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

from sklearn.model_selection import train_test_split
from sklearn.calibration import CalibratedClassifierCV


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

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

from tqdm import tqdm
import os
```

In [2]:

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [13]:

```
# 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 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 40000""", con
)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 40000""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating(0).
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (40000, 10)
```

Out[13]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	130386240(
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	134697600(
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	121901760(

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 [15]:

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

```
(80668, 7)
```

Out[15]:

	UserId	ProductId	ProfileName	Time	Score	Text	COUNT(*)
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 [16]:

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

Out[16]:

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

In [17]:

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

Out[17]:

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 [18]:

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

Out[18]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199577

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

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 [20]:

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

Out[20]:

```
(37415, 10)
```

In [21]:

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

Out[21]:

```
93.5375
```

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 [22]:

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

display.head()
```

Out[22]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1224892
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1212883

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti

In [0]:

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

In [24]:

```
#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()
```

```
(37415, 10)
```

Out[24]:

```
1    31324
0     6091
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 [25]:

```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

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

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

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

```
Our dogs just love them.  I saw them in a pet store and a tag was attached regarding them being ma
de in China and it satisfied me that they were safe.
==================================================
It's Branston pickle, what is there to say. If you've never tried it you most likely wont like it.
If you grew up in the UK its a staple on cheese of cold meat sandwiches. It's on my lunch sandwich
today! :)
```

```
==================================================
First Impression: The friendly folks over at "Exclusively Dog" heard about my website and sent me
5 of their products to test.<br />Let me just start off by saying that I Love how sweet all of
these treats taste. Dad was/is considering trying one because they look and smell so much like hum
an cookies. Plus the ingredients are very straight forward, they are probably healthier than most
the stuff Mom eats... But there in lies the problem. Dad thinks that they are too sweet for a pupp
y of any age. The second ingredient in almost all of them is sugar. As we all know puppies have a
hard time processing sugar, and just like humans can develop diabetes.<br /><br />Conclusion: Your
puppy is nearly guaranteed to LOVE the taste. However these should only be used as an occasional t
reat! If you were to feed your puppies these sugary sweet morsels every day, they would soon plump
up. If you puppy is already overweight or does not exercise regularly, you may want to think
twice. On the PRO side they are all natural, with no animal bi-products! 3 out of 4 paws, because
Dad made me! If we were judging on taste alone they would be a 4.
==================================================
It is hard to find candy that is overly sweet. My wife and Granddaughter both love Pink Grapefruit
anyway and Pink Grapefruit candy has some of the tang of real grapefruit which cuts down on the sw
eetness a bit.<br />I did take away one star because I think they have a bit too much of sugar coa
ting on the pieces but you can scrape some of it off to make it less sweet.<br />My wife uses the
pieces when she has a low sugar spell since she is diabetic and sometimes when she has her insulin
injections and doesn't eat quickly enough after that her blood sugar drops too low. Since I bought
this she hasn't had that problem, but has to guard her supply from my Granddaughter though.<br />I
have bought a pack for myself as well since I don't eat candy that often since I don't like overly
sweet candy. This candy tastes good to me. I want to try the fruit salad next time just to have so
me change in taste. It has lime, grapefruit, lemon, orange, cherry and passion fruit and I like al
l of those flavors except cherry. But my wife likes cherry flavor so I can give those to her. Wish
they had watermelon instead of cherry in that mix but its no big deal.
==================================================
```

In [26]:

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

print(sent_0)
```

```
Our dogs just love them.  I saw them in a pet store and a tag was attached regarding them being ma
de in China and it satisfied me that they were safe.
```

In [27]:

```
# 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)
```

```
Our dogs just love them.  I saw them in a pet store and a tag was attached regarding them being ma
de in China and it satisfied me that they were safe.
==================================================
It's Branston pickle, what is there to say. If you've never tried it you most likely wont like it.
If you grew up in the UK its a staple on cheese of cold meat sandwiches. It's on my lunch sandwich
today! :)
==================================================
First Impression: The friendly folks over at "Exclusively Dog" heard about my website and sent me
```

```
5 of their products to test.Let me just start off by saying that I Love how sweet all of these tre
ats taste. Dad was/is considering trying one because they look and smell so much like human cookie
s. Plus the ingredients are very straight forward, they are probably healthier than most the stuff
Mom eats... But there in lies the problem. Dad thinks that they are too sweet for a puppy of any a
ge. The second ingredient in almost all of them is sugar. As we all know puppies have a hard time
processing sugar, and just like humans can develop diabetes.Conclusion: Your puppy is nearly guara
nteed to LOVE the taste. However these should only be used as an occasional treat! If you were to
feed your puppies these sugary sweet morsels every day, they would soon plump up. If you puppy is
already overweight or does not exercise regularly, you may want to think twice. On the PRO side th
ey are all natural, with no animal bi-products! 3 out of 4 paws, because Dad made me! If we were j
udging on taste alone they would be a 4.
==================================================
It is hard to find candy that is overly sweet. My wife and Granddaughter both love Pink Grapefruit
anyway and Pink Grapefruit candy has some of the tang of real grapefruit which cuts down on the sw
eetness a bit.I did take away one star because I think they have a bit too much of sugar coating o
n the pieces but you can scrape some of it off to make it less sweet.My wife uses the pieces when
she has a low sugar spell since she is diabetic and sometimes when she has her insulin injections
and doesn't eat quickly enough after that her blood sugar drops too low. Since I bought this she h
asn't had that problem, but has to guard her supply from my Granddaughter though.I have bought a p
ack for myself as well since I don't eat candy that often since I don't like overly sweet candy. T
his candy tastes good to me. I want to try the fruit salad next time just to have some change in t
aste. It has lime, grapefruit, lemon, orange, cherry and passion fruit and I like all of those fla
vors except cherry. But my wife likes cherry flavor so I can give those to her. Wish they had wate
rmelon instead of cherry in that mix but its no big deal.
```

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 [29]:

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

```
First Impression: The friendly folks over at "Exclusively Dog" heard about my website and sent me
5 of their products to test.<br />Let me just start off by saying that I Love how sweet all of
these treats taste. Dad was/is considering trying one because they look and smell so much like hum
an cookies. Plus the ingredients are very straight forward, they are probably healthier than most
the stuff Mom eats... But there in lies the problem. Dad thinks that they are too sweet for a pupp
y of any age. The second ingredient in almost all of them is sugar. As we all know puppies have a
hard time processing sugar, and just like humans can develop diabetes.<br /><br />Conclusion: Your
puppy is nearly guaranteed to LOVE the taste. However these should only be used as an occasional t
reat! If you were to feed your puppies these sugary sweet morsels every day, they would soon plump
up. If you puppy is already overweight or does not exercise regularly, you may want to think
twice. On the PRO side they are all natural, with no animal bi-products! 3 out of 4 paws, because
Dad made me! If we were judging on taste alone they would be a 4.
==================================================
```

In [30]:

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

```
Our dogs just love them.  I saw them in a pet store and a tag was attached regarding them being ma
```

de in China and it satisfied me that they were safe.

In [31]:

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

```
First Impression The friendly folks over at Exclusively Dog heard about my website and sent me 5 o
f their products to test br Let me just start off by saying that I Love how sweet all of these tre
ats taste Dad was is considering trying one because they look and smell so much like human cookies
Plus the ingredients are very straight forward they are probably healthier than most the stuff Mom
eats But there in lies the problem Dad thinks that they are too sweet for a puppy of any age The s
econd ingredient in almost all of them is sugar As we all know puppies have a hard time processing
sugar and just like humans can develop diabetes br br Conclusion Your puppy is nearly guaranteed t
o LOVE the taste However these should only be used as an occasional treat If you were to feed your
puppies these sugary sweet morsels every day they would soon plump up If you puppy is already
overweight or does not exercise regularly you may want to think twice On the PRO side they are all
natural with no animal bi products 3 out of 4 paws because Dad made me If we were judging on taste
alone they would be a 4
```

In [0]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [33]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 37415/37415 [00:15<00:00, 2464.32it/s]
```

In [0]:

```
preprocessed_reviews[1500]
```

Out[0]:

```
'first impression friendly folks exclusively dog heard website sent products test let start saying
love sweet treats taste dad considering trying one look smell much like human cookies plus
ingredients straight forward probably healthier stuff mom eats lies problem dad thinks sweet puppy
age second ingredient almost sugar know puppies hard time processing sugar like humans develop
diabetes conclusion puppy nearly guaranteed love taste however used occasional treat feed puppies
sugary sweet morsels every day would soon plump puppy already overweight not exercise regularly
may want think twice pro side natural no animal bi products paws dad made judging taste alone
would'
```

In [34]:

```
# 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())
```

```
100%|██████████| 37415/37415 [00:10<00:00, 3725.33it/s]
```

[3.2] Preprocessing Review Summary

In [0]:

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

[4] Featurization

[4.1] BAG OF WORDS

In [0]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

```
some feature names  ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaa', 'aaaaaaaaaaaaaa',
'aaaaaaahhhhhh', 'aaaaaawwwwwwwww', 'aaaand', 'aaah']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (37415, 35636)
the number of unique words  35636
```

[4.2] Bi-Grams and n-Grams.

In [0]:

```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-
learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_s
hape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (37415, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [0]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()[0:10])
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.get_shape()[
1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able buy', 'able chew',
'able drink', 'able eat', 'able enjoy', 'able figure', 'able find', 'able finish']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (37415, 22294)
the number of unique words including both unigrams and bigrams  22294
```

[4.4] Word2Vec

In [0]:

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

In [0]:

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.
```

```
# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=Tr
ue)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to train your
own w2v ")
```

```
[('awesome', 0.8273319602012634), ('fantastic', 0.7991375923156738), ('good', 0.7857114672660828),
('wonderful', 0.7802661657333374), ('excellent', 0.7660344243049622), ('perfect',
0.7592657208442688), ('amazing', 0.7589268684387207), ('terrific', 0.7399769425392151), ('decent',
0.7050701975822449), ('nice', 0.6778730154037476)]
==================================================
[('greatest', 0.7281339168548584), ('closest', 0.716359555721283), ('best', 0.7081298828125), ('iv
e', 0.6644027829170227), ('nastiest', 0.6624605655670166), ('tastiest', 0.6618518233299255),
('foul', 0.6589347720146179), ('experienced', 0.6563234925270081), ('awful', 0.6389572024345398),
('horrible', 0.6330024600028992)]
```

In [0]:

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

```
number of words that occured minimum 5 times  11636
sample words  ['dogs', 'love', 'saw', 'pet', 'store', 'tag', 'attached', 'regarding', 'made', 'chi
na', 'satisfied', 'safe', 'loves', 'chicken', 'product', 'wont', 'buying', 'anymore', 'hard',
'find', 'products', 'usa', 'one', 'isnt', 'bad', 'good', 'take', 'chances', 'till', 'know',
'going', 'imports', 'available', 'victor', 'traps', 'unreal', 'course', 'total', 'fly', 'pretty',
'stinky', 'right', 'nearby', 'used', 'bait', 'seasons', 'ca', 'not', 'beat', 'great']
```

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

[4.4.1.1] Avg W2v

In [0]:

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

```
print(len(sent_vectors[0]))
```

```
100%|██████████| 37415/37415 [01:17<00:00, 485.52it/s]
```

```
37415
50
```

[4.4.1.2] TFIDF weighted W2v

In [0]:

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

In [0]:

```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████| 37415/37415 [14:19<00:00, 43.56it/s]
```

[5] Assignment 7: SVM

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

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

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

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

4. **Feature importance**

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

5. **Feature engineering**

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

6. **Representation of results**

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

7. **Conclusion**

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

Note: Data Leakage

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

Applying SVM

[5.1] Linear SVM

[5.1.1] Applying Linear SVM on BOW, SET 1

In [0]:

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

In [35]:

```
final['PreprocessedText'] = preprocessed_reviews
final['PreprocessedSummary'] = preprocessed_summary
final['TotalText'] = final['PreprocessedText'] + final['PreprocessedSummary']
final.head()
```

Out[35]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
22621	24751	2734888454	A1C298ITT645B6	Hugh G. Pritchard	0	0	1	1195
22620	24750	2734888454	A13ISQV0U9GZIC	Sandikaye	1	1	0	1192
2546	2774	B00002NCJC	A196AJHU9EASJN	Alex Chaffee	0	0	1	1282
2547	2775	B00002NCJC	A13RRPGE79XFFH	reader48	0	0	1	1281
1145	1244	B00002Z754	A3B8RCEI0FXFI6	B G Chase	10	10	1	9622

In [0]:

```
#sorting based on time
final["Time"] = pd.to_datetime(final["Time"],unit="s")
final = final.sort_values( by ="Time")
```

In [37]:

```
#splitting data into train , test
X_train, X_test, y_train, y_test = train_test_split(
    final['TotalText'], final['Score'], test_size=0.30, random_state=0)

X_train_cv, X_cv, y_train_cv, y_cv = train_test_split(X_train, y_train, test_size=0.33)

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

```
(26190,) (26190,)
(17547,) (17547,)
(8643,) (8643,)
(11225,) (11225,)
```

In [86]:

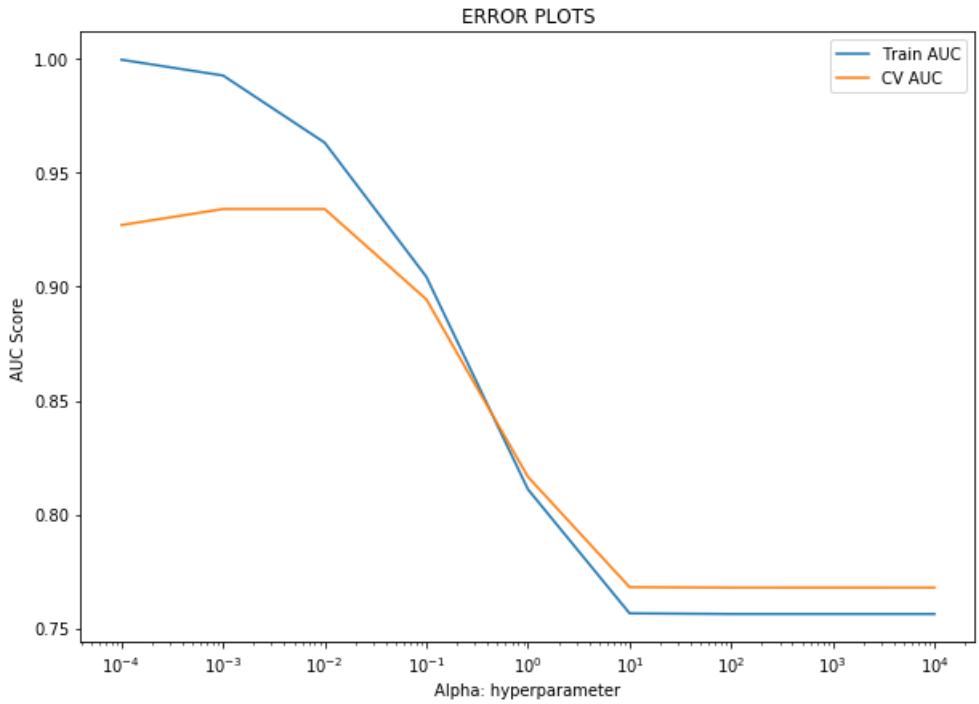
```
count_vect = CountVectorizer()
X_train_cv_BOW = count_vect.fit_transform(X_train_cv)
X_cv_BOW = count_vect.transform(X_cv)
X_test_BOW = count_vect.transform(X_test)

print("After vectorizations")
print(X_train_cv_BOW.shape, y_train_cv.shape)
print(X_cv_BOW.shape, y_cv.shape)
print(X_test_BOW.shape, y_test.shape)
```

```
After vectorizations
(17547, 39328) (17547,)
```

```
(8643, 39328) (8643,)
(6000, 39328) (6000,)
```

In [79]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_auc_score
train_auc = []
cv_auc = []

# Creating alpha values in the range from 10^-4
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

from sklearn.model_selection import GridSearchCV

for i in range(len(alpha)):
  sgd = SGDClassifier(loss="hinge",alpha=alpha[i])
  clf = CalibratedClassifierCV(sgd,cv=10,method="isotonic")
  clf.fit(X_train_cv_BOW,y_train_cv)


  y_train_pred =  clf.predict_proba(X_train_cv_BOW)[:,1]
  y_cv_pred =  clf.predict_proba(X_cv_BOW)[:,1]

  train_auc.append(roc_auc_score(y_train_cv,y_train_pred))
  cv_auc.append(roc_auc_score(y_cv, y_cv_pred))


plt.figure(figsize=(10,7))
plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC Score")
plt.title("ERROR PLOTS")
plt.xscale("log")
plt.show()
```



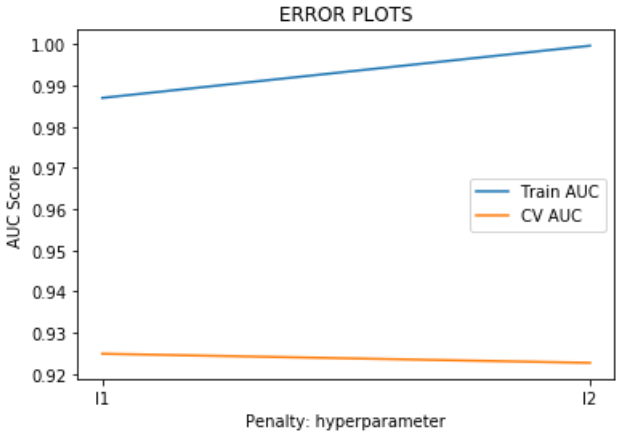
In [0]:

```
optimal_alpha_bow = 0.0001
```

In [0]:

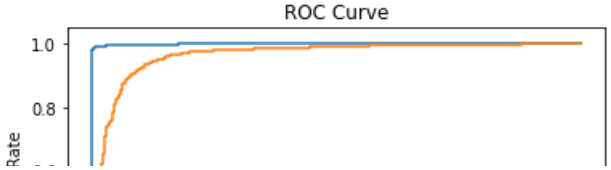
```
train_auc = []
cv_auc = []
```

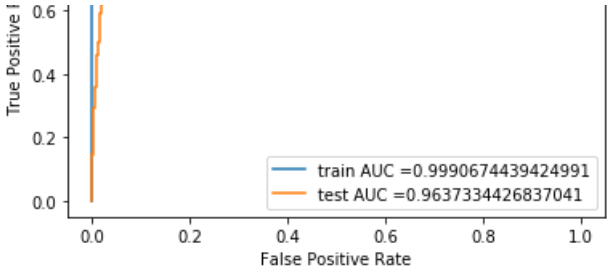
```
penalty = ["l1","l2"]
for i in range(len(penalty)):
    sgd = SGDClassifier(loss="hinge",penalty=penalty[i],alpha=optimal_alpha_bow)
    model = CalibratedClassifierCV(sgd,cv=10,method="isotonic")
    model.fit(X_train_cv_BOW,y_train_cv)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    y_train_pred =  model.predict_proba(X_train_cv_BOW)[:,1]
    y_cv_pred =  model.predict_proba(X_cv_BOW)[:,1]

    train_auc.append(roc_auc_score(y_train_cv,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(penalty, train_auc, label='Train AUC')
plt.plot(penalty, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("Penalty: hyperparameter")
plt.ylabel("AUC Score")
plt.title("ERROR PLOTS")
# plt.xscale("log")
plt.show()
```



In [87]:

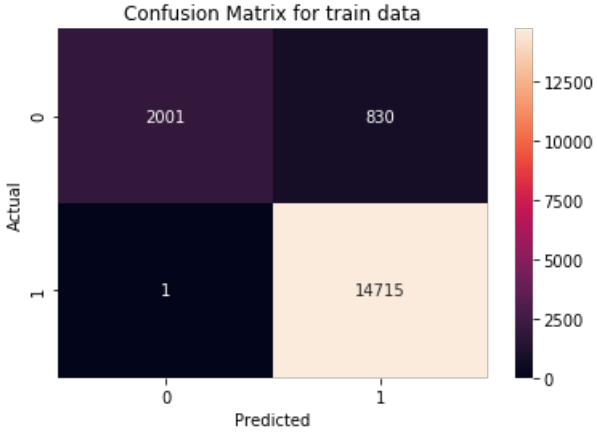
```
from sklearn.metrics import roc_curve, auc


sgd_model = SGDClassifier(alpha=optimal_alpha_bow,penalty="l2",loss="hinge",class_weight =
'balanced')
calb_model = CalibratedClassifierCV(sgd_model,method="sigmoid")
calb_model.fit(X_train_cv_BOW, y_train_cv)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train_cv, calb_model.predict_proba(X_train_cv_BOW)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, calb_model.predict_proba(X_test_BOW)[:,1])

model_optimal_sgd_bow_train = auc(train_fpr, train_tpr)
model_optimal_sgd_bow_test = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_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.title("ROC Curve")
plt.show()
```





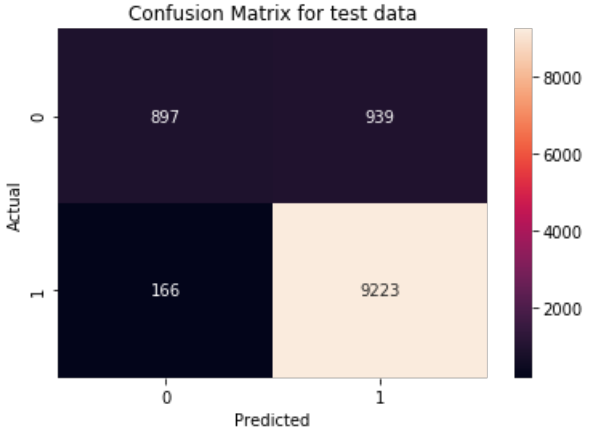
In [0]:

```
conf_matrix = confusion_matrix(y_train_cv, calb_model.predict(X_train_cv_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 train data")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [0]:

```
conf_matrix = confusion_matrix(y_test, calb_model.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("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [88]:

```
#Findig top Features
```

```
clasf = SGDClassifier(penalty='l2',alpha=optimal_alpha_bow,loss='hinge',class_weight = 'balanced')
clasf.fit(X_train_cv_BOW,y_train_cv)

top_pos_features = (-clasf.coef_[0,:]).argsort() #Here -ve sign indicates order in descending
order
top_neg_features = (clasf.coef_[0,:]).argsort()

top_pos_features = np.take(count_vect.get_feature_names(),top_pos_features[:10])
top_neg_features = np.take(count_vect.get_feature_names(),top_neg_features[:10])

print('The top 10 important features from the positive class is: \n')
print(top_pos_features)

print('\nThe top 10 important features from the negative class is: \n')
print(top_neg_features)
```

```
The top 10 important features from the positive class is:

['pleasantly' 'yummy' 'delicious' 'alone' 'worried' 'excellent' 'hooked'
 'amazing' 'lovely' 'buygood']

The top 10 important features from the negative class is:

['worst' 'rip' 'coffeenot' 'horrible' 'singleanyone' 'productnot'
 'shippingshipping' 'disappointing' 'cacao' 'lighttasty']
```

[5.1.2] Applying Linear SVM on TFIDF, SET 2

In [0]:

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

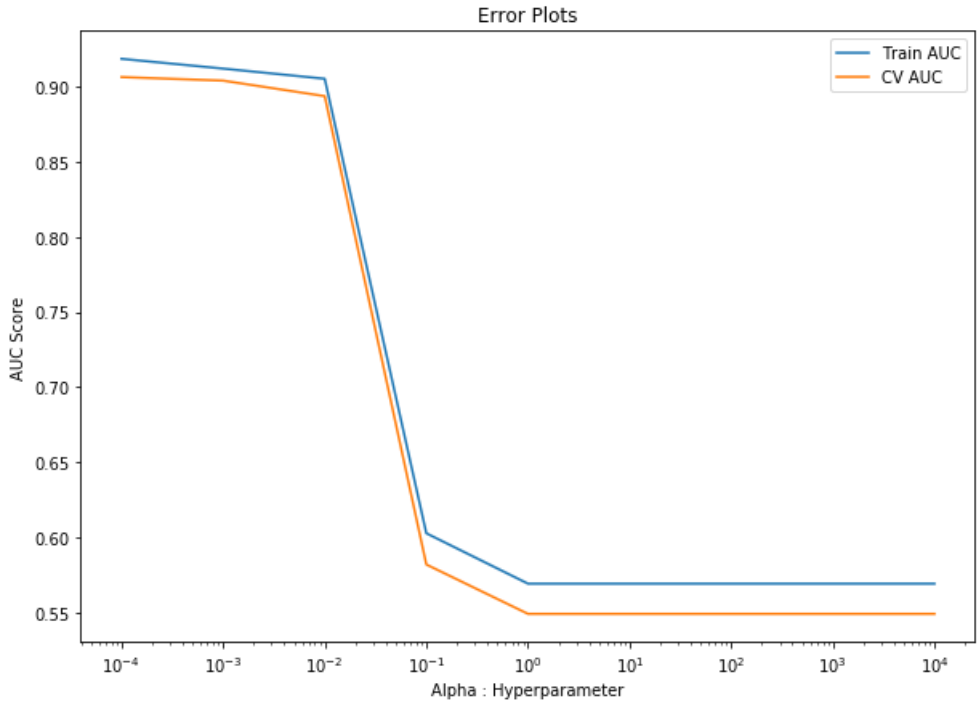
In [0]:

```
tfidf_vect = TfidfVectorizer(min_df=10 , max_features=500)

X_train_cv_tfidf = tfidf_vect.fit_transform(X_train_cv)
X_cv_tfidf = tfidf_vect.transform(X_cv)
X_test_tfidf = tfidf_vect.transform(X_test)
```

In [43]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_auc_score
train_auc = []
cv_auc = []


# Creating alpha values in the range from 10^-4
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

from sklearn.model_selection import GridSearchCV

for i in range(len(alpha)):
  sgd = SGDClassifier(loss="hinge" , alpha=alpha[i])
  clf = CalibratedClassifierCV(sgd , cv= 10 , method='isotonic')
  clf.fit(X_train_cv_tfidf,y_train_cv)

  y_train_pred = clf.predict_proba(X_train_cv_tfidf)[:,1]
  y_cv_pred = clf.predict_proba(X_cv_tfidf)[:,1]

  train_auc.append(roc_auc_score(y_train_cv,y_train_pred))
  cv_auc.append(roc_auc_score(y_cv,y_cv_pred))


plt.figure(figsize=(10, 7))
plt.plot(alpha,train_auc,label='Train AUC')
plt.plot(alpha,cv_auc,label = 'CV AUC')
plt.legend()
plt.xlabel("Alpha : Hyperparameter")
plt.ylabel("AUC Score")
```

```
plt.title("Error Plots")
plt.xscale("log")
plt.show()
```

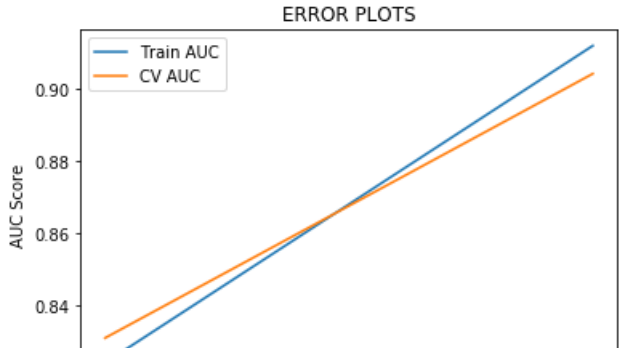


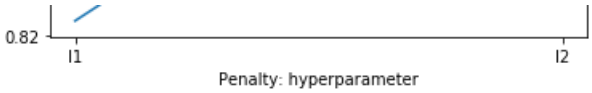
In [0]:

```
optimal_alpha_tfidf = 0.001
```

In [45]:

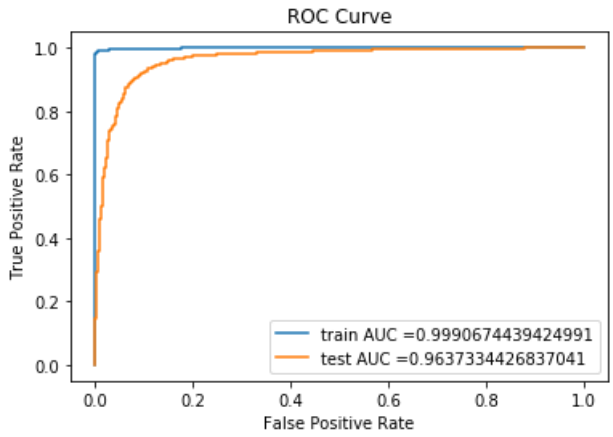
```
train_auc = []
cv_auc = []
penalty = ["l1","l2"]

for i in range(len(penalty)):
  sgd = SGDClassifier(loss="hinge" , penalty=penalty[i],alpha = optimal_alpha_tfidf)
  model = CalibratedClassifierCV(sgd,cv=10,method="isotonic")
  model.fit(X_train_cv_tfidf,y_train_cv)

  y_train_pred = model.predict_proba(X_train_cv_tfidf)[:,1]
  y_cv_pred = model.predict_proba(X_cv_tfidf)[:,1]

  train_auc.append(roc_auc_score(y_train_cv,y_train_pred))
  cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

plt.plot(penalty, train_auc, label='Train AUC')
plt.plot(penalty, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("Penalty: hyperparameter")
plt.ylabel("AUC Score")
plt.title("ERROR PLOTS")
# plt.xscale("log")
plt.show()
```





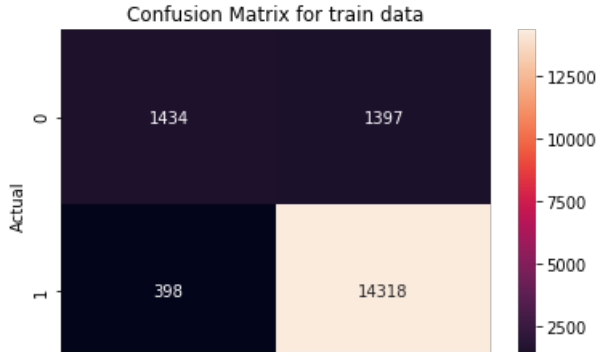
In [90]:

```
from sklearn.metrics import roc_curve , auc

sgd_model  = SGDClassifier(alpha = optimal_alpha_bow,penalty="l2" , loss="hinge",class_weight = 'ba
lanced')
cal_model = CalibratedClassifierCV(sgd_model,method="sigmoid")
cal_model.fit(X_train_cv_tfidf,y_train_cv)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train_cv, calb_model.predict_proba(X_train_cv_BOW)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, calb_model.predict_proba(X_test_BOW)[:,1])

model_optimal_sgd_tfidf_train = auc(train_fpr, train_tpr)
model_optimal_sgd_tfidf_test = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_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.title("ROC Curve")
plt.show()
```



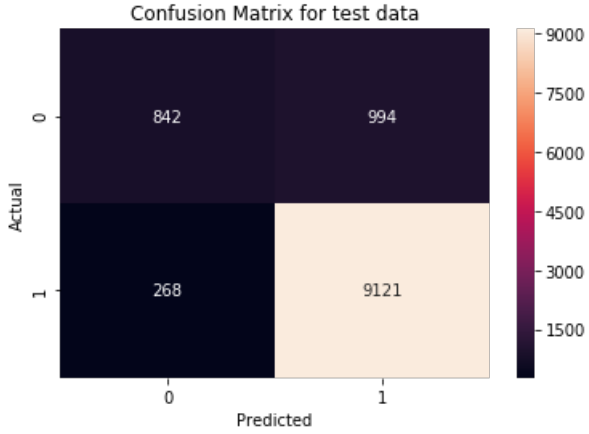
In [0]:

```
conf_matrix = confusion_matrix(y_train_cv,cal_model.predict(X_train_cv_tfidf))
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("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [0]:

```
conf_matrix = confusion_matrix(y_test,cal_model.predict(X_test_tfidf))
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("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [0]:

```
#Findig top Features

clasf = SGDClassifier(penalty='l2',alpha=optimal_alpha_tfidf,loss='hinge')
clasf.fit(X_train_cv_tfidf,y_train_cv)

top_pos_features = (-clasf.coef_[0,:]).argsort() #Here -ve sign indicates order in descending
order
top_neg_features = (clasf.coef_[0,:]).argsort()

top_pos_features = np.take(tfidf_vect.get_feature_names(),top_pos_features[:10])
top_neg_features = np.take(tfidf_vect.get_feature_names(),top_neg_features[:10])


print('The top 10 important features from the positive class is: \n')
print(top_pos_features)

print('\nThe top 10 important features from the negative class is: \n')
print(top_neg_features)
```

```
The top 10 important features from the positive class is:

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

The top 10 important features from the negative class is:

['disappointed' 'money' 'not' 'bad' 'away' 'nothing' 'thought' 'received'
 'even' 'maybe']
```

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

In [0]:

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

In [92]:

In [92]:

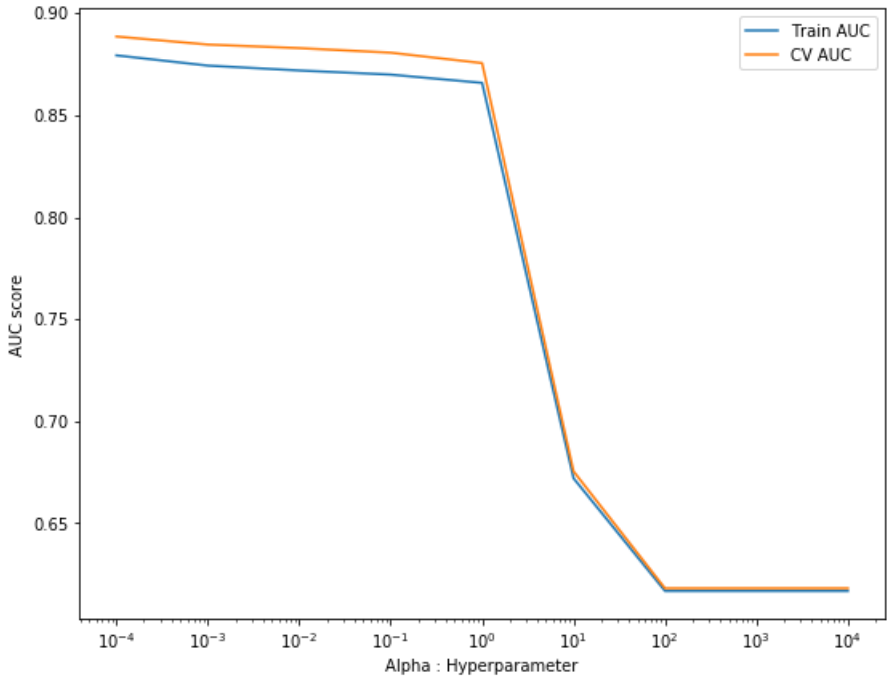
```
# Train your own Word2Vec model using your own text corpus

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

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

X_cv_sentance=[]
for sentance in X_cv:
    X_cv_sentance.append(sentance.split())

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


w2v_words = list(w2v_model.wv.vocab)
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)


X_cv_vectors = []
for sent in X_cv_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_cv_vectors.append(sent_vec)
```

```
number of words that occured minimum 5 times  8317
```

In [48]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []

alpha = [0.0001,0.001,0.01,0.1,1.0,10,100,1000,10000]

for i in range(len(alpha)):
```

```
for i in range(len(alpha)):
  sgd = SGDClassifier(loss="hinge", alpha = alpha[i])
  clf = CalibratedClassifierCV(sgd , cv =10 , method='sigmoid')
  clf.fit(X_train_vectors,y_train_cv)

  y_train_pred = clf.predict_proba(X_train_vectors)[:,1]
  y_cv_pred = clf.predict_proba(X_cv_vectors)[:,1]

  train_auc.append(roc_auc_score(y_train_cv,y_train_pred))
  cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

plt.figure(figsize=(9,7))
plt.plot(alpha, train_auc , label = 'Train AUC')
plt.plot(alpha , cv_auc , label = 'CV AUC')
plt.legend()
plt.xlabel('Alpha : Hyperparameter')
plt.ylabel('AUC score')
plt.xscale("log")
plt.show()
```



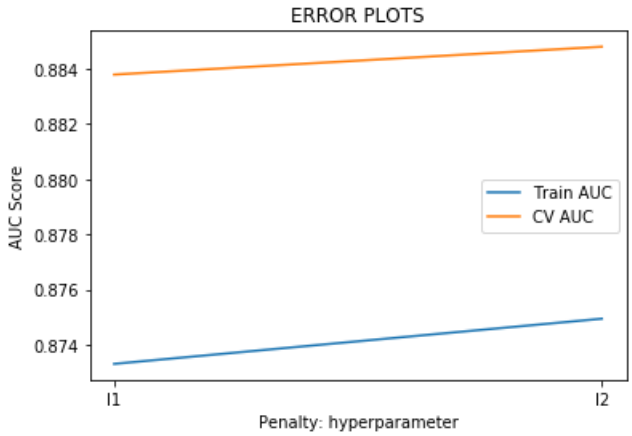
In [0]:

```
optimal_alpha_AvgW2V = 0.1
```

In [50]:

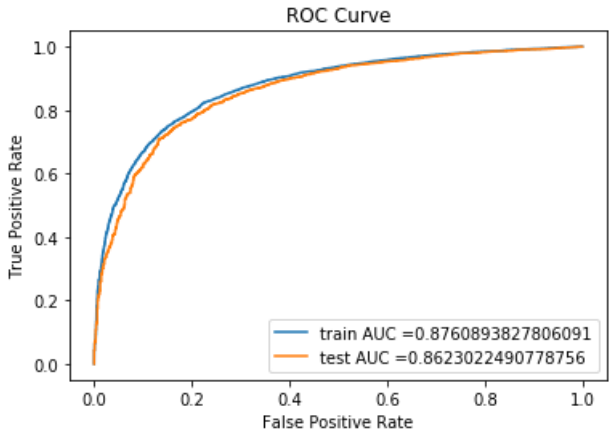
```
train_auc = []
cv_auc = []
penalty = ["l1","l2"]

for i in range(len(penalty)):
  sgd = SGDClassifier(loss="hinge" , penalty=penalty[i],alpha = optimal_alpha_tfidf)
  model = CalibratedClassifierCV(sgd,cv=10,method="isotonic")
  model.fit(X_train_vectors,y_train_cv)

  y_train_pred = model.predict_proba(X_train_vectors)[:,1]
  y_cv_pred = model.predict_proba(X_cv_vectors)[:,1]

  train_auc.append(roc_auc_score(y_train_cv,y_train_pred))
  cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

plt.plot(penalty, train_auc, label='Train AUC')
plt.plot(penalty, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("Penalty: hyperparameter")
plt.ylabel("AUC Score")
plt.title("ERROR PLOTS")
# plt.xscale("log")
```

```
plt.show()
```



In [93]:

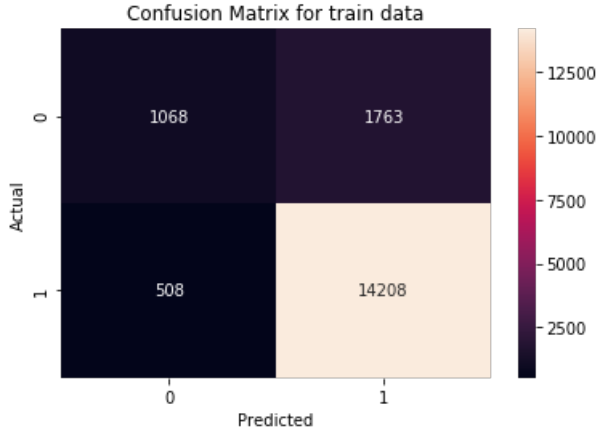
```
from sklearn.metrics import roc_curve , auc

sgd_model  = SGDClassifier(alpha = optimal_alpha_tfidf,penalty="l2" , loss="hinge",class_weight = '
balanced')
cal_model = CalibratedClassifierCV(sgd_model,method="sigmoid")
cal_model.fit(X_train_vectors,y_train_cv)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train_cv, cal_model.predict_proba(X_train_vectors)[
:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, cal_model.predict_proba(X_test_vectors)[:,1])

model_optimal_sgd_avgw2v_train = auc(train_fpr, train_tpr)
model_optimal_sgd_avgw2v_test = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_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.title("ROC Curve")
plt.show()
```



In [0]:

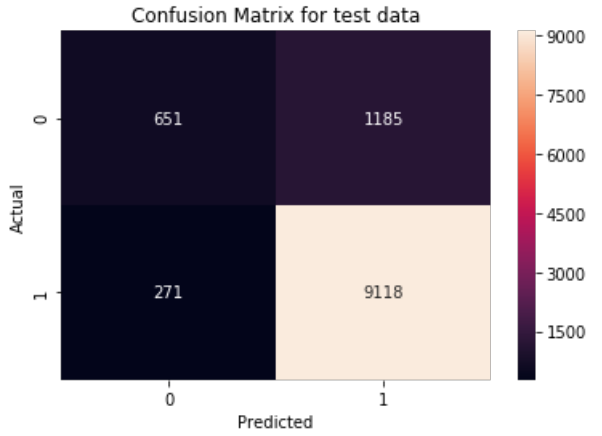
```
conf_matrix = confusion_matrix(y_train_cv,cal_model.predict(X_train_vectors))
class_label = [0,1]
df_conf_matrix = pd.DataFrame(conf_matrix, index = class_label , columns = class_label)
sns.heatmap(df_conf_matrix,annot=True , fmt = 'd')
plt.title("Confusion Matrix for train data")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
plt.show()
```



In [0]:

```
conf_matrix = confusion_matrix(y_test,cal_model.predict(X_test_vectors))
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("Predicted")
plt.ylabel("Actual")
plt.show()
```



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

In [0]:

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

In [95]:

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

model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train_cv)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))

tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

X_train_tfidfw2v = []; # the tfidf-w2v for each sentence/review is stored 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/review
    for word in sent: # for each word in a review/sentence
```

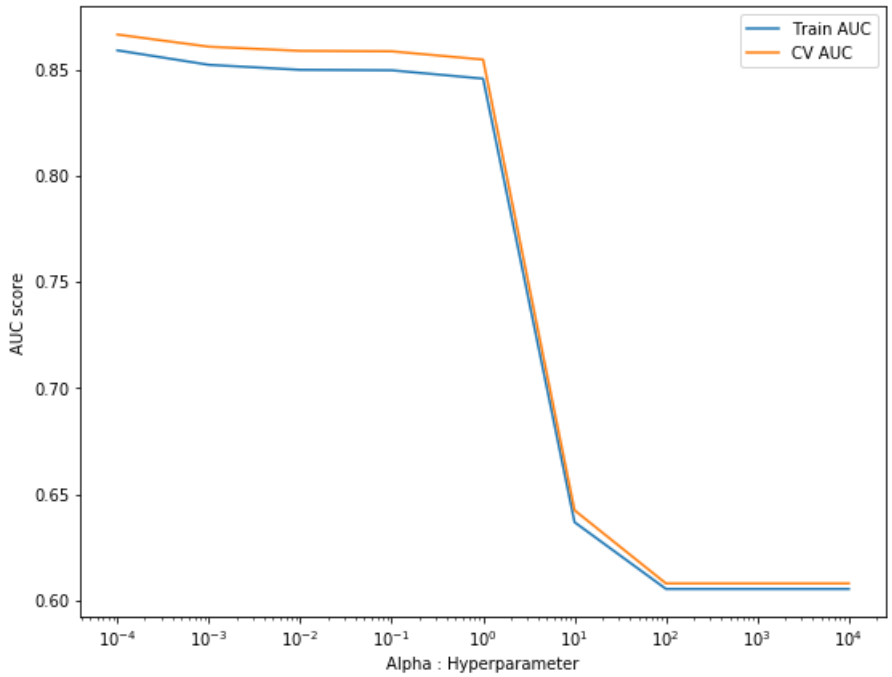
```
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]

            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_train_tfidfw2v.append(sent_vec)
    row += 1


X_test_tfidfw2v = []; # the tfidf-w2v for each sentence/review is stored 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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]

            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_test_tfidfw2v.append(sent_vec)
    row += 1



X_cv_tfidfw2v = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(X_cv_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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]

            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_cv_tfidfw2v.append(sent_vec)
    row += 1
```

```
100%|██████████| 17547/17547 [07:22<00:00, 35.13it/s]
100%|██████████| 6000/6000 [02:28<00:00, 32.13it/s]
100%|██████████| 8643/8643 [03:39<00:00, 39.43it/s]
```

In [53]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []

alpha = [0.0001,0.001,0.01,0.1,1.0,10,100,1000,10000]

for i in range(len(alpha)):
  sgd = SGDClassifier(loss="hinge", alpha = alpha[i])
  clf = CalibratedClassifierCV(sgd , cv =10 , method='sigmoid')
  clf.fit(X_train_tfidfw2v,y_train_cv)

  y_train_pred = clf.predict_proba(X_train_tfidfw2v)[:,1]
  y_cv_pred = clf.predict_proba(X_cv_tfidfw2v)[:,1]

  train_auc.append(roc_auc_score(y_train_cv,y_train_pred))
  cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

plt.figure(figsize=(9,7))
plt.plot(alpha, train_auc , label = 'Train AUC')
plt.plot(alpha , cv_auc , label = 'CV AUC')
```

```
plt.legend()
plt.xlabel('Alpha : Hyperparameter')
plt.ylabel('AUC score')
plt.xscale("log")
plt.show()
```

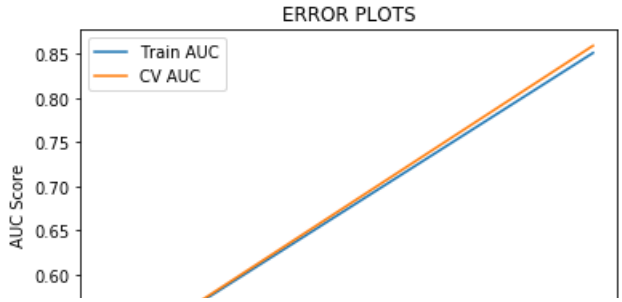


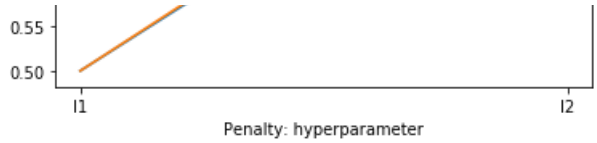
In [0]:

```
optimal_alpha_tfidfW2V = 0.1
```

In [55]:

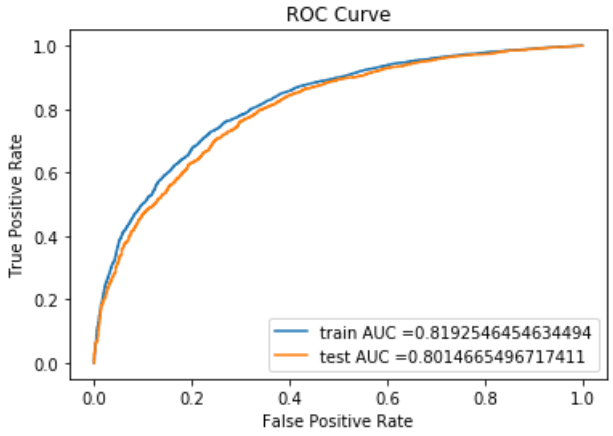
```
train_auc = []
cv_auc = []
penalty = ["l1","l2"]

for i in range(len(penalty)):
  sgd = SGDClassifier(loss="hinge" , penalty=penalty[i],alpha = optimal_alpha_tfidfW2V)
  model = CalibratedClassifierCV(sgd,cv=10,method="isotonic")
  model.fit(X_train_tfidfw2v,y_train_cv)

  y_train_pred = model.predict_proba(X_train_tfidfw2v)[:,1]
  y_cv_pred = model.predict_proba(X_cv_tfidfw2v)[:,1]

  train_auc.append(roc_auc_score(y_train_cv,y_train_pred))
  cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

plt.plot(penalty, train_auc, label='Train AUC')
plt.plot(penalty, cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("Penalty: hyperparameter")
plt.ylabel("AUC Score")
plt.title("ERROR PLOTS")
# plt.xscale("log")
plt.show()
```





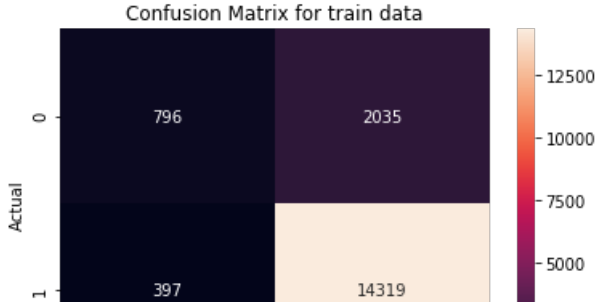
In [96]:

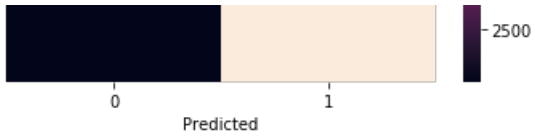
```
from sklearn.metrics import roc_curve , auc

sgd_model  = SGDClassifier(alpha = optimal_alpha_tfidfW2V,penalty="l2" , loss="hinge",class_weight
= 'balanced')
cal_model = CalibratedClassifierCV(sgd_model,method="sigmoid")
cal_model.fit(X_train_tfidfw2v,y_train_cv)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

train_fpr, train_tpr, thresholds = roc_curve(y_train_cv, cal_model.predict_proba(X_train_tfidfw2v)
[:,1])
test_fpr, test_tpr, thresholds = roc_curve(y_test, cal_model.predict_proba(X_test_tfidfw2v)[:,1])

model_optimal_sgd_tfidfw2v_train = auc(train_fpr, train_tpr)
model_optimal_sgd_tfidfw2v_test = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_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.title("ROC Curve")
plt.show()
```



In [0]:

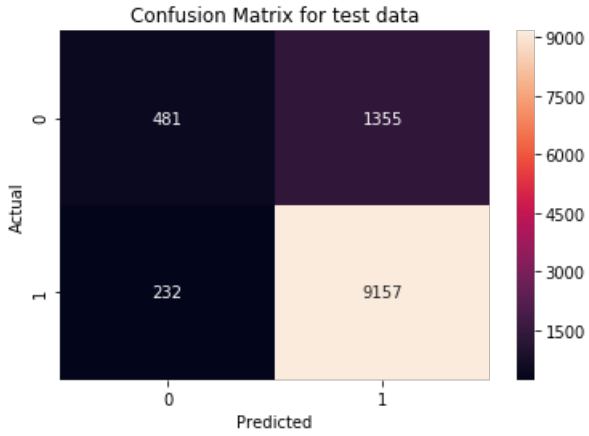
```
conf_matrix = confusion_matrix(y_train_cv,cal_model.predict(X_train_tfidfw2v))
class_label = [0,1]
df_conf_matrix = pd.DataFrame(conf_matrix, index = class_label , columns = class_label)
sns.heatmap(df_conf_matrix,annot=True , fmt = 'd')
plt.title("Confusion Matrix for train data")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```





In [0]:

```
conf_matrix = confusion_matrix(y_test,cal_model.predict(X_test_tfidfw2v))
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("Predicted")
plt.ylabel("Actual")
plt.show()
```



[5.2] RBF SVM

[5.2.1] Applying RBF SVM on BOW, SET 1

In [0]:

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

In [0]:

```
#taking 20k points
rbf_final = final.take(np.random.permutation(len(final))[:20000])
```

In [58]:

```
rbf_final.head()
```

Out[58]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
4982	5406	B00622CYVS	A26UBFCRT3FOAA	Kiddogmom	0	0	1	201 02-
36979	40175	B0012YEKCM	ACBL0UE0EIM00	Kgo	1	1	1	201 05-

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
37023	40221	B0030GOO5S	A3DZCNE8GA9H2Q	Nathan@MS	1	1	0	201 12-
6883	7527	B000OIWY8Y	A2T7VBHCN8I17	sara valenti	0	0	1	201 11-
18037	19652	B000084ETV	AK3Q0YL8EX7MD	Teach5233	0	0	1	201 10-

In [59]:

```
#splitting data into train , test
X_train, X_test, y_train, y_test = train_test_split(
    rbf_final['TotalText'], rbf_final['Score'], test_size=0.30, random_state=0)

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(14000,) (14000,)
(6000,) (6000,)
```

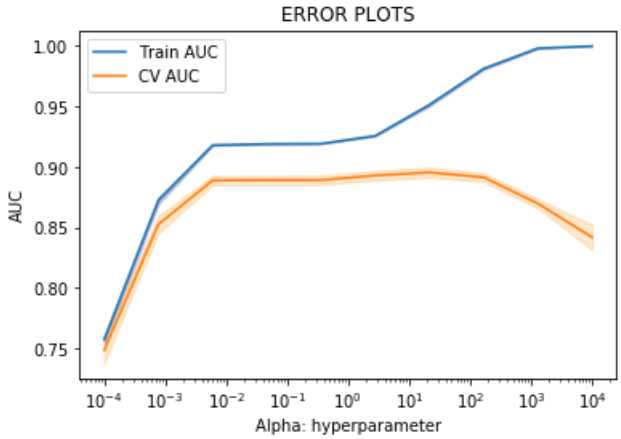
In [0]:

```
count_vect = CountVectorizer(min_df = 10,max_features=500)
X_train_bow = count_vect.fit_transform(X_train)
X_test_bow = count_vect.transform(X_test)
```

In [61]:

```
from sklearn.model_selection import  GridSearchCV
from sklearn.svm import  SVC
svmrb = SVC()
cval = np.logspace(-4,4,10)
parameters = {'C':cval}

clf = GridSearchCV(svmrb, parameters, cv=3, scoring='roc_auc',return_train_score = True)
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(cval, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(cval,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='da
rkblue')

plt.plot(cval, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(cval,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
```

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

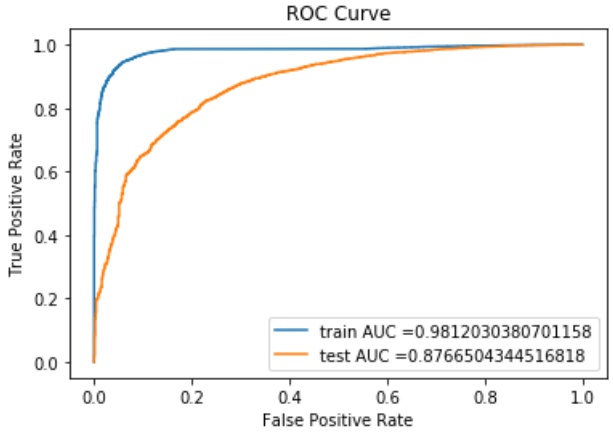


In [0]:

```
optimal_rbf_bow = 100
```

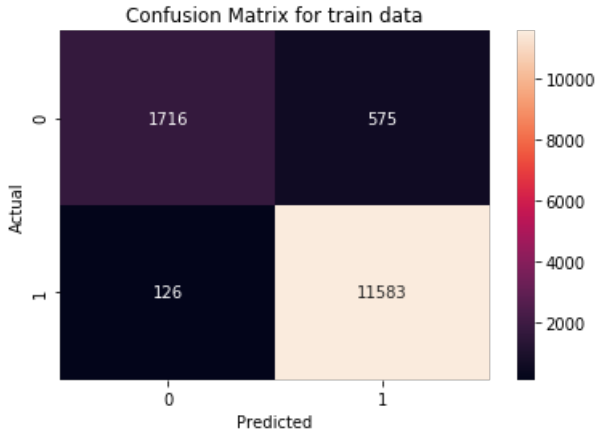
In [82]:

```
from sklearn.svm import SVC

model_rbf_bow = SVC(kernel='rbf' , C = optimal_rbf_bow , class_weight = 'balanced')
model_rbf_bow.fit(X_train_bow,y_train)

roc_train_score = model_rbf_bow.decision_function(X_train_bow)
roc_test_score = model_rbf_bow.decision_function(X_test_bow)

train_fpr, train_tpr, thresholds = roc_curve(y_train, roc_train_score)
test_fpr, test_tpr, thresholds = roc_curve(y_test, roc_test_score)

auc_bow = auc(test_fpr, test_tpr)


model_optimal_svm_bow_train = auc(train_fpr, train_tpr)
model_optimal_svm_bow_test = auc(test_fpr, test_tpr)


plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_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.title("ROC Curve")
plt.show()
```



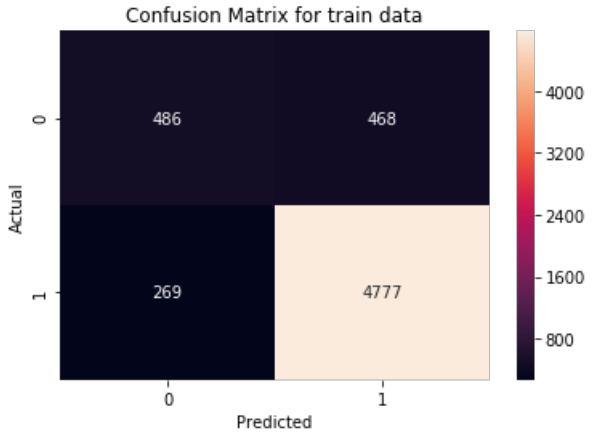
In [64]:

```
conf_matrix = confusion_matrix(y_train,model_rbf_bow.predict(X_train_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 train data")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [65]:

```
conf_matrix = confusion_matrix(y_test,model_rbf_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 train data")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



[5.2.2] Applying RBF SVM on TFIDF, SET 2

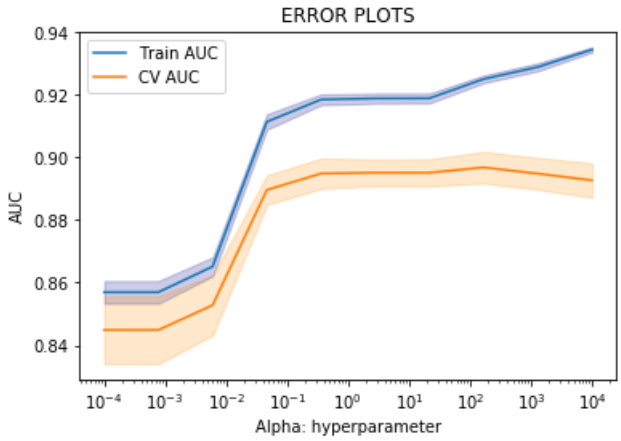
In [0]:

```
tfidf = TfidfVectorizer(min_df = 10, max_features = 500)

X_train_tfidf = tfidf.fit_transform(X_train)
X_test_tfidf = tfidf.transform(X_test)
```

In [67]:

```
from sklearn.model_selection import  GridSearchCV
from sklearn.svm import  SVC
svmrb = SVC()
cval = np.logspace(-4,4,10)
```

```
parameters = {'C':cval}

clf = GridSearchCV(svmrb, parameters, cv=3, scoring='roc_auc',return_train_score= True)
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.plot(cval, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(cval,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='da
rkblue')

plt.plot(cval, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(cval,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.xscale('log')
plt.title("ERROR PLOTS")
plt.show()
```



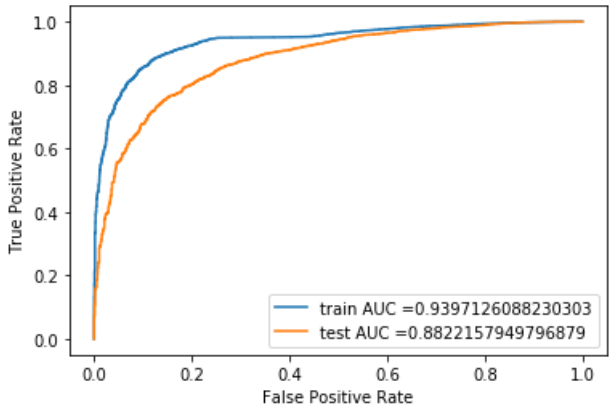
In [0]:

```
optimal_rbf_tfidf = 10000
```

In [83]:

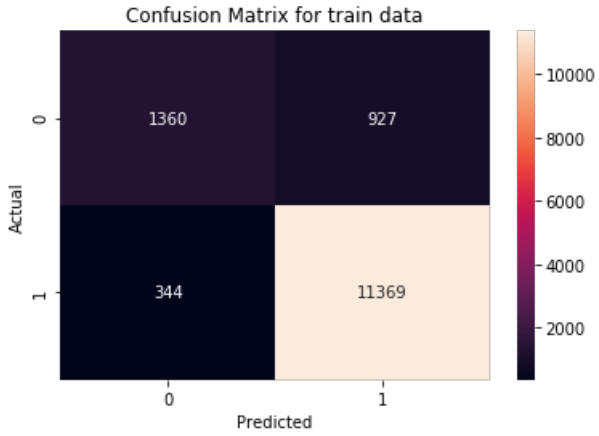
```
from sklearn.svm import SVC

model_rbf_tfidf = SVC(kernel='rbf' , C = optimal_rbf_tfidf , class_weight = 'balanced')
model_rbf_tfidf.fit(X_train_tfidf,y_train)

roc_train_score = model_rbf_tfidf.decision_function(X_train_tfidf)
roc_test_score = model_rbf_tfidf.decision_function(X_test_tfidf)

train_fpr, train_tpr, thresholds = roc_curve(y_train, roc_train_score)
test_fpr, test_tpr, thresholds = roc_curve(y_test, roc_test_score)

#auc_bow = auc(test_fpr, test_tpr)

model_optimal_svm_tfidf_train = auc(train_fpr, train_tpr)
model_optimal_svm_tfidf_test = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_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.title("ROC Curve")
plt.show()
```

ROC Curve



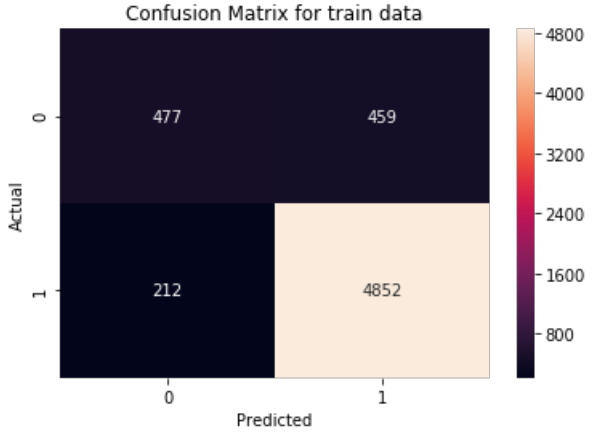
In [0]:

```
conf_matrix = confusion_matrix(y_train,model_rbf_tfidf.predict(X_train_tfidf))
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("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [0]:

```
conf_matrix = confusion_matrix(y_test,model_rbf_tfidf.predict(X_test_tfidf))
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("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [0]:

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

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

In [0]:

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

In [70]:

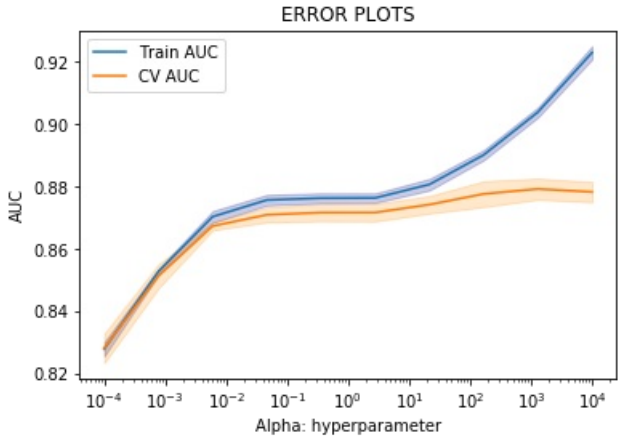
```
# Train your own Word2Vec model using your own text corpus

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)
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  7409
```

In [71]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

svmrb = SVC()
cval = np.logspace(-4,4,10)
parameters = {'C':cval}

clf = GridSearchCV(svmrb , parameters, cv=3 , scoring = 'roc_auc' ,return_train_score= True)
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.plot(cval, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(cval,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='da
rkblue')

plt.plot(cval, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(cval,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.xscale('log')
plt.title("ERROR PLOTS")
plt.show()
```

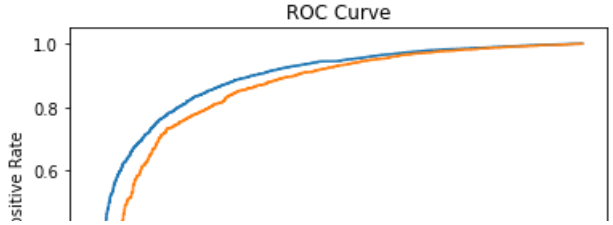


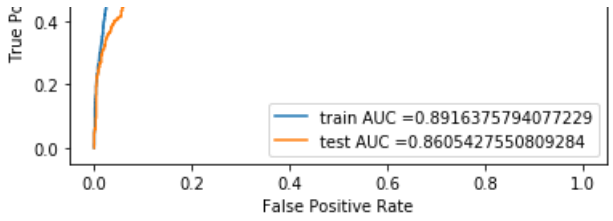
In [0]:

```
optimal_rbf_AvgW2V = 100
```

In [84]:

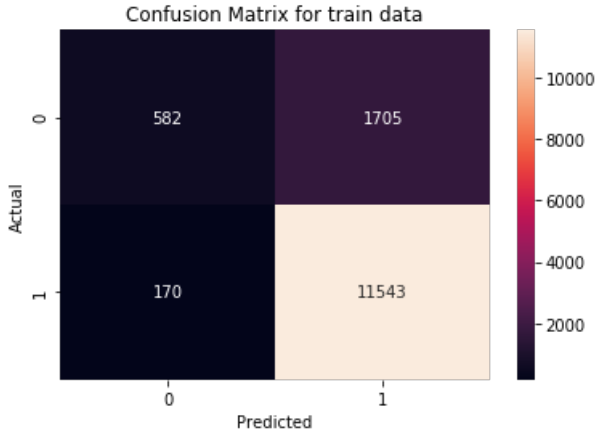
```
from sklearn.svm import SVC

model_rbf_AvgW2V = SVC(kernel='rbf' , C = optimal_rbf_AvgW2V , class_weight = 'balanced')
model_rbf_AvgW2V.fit(X_train_vectors,y_train)

roc_train_score = model_rbf_AvgW2V.decision_function(X_train_vectors)
roc_test_score = model_rbf_AvgW2V.decision_function(X_test_vectors)

train_fpr, train_tpr, thresholds = roc_curve(y_train, roc_train_score)
test_fpr, test_tpr, thresholds = roc_curve(y_test, roc_test_score)

#auc_bow = auc(test_fpr, test_tpr)

model_optimal_svm_AvgW2V_train = auc(train_fpr, train_tpr)
model_optimal_svm_AvgW2V_test = auc(test_fpr, test_tpr)


plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_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.title("ROC Curve")
plt.show()
```





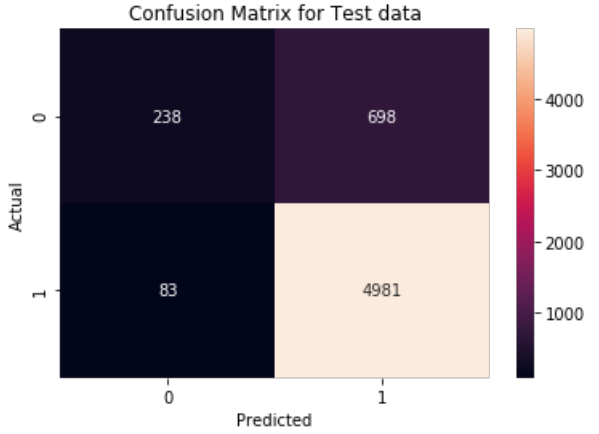
In [0]:

```
conf_matrix = confusion_matrix(y_train,model_rbf_AvgW2V.predict(X_train_vectors))
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("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [0]:

```
conf_matrix = confusion_matrix(y_test,model_rbf_AvgW2V.predict(X_test_vectors))
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("Predicted")
plt.ylabel("Actual")
plt.show()
```



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

In [0]:

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

In [74]:

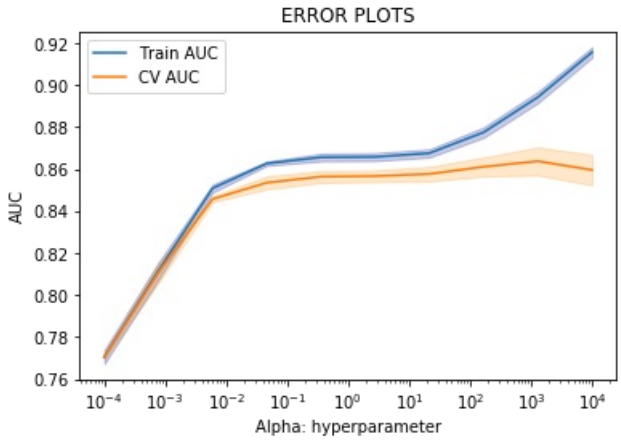
```
# Please write all the code with proper documentation

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 value
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 cell_val = tfidf

X_train_tfidfw2v = []; # the tfidf-w2v for each sentence/review is stored 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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]

            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_train_tfidfw2v.append(sent_vec)
    row += 1


X_test_tfidfw2v = []; # the tfidf-w2v for each sentence/review is stored 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/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]

            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    X_test_tfidfw2v.append(sent_vec)
    row += 1
```

```
100%|██████████| 14000/14000 [04:48<00:00, 48.51it/s]
100%|██████████| 6000/6000 [02:00<00:00, 49.74it/s]
```

In [75]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

svmrb = SVC()
cval = np.logspace(-4,4,10)
parameters = {'C':cval}

clf = GridSearchCV(svmrb , parameters, cv=3 , scoring = 'roc_auc' ,return_train_score= True)
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.plot(cval, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(cval,train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='da
rkblue')
```

```
plt.plot(cval, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(cval,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')
plt.legend()
plt.xlabel("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.xscale('log')
plt.title("ERROR PLOTS")
plt.show()
```

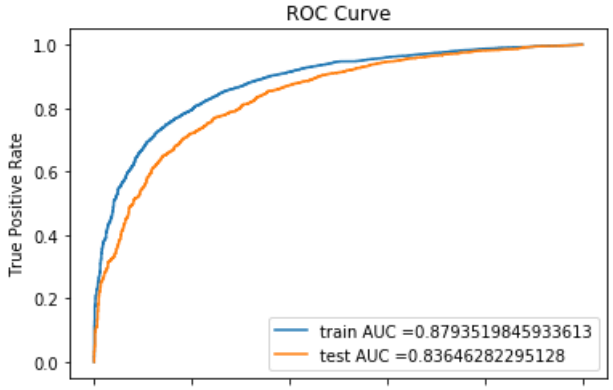


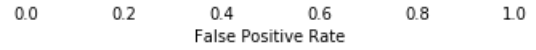
In [0]:

```
optimal_rbf_tfidfW2V = 100
```

In [85]:

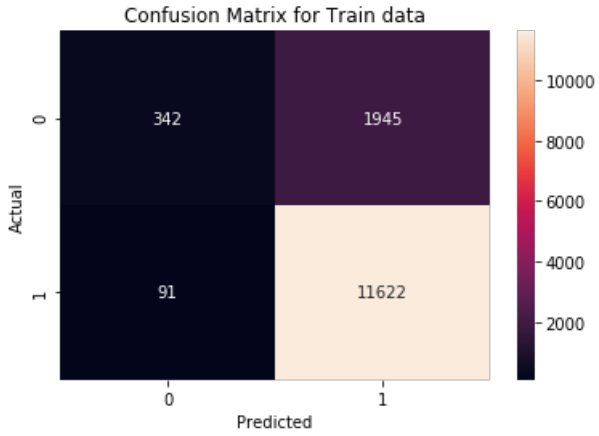
```
from sklearn.svm import SVC

model_rbf_tfidfw2v = SVC(kernel='rbf' , C = optimal_rbf_tfidfW2V , class_weight = 'balanced')
model_rbf_tfidfw2v.fit(X_train_tfidfw2v,y_train)

roc_train_score = model_rbf_tfidfw2v.decision_function(X_train_tfidfw2v)
roc_test_score = model_rbf_tfidfw2v.decision_function(X_test_tfidfw2v)

train_fpr, train_tpr, thresholds = roc_curve(y_train, roc_train_score)
test_fpr, test_tpr, thresholds = roc_curve(y_test, roc_test_score)

#auc_bow = auc(test_fpr, test_tpr)


model_optimal_svm_tfidfW2V_train = auc(train_fpr, train_tpr)
model_optimal_svm_tfidfW2V_test = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_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.title("ROC Curve")
plt.show()
```





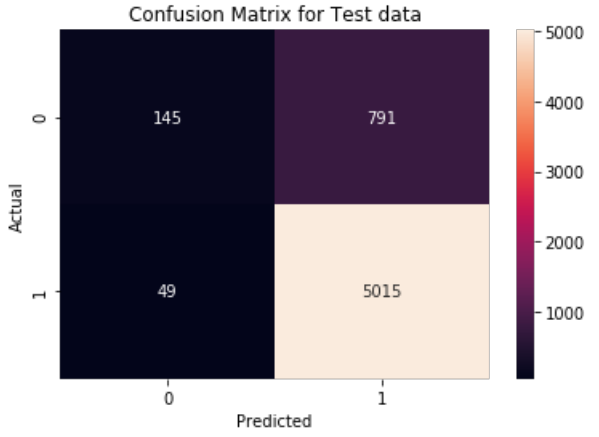
In [0]:

```
conf_matrix = confusion_matrix(y_train,model_rbf_tfidfw2v.predict(X_train_tfidfw2v))
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("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [0]:

```
conf_matrix = confusion_matrix(y_test,model_rbf_tfidfw2v.predict(X_test_tfidfw2v))
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("Predicted")
plt.ylabel("Actual")
plt.show()
```



[6] Conclusions

In [0]:

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

In [97]:

```
from prettytable import  PrettyTable

#Models
```

```
#Models

Model_Names = ['SGD(Linear SVM) for BoW','SGD(Linear SVM) for TFIDF',\
         'SGD(Linear SVM) for Avg_Word2Vec',\
         'SGD(Linear SVM) for tfidf_Word2Vec',\
         "SVC(RBF Kernel) for BOW","SVC(RBF Kernel) for TFIDF","SVC(RBF Kernel) for Avg_Word2Vec",'
SVC(RBF Kernel) for tfidf_Word2Vec"]

Optimal_Alphas =
[optimal_alpha_bow,optimal_alpha_tfidf,optimal_alpha_AvgW2V,optimal_alpha_tfidfW2V,
                 "NA","NA","NA","NA"]

Optimal_C = ["NA", "NA" ,"NA" , "NA"
,optimal_rbf_bow,optimal_rbf_tfidf,optimal_rbf_AvgW2V,optimal_rbf_tfidfW2V]

Optimal_AUCs_train =
[model_optimal_sgd_bow_train,model_optimal_sgd_tfidf_train,model_optimal_sgd_avgw2v_train ,
model_optimal_sgd_tfidfw2v_train ,

model_optimal_svm_bow_train,model_optimal_svm_tfidf_train,model_optimal_svm_AvgW2V_train,model_opti
mal_svm_tfidfW2V_train]

Optimal_AUCs_test =
[model_optimal_sgd_bow_test,model_optimal_sgd_tfidf_test,model_optimal_sgd_avgw2v_test,model_optima
l_sgd_tfidfw2v_test,

model_optimal_svm_bow_test,model_optimal_svm_tfidf_test,model_optimal_svm_AvgW2V_test,model_optimal
_svm_tfidfW2V_test]

Numbers = [1,2,3,4,5,6,7,8]

PreTable = PrettyTable()

PreTable.add_column("S.No" , Numbers)
PreTable.add_column("Models" , Model_Names)
PreTable.add_column("Best Alphas" , Optimal_Alphas)
PreTable.add_column("Best C values",Optimal_C)
PreTable.add_column("Optimal_Train_AUCs" , Optimal_AUCs_train )
PreTable.add_column("optimal_test_AUCs",Optimal_AUCs_test)

print(PreTable)
```

```
+------+-------------------------------------+-------------+---------------+--------------------+---
--------------+
| S.No |               Models                | Best Alphas | Best C values | Optimal_Train_AUCs |
optimal_test_AUCs  |
+------+-------------------------------------+-------------+---------------+--------------------+---
--------------+
|  1   |      SGD(Linear SVM) for BoW        |    0.0001   |       NA      | 0.9990674439424991 |
0.9637334426837041 |
|  2   |     SGD(Linear SVM) for TFIDF       |    0.001    |       NA      | 0.9990674439424991 | 0.
637334426837041 |
|  3   |  SGD(Linear SVM) for Avg_Word2Vec   |     0.1     |       NA      | 0.8760893827806091 | 0.
8623022490778756 |
|  4   | SGD(Linear SVM) for tfidf_Word2Vec  |     0.1     |       NA      | 0.8192546454634494 | 0.
8014665496717411 |
|  5   |      SVC(RBF Kernel) for BOW        |      NA     |      100      | 0.9812030380701158 | 0.
766504344516818 |
|  6   |     SVC(RBF Kernel) for TFIDF       |      NA     |     10000     | 0.9397126088230303 | 0.
822157949796879 |
|  7   |  SVC(RBF Kernel) for Avg_Word2Vec   |      NA     |      100      | 0.8916375794077229 | 0.
8605427550809284 |
|  8   | SVC(RBF Kernel) for tfidf_Word2Vec  |      NA     |      100      | 0.8793519845933613 |  0
.83646282295128  |
+------+-------------------------------------+-------------+---------------+--------------------+---
--------------+
```