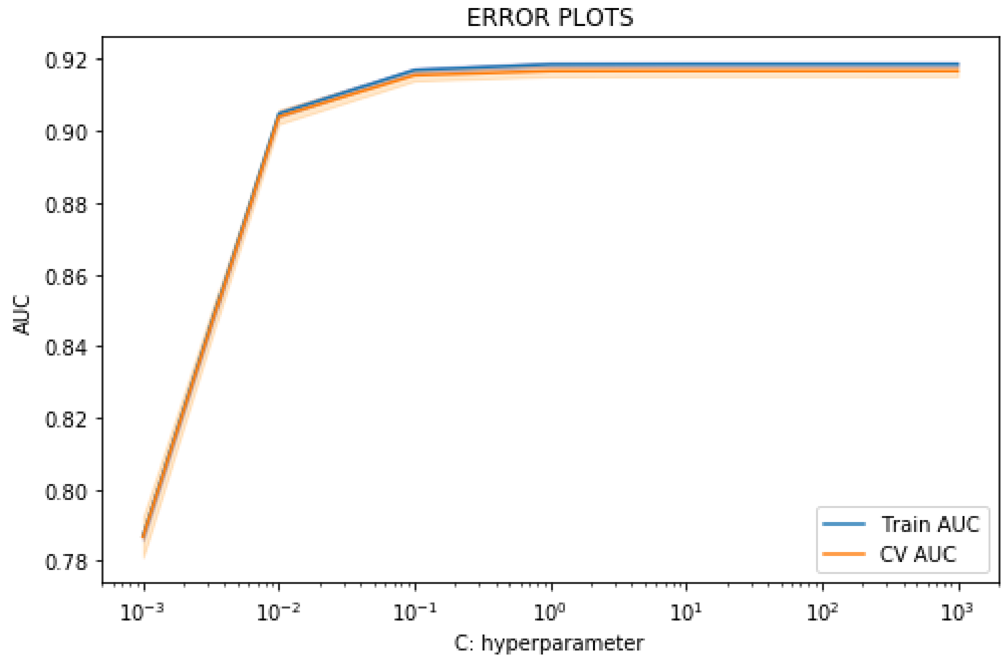
```python
print("number of words that occured minimum 5 times ",len(w2v_words))

X_train_vectors = []
for sent in X_train_sentance:
    sent_vec = np.zeros(100)
    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
    X_train_vectors.append(sent_vec)


X_test_vectors = []
for sent in X_test_sentance:
    sent_vec = np.zeros(100)
    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
    X_test_vectors.append(sent_vec)
```

```
number of words that occured minimum 5 times  16057
```

In [0]:

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

c_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

model = LogisticRegression(penalty="l1")
parameters = {'C':c_values}
clf = GridSearchCV(model, parameters, cv=3, scoring='roc_auc',n_jobs=-1
```

```
)
clf.fit(X_train_vectors, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(8,5))
plt.plot(c_values, train_auc, label='Train AUC')

plt.gca().fill_between(c_values,train_auc - train_auc_std,train_auc + t
rain_auc_std,alpha=0.2,color='darkblue')

plt.plot(c_values, cv_auc, label='CV AUC')

plt.gca().fill_between(c_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std
,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.xscale("log")
plt.title("ERROR PLOTS")
plt.show()
```

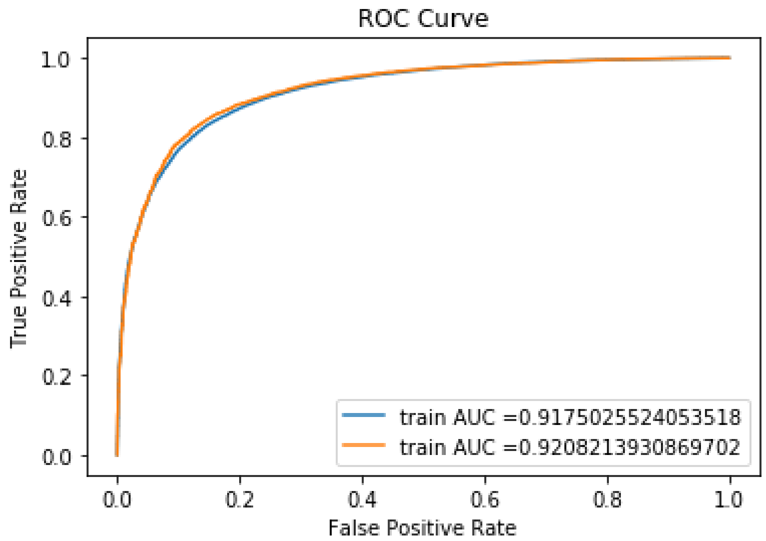


In [0]:
```
auc_l1_AvgW2V = clf.best_score_
optimal_c_l1_AvgW2V = clf.best_params_["C"]
print(optimal_c_l1_AvgW2V)
```

```
10
```

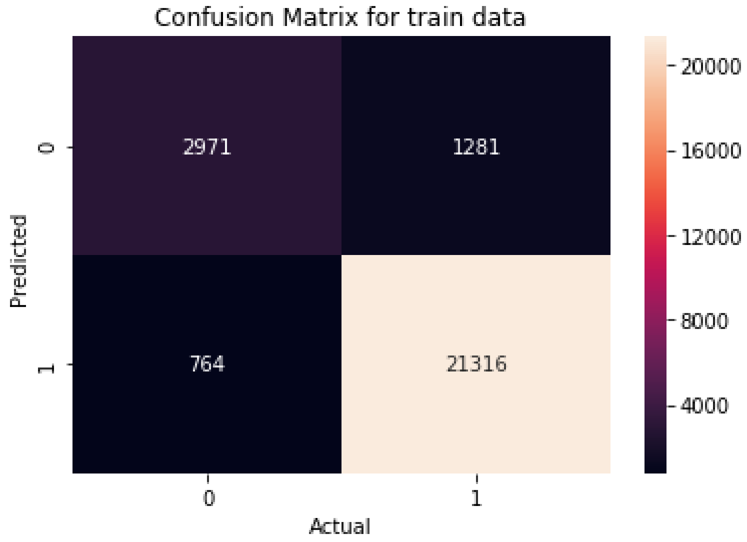
In [0]:
```
model_l1_AvgW2V = LogisticRegression(C=optimal_c_l1_AvgW2V,penalty="l1"
)
model_l1_AvgW2V.fit(X_train_vectors,y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, model_l1_AvgW2V.p
redict_proba(X_train_vectors)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, model_l1_AvgW2V.pred
ict_proba(X_test_vectors)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test
```

```
_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()
```

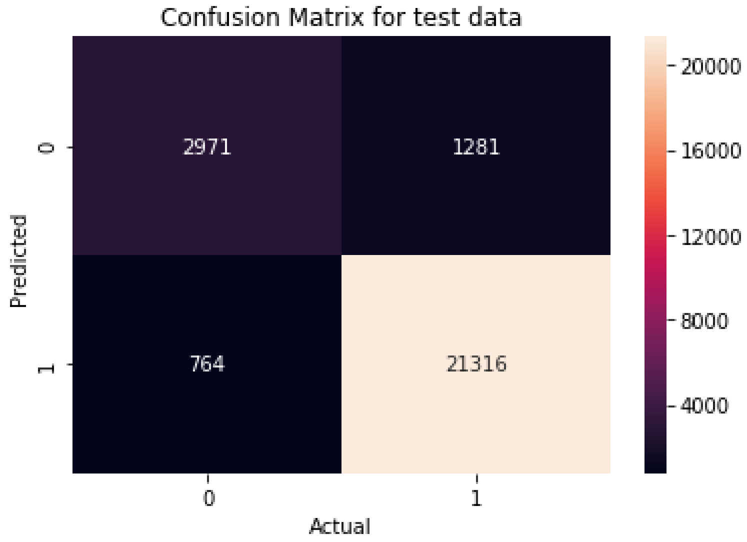


In [0]:

```
conf_matr = confusion_matrix(y_train,model_l1_AvgW2V.predict(X_train_ve
ctors))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')

plt.title("Confusion Matrix for train data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



In [0]:
```
conf_matr = confusion_matrix(y_test,model_l1_AvgW2V.predict(X_test_vect
ors))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')

plt.title("Confusion Matrix for test data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```

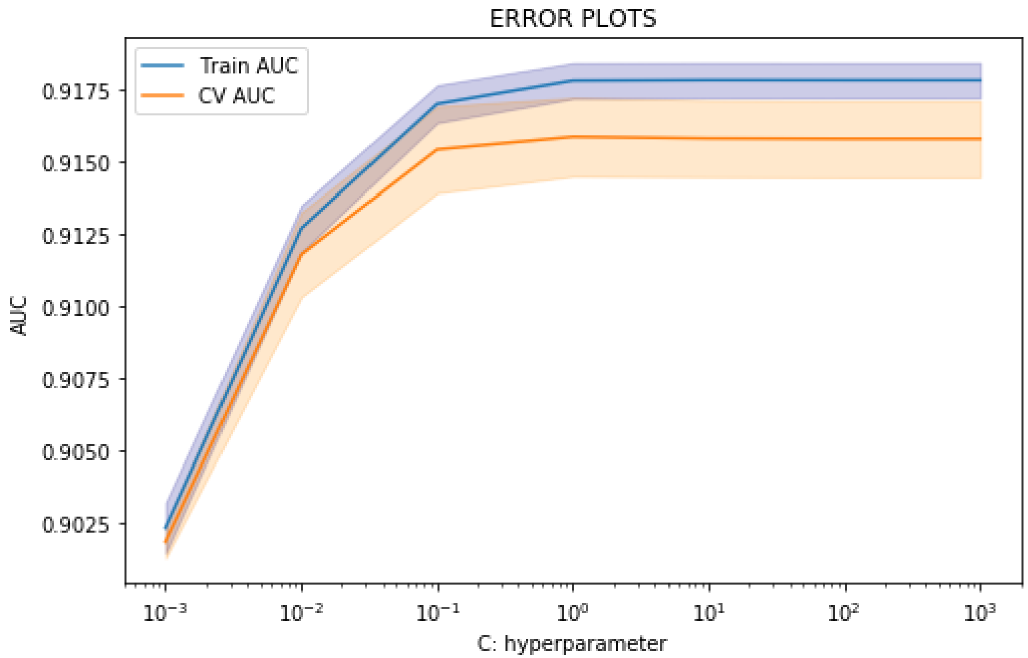


[5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

In [0]:
```
# Please write all the code with proper documentation
```

In [0]:
```
c_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

model = LogisticRegression(penalty="l2")
parameters = {'C':c_values}
clf = GridSearchCV(model, parameters, cv=3, scoring='roc_auc',n_jobs=-1
)
clf.fit(X_train_vectors, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(8,5))
plt.plot(c_values, train_auc, label='Train AUC')
```

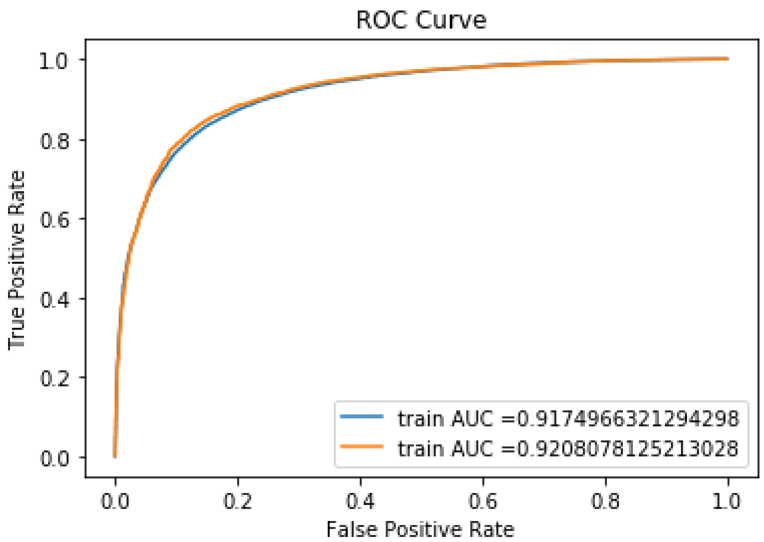
```
plt.gca().fill_between(c_values,train_auc - train_auc_std,train_auc + t
rain_auc_std,alpha=0.2,color='darkblue')

plt.plot(c_values, cv_auc, label='CV AUC')

plt.gca().fill_between(c_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std
,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.xscale("log")
plt.title("ERROR PLOTS")
plt.show()
```



In [0]:
```
auc_l2_AvgW2V = clf.best_score_
optimal_c_l2_AvgW2V = clf.best_params_["C"]
print(optimal_c_l2_AvgW2V)
```

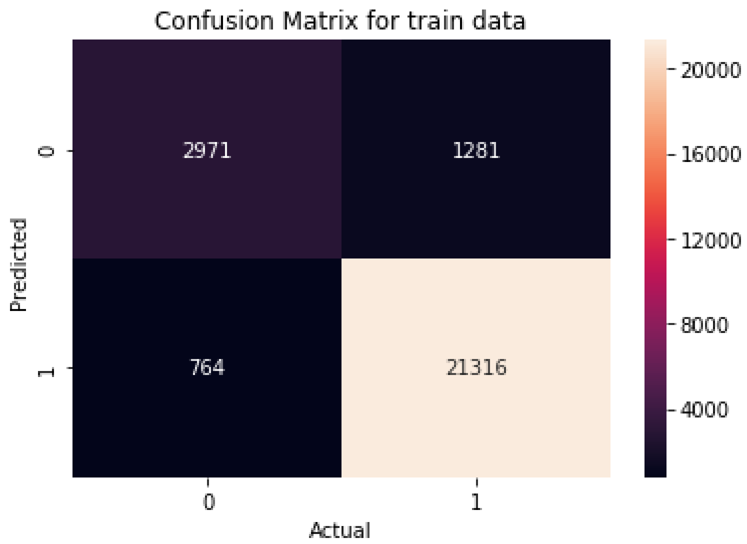
1

In [0]:
```
model_l2_AvgW2V = LogisticRegression(C=optimal_c_l2_AvgW2V,penalty="l1"
)
model_l2_AvgW2V.fit(X_train_vectors,y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, model_l2_AvgW2V.p
redict_proba(X_train_vectors)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, model_l2_AvgW2V.pred
ict_proba(X_test_vectors)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test
_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()
```



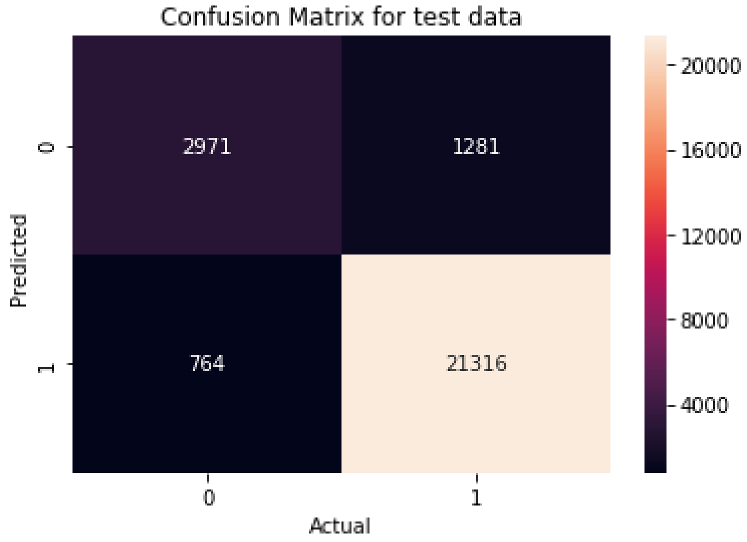
In [0]:
```
conf_matr = confusion_matrix(y_train,model_l2_AvgW2V.predict(X_train_ve
```

```
ctors))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')

plt.title("Confusion Matrix for train data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



In [0]:
```
conf_matr = confusion_matrix(y_test,model_l2_AvgW2V.predict(X_test_vect
ors))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')
```

```
plt.title("Confusion Matrix for test data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



[5.4] Logistic Regression on TFIDF W2V, SET 4

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

In [0]:
```
# Please write all the code with proper documentation
```

In [0]:
```
# Please write all the code with proper documentation

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
```

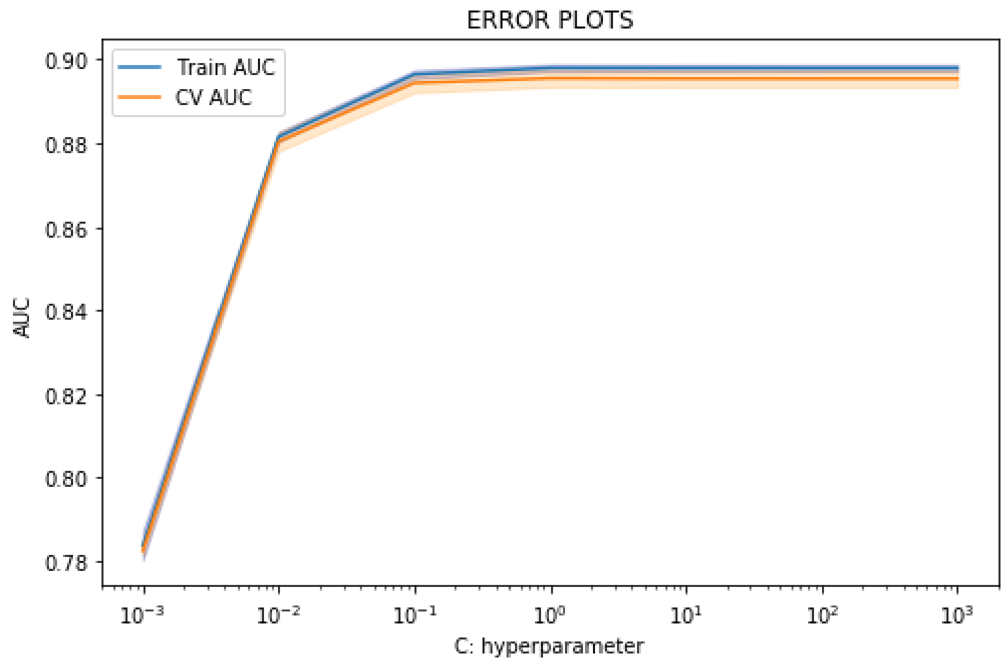
```
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

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

X_train_tfidfw2v = []; # the tfidf-w2v for each sentence/review is stor
ed in this list
row=0;
for sent in tqdm(X_train_sentance): # for each review/sentence
    sent_vec = np.zeros(100) # 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 = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_train_tfidfw2v.append(sent_vec)
    row += 1


X_test_tfidfw2v = []; # the tfidf-w2v for each sentence/review is store
d in this list
row=0;
for sent in tqdm(X_test_sentance): # for each review/sentence
    sent_vec = np.zeros(100) # 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 = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
```

```
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_test_tfidfw2v.append(sent_vec)
    row += 1
```

```
100%|██████████| 61441/61441 [52:13<00:00, 19.61it/s]
100%|██████████| 26332/26332 [22:19<00:00, 20.79it/s]
```

In [0]:

```
c_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

model = LogisticRegression(penalty="l1")
parameters = {'C':c_values}
clf = GridSearchCV(model, parameters, cv=3, scoring='roc_auc',n_jobs=-1
)
clf.fit(X_train_tfidfw2v, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(8,5))
plt.plot(c_values, train_auc, label='Train AUC')

plt.gca().fill_between(c_values,train_auc - train_auc_std,train_auc + t
rain_auc_std,alpha=0.2,color='darkblue')

plt.plot(c_values, cv_auc, label='CV AUC')

plt.gca().fill_between(c_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std
,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.xscale("log")
plt.title("ERROR PLOTS")
plt.show()
```

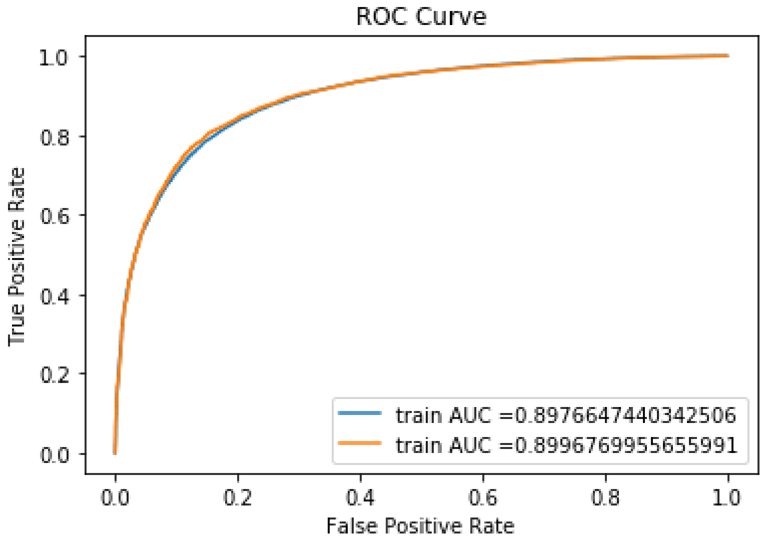


In [0]:
```
auc_l1_tfidfw2v = clf.best_score_
optimal_c_l1_tfidfw2v = clf.best_params_["C"]
print(optimal_c_l1_tfidfw2v)
```

1

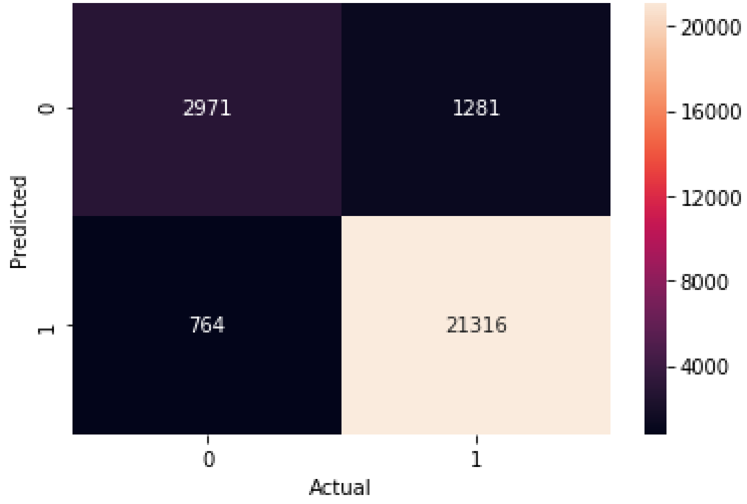
In [0]:
```
model_l1_tfidfw2v = LogisticRegression(C=optimal_c_l1_tfidfw2v,penalty=
"l1")
model_l1_tfidfw2v.fit(X_train_tfidfw2v,y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, model_l1_tfidfw2v
.predict_proba(X_train_tfidfw2v)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, model_l1_tfidfw2v.pr
edict_proba(X_test_tfidfw2v)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test
```

```
_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()
```



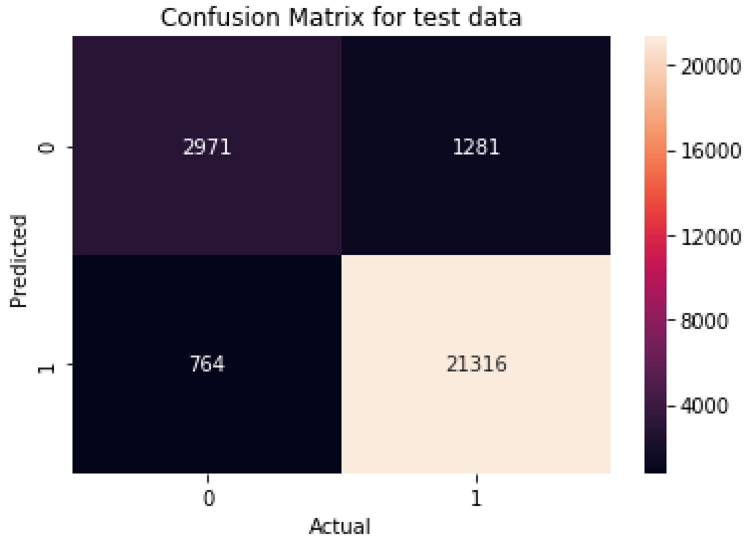
In [0]:
```
conf_matr = confusion_matrix(y_train,model_l1_tfidfw2v.predict(X_train_
tfidfw2v))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')

plt.title("Confusion Matrix for train data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```

Confusion Matrix for train data



In [0]:

```
conf_matr = confusion_matrix(y_test,model_l1_tfidfw2v.predict(X_test_tf
idfw2v))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')

plt.title("Confusion Matrix for test data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```

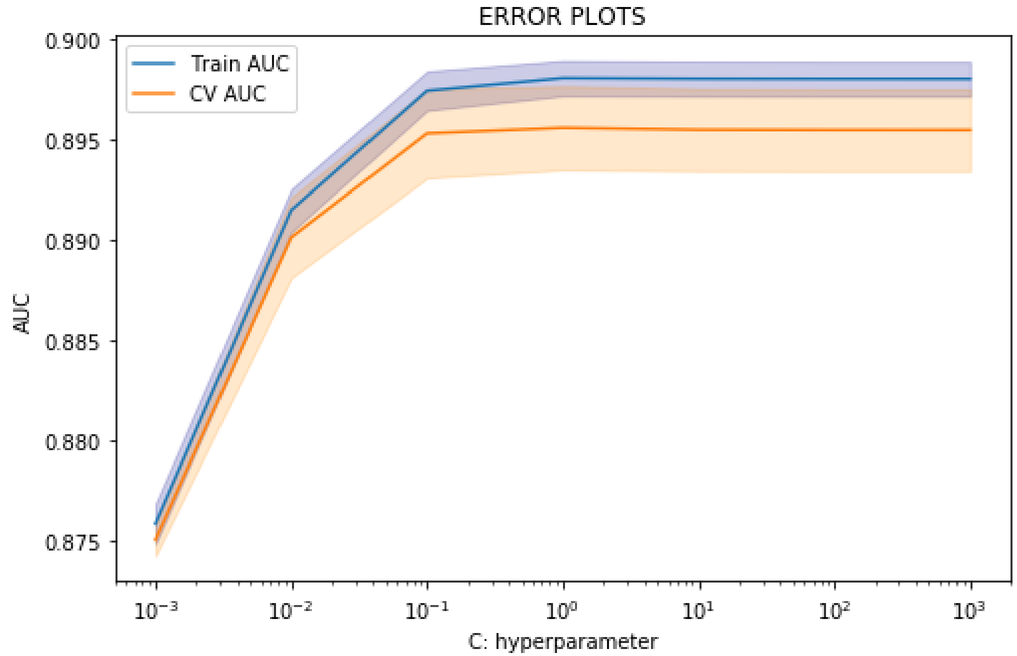


[5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

In [0]:
```
# Please write all the code with proper documentation
```

In [0]:
```
c_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

model = LogisticRegression(penalty="l2")
parameters = {'C':c_values}
clf = GridSearchCV(model, parameters, cv=3, scoring='roc_auc',n_jobs=-1
)
clf.fit(X_train_tfidfw2v, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']

plt.figure(figsize=(8,5))
plt.plot(c_values, train_auc, label='Train AUC')
```

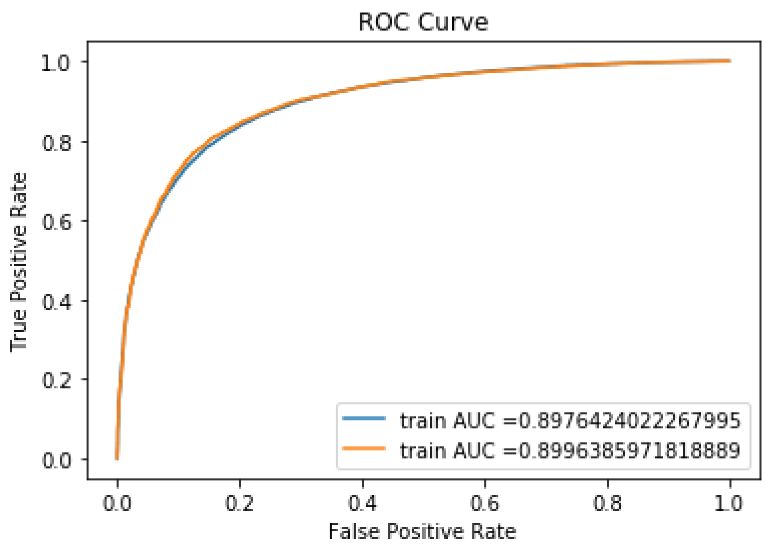
```
plt.gca().fill_between(c_values,train_auc - train_auc_std,train_auc + t
rain_auc_std,alpha=0.2,color='darkblue')

plt.plot(c_values, cv_auc, label='CV AUC')

plt.gca().fill_between(c_values,cv_auc - cv_auc_std,cv_auc + cv_auc_std
,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("C: hyperparameter")
plt.ylabel("AUC")
plt.xscale("log")
plt.title("ERROR PLOTS")
plt.show()
```



In [0]:
```
auc_l2_tfidfw2v = clf.best_score_
optimal_c_l2_tfidfw2v = clf.best_params_["C"]
print(optimal_c_l2_tfidfw2v)
```

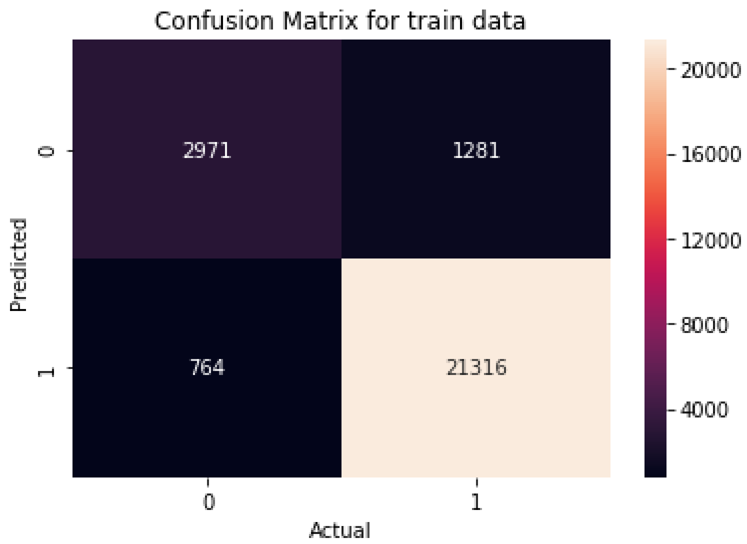
1

In [0]:
```
model_l2_tfidfw2v = LogisticRegression(C=optimal_c_l2_tfidfw2v,penalty=
"l2")
model_l2_tfidfw2v.fit(X_train_tfidfw2v,y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, model_l2_tfidfw2v
.predict_proba(X_train_tfidfw2v)[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, model_l2_tfidfw2v.pr
edict_proba(X_test_tfidfw2v)[:,1])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test
_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.show()
```



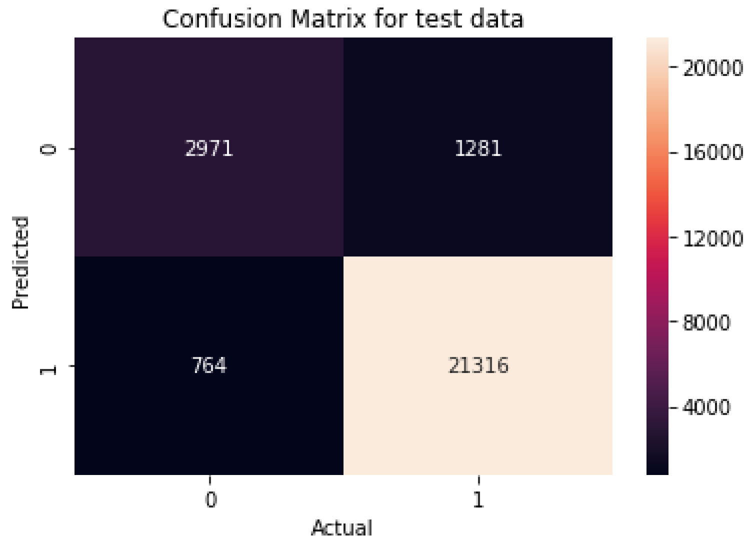
In [0]:
```
conf_matr = confusion_matrix(y_train,model_l2_tfidfw2v.predict(X_train_
```

```
tfidfw2v))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')

plt.title("Confusion Matrix for train data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



In [0]:
```
conf_matr = confusion_matrix(y_test,model_l2_tfidfw2v.predict(X_test_tf
idfw2v))

class_label = [0,1]
df_conf_matr = pd.DataFrame(conf_matr,index=class_label,columns=class_l
abel)

sns.heatmap(df_conf_matrix, annot=True, fmt='d')
```

```
plt.title("Confusion Matrix for test data")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



[6] Conclusions

In [0]:

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

In [0]:

```
from prettytable import PrettyTable

#Model Names
name = ['Log Reg for BOW L1','Log Reg for BOW L2','Log Reg for TFIDF L
1','Log Reg for TFIDF L2'
       ,'Log Reg for AVGW2VEC L1','Log Reg for AVGW2VEC L2',
         'Log Reg for TFIDFW2VEC L1','Log Reg for TFIDFW2VEC L2']

#optimal C values

Optimal_C = [optimal_C_Value, optimal_C_l2_Value, optimal_c_l1_tfidf,
```

```
            optimal_c_l2_tfidf, optimal_c_l1_AvgW2V, optimal_c_l2_AvgW2
V,
            optimal_c_l1_tfidfw2v, optimal_c_l2_tfidfw2v]


#Best AUC for C values

Best_AUC = [Best_AUC_L1_BOW,Best_AUC_L2_BOW,auc_l1_tfidf,auc_l2_tfidf,a
uc_l1_AvgW2V,auc_l2_AvgW2V,auc_l1_tfidfw2v,auc_l2_tfidfw2v]

#Serial Numbers

numbers = [1,2,3,4,5,6,7,8]


ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbers)
ptable.add_column("MODELS",name)
ptable.add_column("Best C",Optimal_C)
ptable.add_column("AUC",Best_AUC)

# Printing the Table
print(ptable)
```

```
+-------+---------------------------+--------+--------------------+
| S.NO. |           MODELS          | Best C |        AUC         |
+-------+---------------------------+--------+--------------------+
|   1   |     Log Reg for BOW L1    |   1    |  0.93445458831526  |
|   2   |     Log Reg for BOW L2    |  0.1   | 0.9414830173466573 |
|   3   |    Log Reg for TFIDF L1   |   1    | 0.9462390122277619 |
|   4   |    Log Reg for TFIDF L2   |   1    | 0.9491751543032072 |
|   5   |  Log Reg for AVGW2VEC L1  |   10   | 0.9167540300259901 |
|   6   |  Log Reg for AVGW2VEC L2  |   1    | 0.9158518043629623 |
|   7   | Log Reg for TFIDFW2VEC L1 |   1    | 0.8955406758703505 |
|   8   | Log Reg for TFIDFW2VEC L2 |   1    | 0.8955466942905246 |
+-------+---------------------------+--------+--------------------+
```