Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [1]:

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


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os
```

In [2]:

```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000""", co
n)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 ORDER BY Time ASC""",
con)

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

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

```
Number of data points in our data (525814, 10)
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
0	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	1	93934080
1	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	1	94080960

2	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Tim
	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	1	94409280

In [3]:

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

In [4]:

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

```
(80668, 7)
```

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	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 [5]:

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

Out[5]:

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

In [6]:

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

Out[6]:

```
393063
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove

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

In [7]:

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

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	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

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

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

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

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [8]:

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='qui
cksort', na_position='last')
```

In [9]:

In [9]:

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

Out[9]:

```
(364173, 10)
```

In [10]:

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

Out[10]:

```
69.25890143662969
```

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

In [11]:

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

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	12248928
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	12128832

In [12]:

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

In [13]:

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

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

```
(364171, 10)
```

Out[13]:

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

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [14]:

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

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

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

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

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer
bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back
in 2005 for gosh sakes.  I admit that Amazon agreed to credit me for cost plus part of shipping, b
ut geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe that carries pods so t
hat I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured
out a way to fix that. I still like it but it could be better.
==================================================
Originally found the syrup for sale in a tourist shop in North Georgia mountains. Could not find i
t locally and have been ordering from manufacturer ever since. I am diabetic and really think this
syrup is better than the 'full blown' stuff!
==================================================
```

In [15]:

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

```
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
```

In [16]:

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an
-element
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
==================================================
I was really looking forward to these pods based on the reviews.  Starbucks is good, but I prefer
bolder taste.... imagine my surprise when I ordered 2 boxes - both were expired! One expired back
in 2005 for gosh sakes.  I admit that Amazon agreed to credit me for cost plus part of shipping, b
ut geez, 2 years expired!!!  I'm hoping to find local San Diego area shoppe that carries pods so t
hat I can try something different than starbucks.
==================================================
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today's Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70's it was poisonous until they figured
out a way to fix that. I still like it but it could be better.
==================================================
Originally found the syrup for sale in a tourist shop in North Georgia mountains. Could not find i
t locally and have been ordering from manufacturer ever since. I am diabetic and really think this
syrup is better than the 'full blown' stuff!
```

In [17]:

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

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

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

In [18]:

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

```
Great ingredients although, chicken should have been 1st rather than chicken broth, the only thing
I do not think belongs in it is Canola oil. Canola or rapeseed is not someting a dog would ever fi
nd in nature and if it did find rapeseed in nature and eat it, it would poison them. Today is Food
industries have convinced the masses that Canola oil is a safe and even better oil than olive or v
irgin coconut, facts though say otherwise. Until the late 70 is it was poisonous until they
figured out a way to fix that. I still like it but it could be better.
==================================================
```

In [19]:

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

```
this witty little book makes my son laugh at loud. i recite it in the car as we're driving along a
nd he always can sing the refrain. he's learned about whales, India, drooping roses:  i love all t
he new words this book  introduces and the silliness of it all.  this is a classic book i am
willing to bet my son will STILL be able to recite from memory when he is  in college
```

In [20]:

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

```
Great ingredients although chicken should have been 1st rather than chicken broth the only thing I
do not think belongs in it is Canola oil Canola or rapeseed is not someting a dog would ever find
in nature and if it did find rapeseed in nature and eat it it would poison them Today is Food indu
stries have convinced the masses that Canola oil is a safe and even better oil than olive or virgi
n coconut facts though say otherwise Until the late 70 is it was poisonous until they figured out
a way to fix that I still like it but it could be better
```

In [21]:

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

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

In [22]:

```
# 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%|████████████████████████████████████████████████████████████████████████| 364171/364171
[03:43<00:00, 1627.37it/s]
```

In [23]:

```
def words_count(x):
    return len(x.split())

preprocessed_reviews_word_count = list(map(words_count,preprocessed_reviews))
```

In [24]:

```
data = preprocessed_reviews[:100000]
scores = final['Score'][:100000]
data_word_count = preprocessed_reviews_word_count[:100000]

print(final['Score'][:100000].value_counts())
```

```
1    85198
0    14802
Name: Score, dtype: int64
```

In [26]:

```
from sklearn.model_selection import train_test_split

data_train,data_test,scores_train,scores_test = train_test_split(data,scores,test_size = 0.2,shuffl
e = False)
data_train,data_cv,scores_train,scores_cv = train_test_split(data_train,scores_train,test_size = 0
.25,shuffle = False)
data_train_word_count,data_test_word_count,scores_train,scores_test =
train_test_split(data_word_count,scores,test_size = 0.2,shuffle = False)
data_train_word_count,data_cv_word_count,scores_train,scores_cv =
train_test_split(data_train_word_count,scores_train,test_size = 0.25,shuffle = False)

print(len(data_train))
print(len(data_cv))
print(len(data_test))
print(len(data_train_word_count))
print(len(data_cv_word_count))
print(len(data_test_word_count))
print(len(scores_train))
print(len(scores_cv))
print(len(scores_test))
```

```
60000
20000
```

```
20000
60000
20000
20000
60000
20000
20000
```

In [ ]:

```
print(scores_train.value_counts())
```

[3.2] Preprocessing Review Summary

In [ ]:

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

[4] Featurization

[4.1] BAG OF WORDS

In [29]:

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(data_train)
data_train_bow = count_vect.fit_transform(data_train)
#data_train_bow_backup = data_train_bow
data_cv_bow = count_vect.transform(data_cv)
data_test_bow = count_vect.transform(data_test)
print(data_train_bow.shape)
print(data_cv_bow.shape)
print(data_test_bow.shape)
print(type(data_train_bow))
```

```
(60000, 47800)
(20000, 47800)
(20000, 47800)
<class 'scipy.sparse.csr.csr_matrix'>
```

[4.2] TF-IDF

In [30]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(data_train)
data_train_tf_idf = tf_idf_vect.fit_transform(data_train)
#data_train_tf_idf_backup = data_train_tf_idf
data_cv_tf_idf = tf_idf_vect.transform(data_cv)
data_test_tf_idf = tf_idf_vect.transform(data_test)
print(data_train_tf_idf.shape)
print(data_cv_tf_idf.shape)
print(data_test_tf_idf.shape)
```

```
(60000, 34374)
(20000, 34374)
(20000, 34374)
```

[4.3] Word2Vec

In [31]:

```
# Train your own Word2Vec model using your own text corpus
x_train = []
for i in data_train:
    x_train.append(i.split())
w2v_model=Word2Vec(x_train,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

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

[4.3.1.1] Avg W2v

In [32]:

```
def avg_W2V(list_of_sentance,w2v_model,w2v_words):
    # 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 t
his to 300 if you use google's w2v
        cnt_words =0; # num of words with a valid vector in the sentence/review
        for word in sent.split(): # 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)
    return sent_vectors

#Brute
data_train_avg_w2v = avg_W2V(data_train,w2v_model,w2v_words)
data_cv_avg_w2v = avg_W2V(data_cv,w2v_model,w2v_words)
data_test_avg_w2v = avg_W2V(data_test,w2v_model,w2v_words)
#data_train_avg_w2v_backup = data_train_avg_w2v
```

```
100%|██████████████████████████████████████████████████| 60000/60000 [04:
19<00:00, 231.05it/s]
100%|██████████████████████████████████████████████████| 20000/20000 [01:
10<00:00, 282.04it/s]
100%|██████████████████████████████████████████████████| 20000/20000 [01:
08<00:00, 290.94it/s]
```

[4.3.1.2] TFIDF weighted W2v

In [33]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(data_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()
```

In [34]:

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

def tf_idf_w2v(list_of_sentance,w2v_model,w2v_words,tfidf_feat,dictionary):
    tfidf_sent_vectors = []; # the tfidf-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
        weight_sum =0; # num of words with a valid vector in the sentence/review
        for word in sent.split(): # 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
                # to reduce the computation we are
                # dictionary[word] = idf value of word in whole corpus
                # 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)
    return tfidf_sent_vectors

data_train_tf_idf_w2v = tf_idf_w2v(data_train,w2v_model,w2v_words,tfidf_feat,dictionary)
data_cv_tf_idf_w2v = tf_idf_w2v(data_cv,w2v_model,w2v_words,tfidf_feat,dictionary)
data_test_tf_idf_w2v = tf_idf_w2v(data_test,w2v_model,w2v_words,tfidf_feat,dictionary)
#data_train_tfidf_w2v_backup = data_train_tfidf_w2v
```

```
100%|████████████████████████████████████████████████████████████████████████████| 60000/60000 [57
:17<00:00, 17.46it/s]
100%|████████████████████████████████████████████████████████████████████████████| 20000/20000 [19
:43<00:00, 16.90it/s]
100%|████████████████████████████████████████████████████████████████████████████| 20000/20000 [18
:16<00:00, 20.23it/s]
```

In [ ]:

[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

In [39]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV

def get_AUC(X_train,y_train,X_cv,y_cv,c_range,reg):
    """This function apply the knn model with given algorithm
       on train and cv data and return AUC values for train and cross validation"""
    auc_train = []
    auc_cv = []
    for c in c_range:
        LR = LogisticRegression(penalty = reg, C = c, class_weight = 'balanced')
        LR.fit(X_train,y_train)

        prob_train = LR.predict_proba(X_train)
        fpr, tpr, threshold = roc_curve(y_train, prob_train[:, 1])
        auc_train.append(auc(fpr,tpr))

        prob_cv = LR.predict_proba(X_cv)
        fpr, tpr, threshold = roc_curve(y_cv, prob_cv[:, 1])
        auc_cv.append(auc(fpr,tpr))

    return auc_train,auc_cv

def plot_AUC_Curves(auc_train,auc_cv,c_range):
    """This function plots the auc curves for the given auc values and c_range"""
    sns.set_style("whitegrid",{'axes.grid' : False})
    plt.plot(np.log(c_range),auc_train,"b-",label = "AUC_Train")
    plt.plot(np.log(c_range),auc_cv,"r-",label = "AUC_Validation")
    plt.scatter(np.log(c_range),auc_train)
    plt.scatter(np.log(c_range),auc_cv)
    plt.legend()
    plt.xlabel("Hyper Parameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()

def apply_roc_curve(X_train,y_train,X_test,y_test,optimal_c,reg):
    """This function apply Logistic Regression model on train and predict labels for test data
      and also find FPR and TPR for train and test data.
```

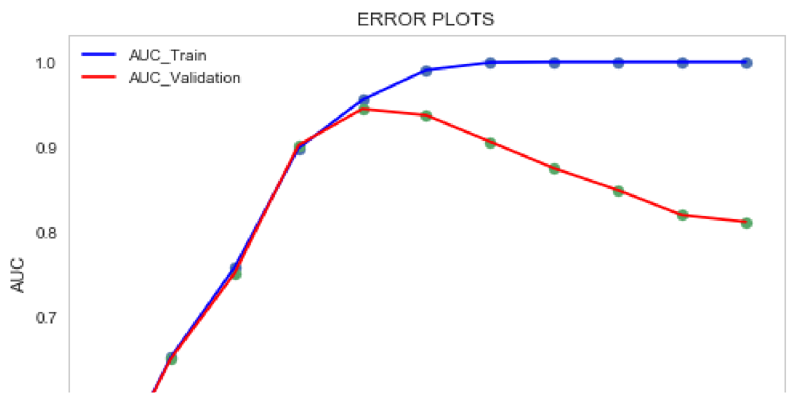
```
       Returns the predicted labels,FPR and TPR values"""
    LR = LogisticRegression(C = optimal_c, penalty = reg,class_weight = 'balanced')
    LR.fit(X_train,y_train)
    prob_train = LR.predict_proba(X_train)
    fpr_train, tpr_train, threshold = roc_curve(y_train, prob_train[:, 1])
    prob_test = LR.predict_proba(X_test)
    fpr_test, tpr_test, threshold = roc_curve(y_test, prob_test[:, 1])

    # predict the class labels
    pred_train = LR.predict(X_train)
    pred_test = LR.predict(X_test)

    w = LR.coef_
    return fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test,w

def plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test):
    """This function plot the roc curves for train and test data"""
    # plot ROC curves for train and test data
    plt.plot(fpr_train,tpr_train,"g-",label = "AUC_Train : "+str(auc(fpr_train, tpr_train)))
    plt.plot(fpr_test,tpr_test,"r-",label = "AUC_Test : "+str(auc(fpr_test, tpr_test)))
    plt.plot([0,1],[0,1],"b-")
    plt.legend(loc="lower right")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.show()

def plot_Confusion_Matrix(actual_labels,predict_labels,title):
    """This function plot the confusion matrix"""
    # Reference : https://seaborn.pydata.org/generated/seaborn.heatmap.html
    cm = confusion_matrix(actual_labels, predict_labels)
    classNames = ['NO','YES']
    cm_data = pd.DataFrame(cm,index = classNames,
                  columns = classNames)
    plt.figure(figsize = (5,4))
    sns.heatmap(cm_data, annot=True,fmt="d")
    plt.title(title)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()
```

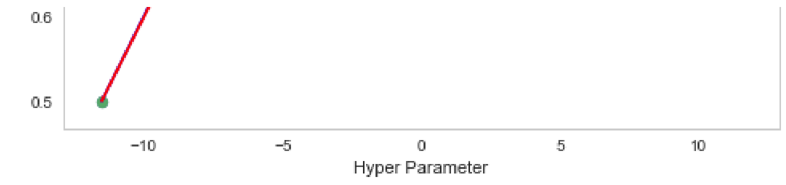
[5.1] Logistic Regression on BOW, SET 1

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

In [40]:

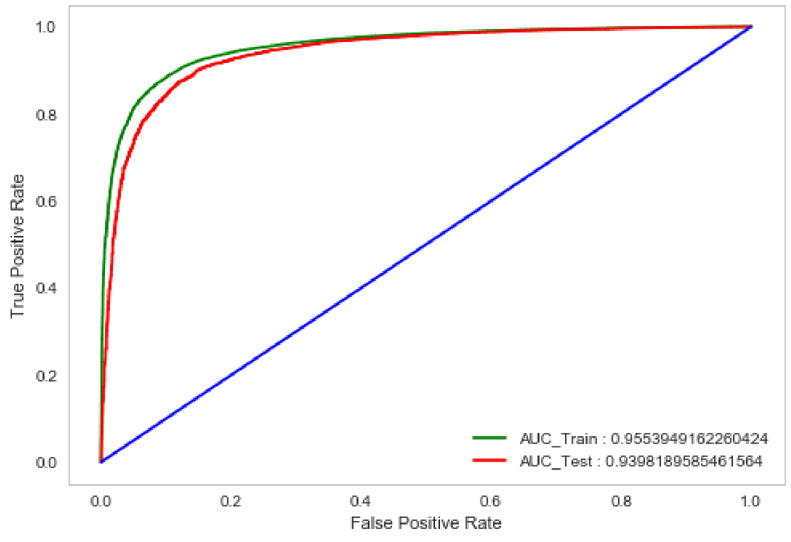
```
# Please write all the code with proper documentation
# dictionary C
hyper_parameter = {"C" : [10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4,10**5]}
# C range
C = hyper_parameter.get("C")

# AUC
train_auc,cv_auc = get_AUC(data_train_bow,scores_train,data_cv_bow,scores_cv,C,'l1')

# plot auc
plot_AUC_Curves(train_auc,cv_auc,C)
```





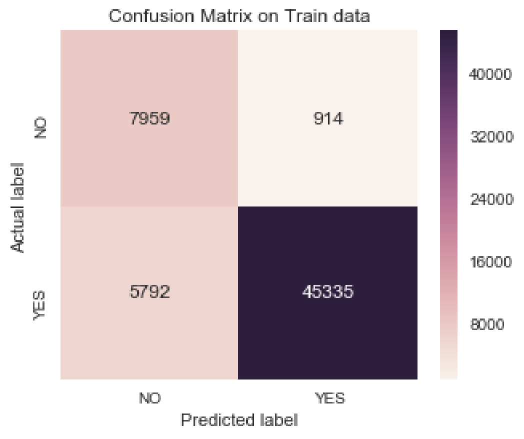
In [43]:

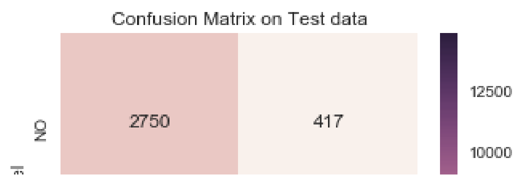
```
# optimal alpha
optimal_C_bow_L1 = C[cv_auc.index(max(cv_auc))]
#optimal_C_bow_L1 = 1
# roc
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test,w =
apply_roc_curve(data_train_bow,scores_train,data_test_bow,scores_test,optimal_C_bow_L1,'l1')
# auc
auc_bow_L1 = auc(fpr_test,tpr_test)
# plot roc
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)
```

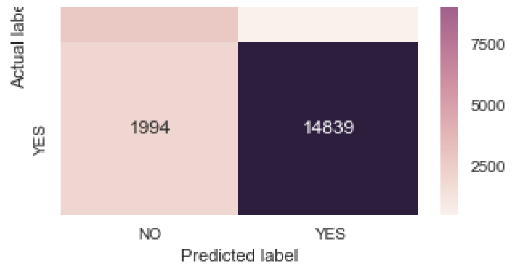


In [44]:

```
# plot confusion matrix
plot_Confusion_Matrix(scores_train,pred_train,"Confusion Matrix on Train data")
plot_Confusion_Matrix(scores_test,pred_test,"Confusion Matrix on Test data")
print("C =",optimal_C_bow_L1)
print("AUC =",auc_bow_L1)
```







```
C = 0.1
AUC = 0.9398189585461564
```

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

In [45]:

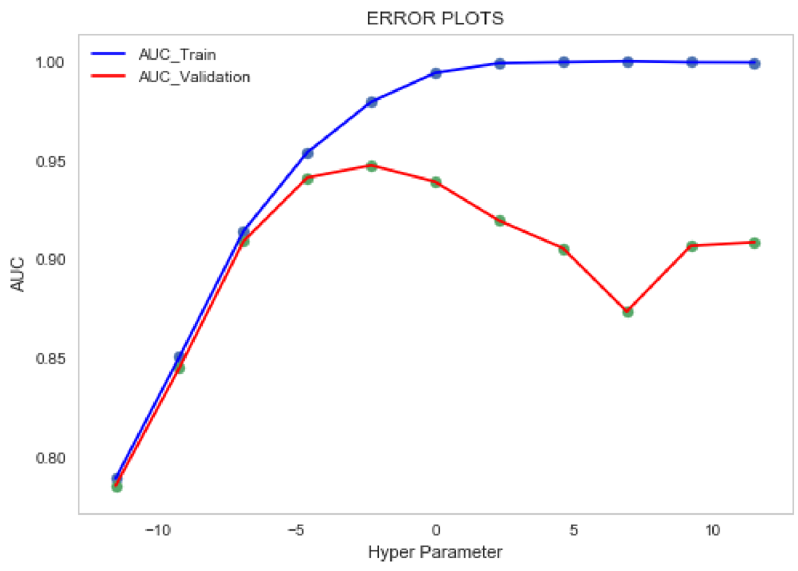
```
# Please write all the code with proper documentation
non_zero = np.count_nonzero(w)
total_val = np.product(w.shape)
sparsity = (total_val - non_zero) / total_val
print("Sparsity =",sparsity)
```

```
Sparsity = 0.9731380753138076
```

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

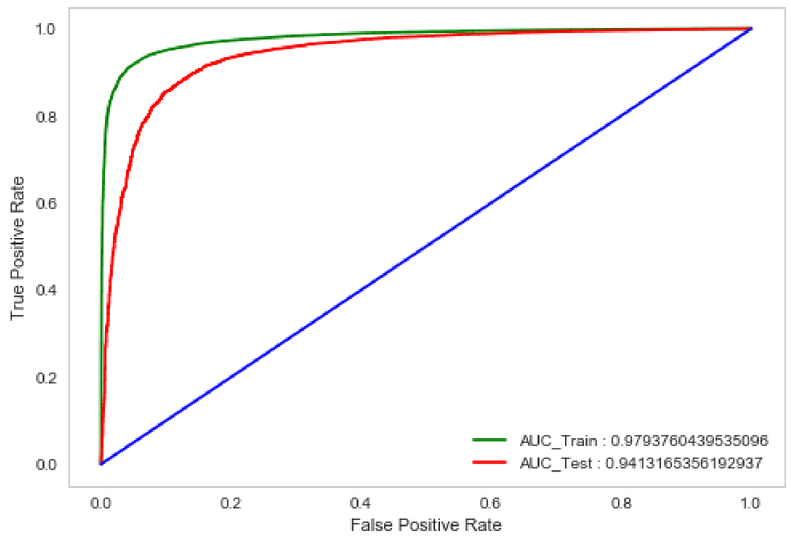
In [47]:

```
# Please write all the code with proper documentation
# dictionary C
hyper_parameter = {"C" : [10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4,10**5]}
# C range
C = hyper_parameter.get("C")
# AUC
train_auc,cv_auc = get_AUC(data_train_bow,scores_train,data_cv_bow,scores_cv,C,'l2')


# plot auc
plot_AUC_Curves(train_auc,cv_auc,C)
```



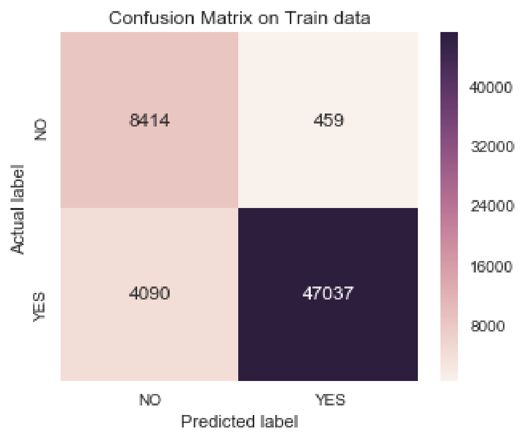
In [48]:

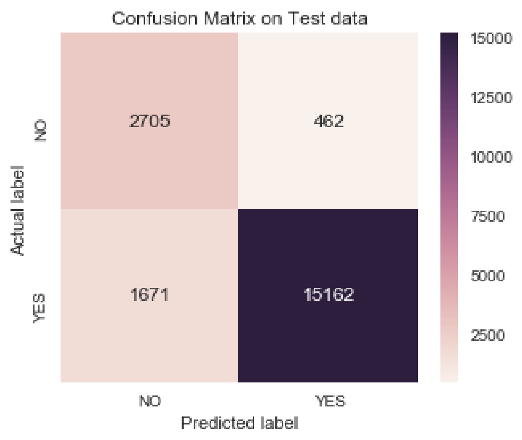
```
# optimal alpha
optimal_C_bow_L2 = C [cv_auc.index(max(cv_auc))]
# roc
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test,w =
apply_roc_curve(data_train_bow,scores_train,data_test_bow,scores_test,optimal_C_bow_L2,'l2')
# auc
auc_bow_L2 = auc(fpr_test,tpr_test)
# plot roc
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)
```



In [49]:

```
# plot confusion matrix
plot_Confusion_Matrix(scores_train,pred_train,"Confusion Matrix on Train data")
plot_Confusion_Matrix(scores_test,pred_test,"Confusion Matrix on Test data")
print("C =",optimal_C_bow_L2)
print("AUC =",auc_bow_L2)
```





```
C = 0.1
AUC = 0.9413165356192937
```

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

In [50]:

```
# Please write all the code with proper documentation
w_L2 = w

#noise
noise = abs(np.random.normal(0,0.1))
data_train_bow_noise = data_train_bow
data_train_bow_noise = data_train_bow_noise.astype(float)

# add noise to data
data_train_bow_noise.data += noise

# train LR model with noise data
LR = LogisticRegression(C = optimal_C_bow_L2, penalty = 'l2')
LR.fit(data_train_bow_noise,scores_train)

#weight vector
w_L2_noise = LR.coef_

# add Epslon
w_L2 = w_L2 + 10**-6
w_L2_noise = w_L2_noise + 10**-6

# percentage change
percentage_change = abs((w_L2-w_L2_noise)/(w_L2))*100
```

In [51]:

```
# percentiles from 0th to 100th
percentiles = np.percentile(percentage_change,np.arange(0,101,1))
def Round(x):
    return round(x,2)

percentiles = list(map(Round,percentiles))
print(percentiles)
```

```
[0.0, 0.88, 1.68, 2.58, 3.37, 4.14, 4.96, 5.7, 6.48, 7.35, 8.16, 8.98, 9.84, 10.51, 11.36, 12.06, 1
2.84, 13.58, 14.24, 14.97, 15.67, 16.32, 17.07, 17.74, 18.33, 18.96, 19.64, 20.28, 20.83, 21.4, 21
.97, 22.54, 23.17, 23.76, 24.37, 24.94, 25.45, 26.03, 26.66, 27.15, 27.65, 28.19, 28.74, 29.32, 29
.88, 30.41, 30.94, 31.51, 32.04, 32.54, 33.14, 33.72, 34.19, 34.7, 35.24, 35.81, 36.35, 36.88, 37.
44, 38.01, 38.54, 39.09, 39.72, 40.25, 40.82, 41.42, 42.04, 42.71, 43.37, 44.04, 44.67, 45.42, 46.
3, 47.06, 47.79, 48.64, 49.54, 50.56, 51.72, 52.85, 54.09, 55.32, 56.74, 58.43, 60.61, 62.82, 65.1
5, 68.55, 72.34, 77.07, 83.44, 92.23, 103.67, 119.56, 139.94, 165.61, 199.27, 260.02, 402.98, 823.
21, 232611.98]
```

In [52]:

```
# percentiles from 99.1th to 100th
percentiles = np.percentile(percentage_change,np.arange(99.1,100.1,0.1))
def Round(x):
    return round(x,2)

percentiles = list(map(Round,percentiles))
print(percentiles)
```

```
[890.86, 1008.85, 1008.85, 1168.73, 1387.06, 1671.22, 2163.42, 3342.92, 8464.88, 232611.98]
```

In [54]:

```
# since sudden range : 9057.08 to 306921.47
threshold = 8464.88
percentage_change_index = np.where(percentage_change>threshold)[1]
feature_names = np.take(count_vect.get_feature_names(),percentage_change_index)
print("Feature names whose % change is more than a threshold:\n")
print(feature_names)
```

```
Feature names whose % change is more than a threshold:

['aigr' 'amazonb' 'appetitoscientists' 'bulletwhile' 'captures' 'casings'
```

```
['alcl' 'amazonb' 'appetitescientists' 'bulletwhile' 'captures' 'casings'
 'catechin' 'cedar' 'charts' 'chillate' 'common' 'dappa' 'diane'
 'differentiating' 'ecopod' 'edamame' 'epigallocatechin' 'expenditure'
 'expendituregreen' 'fdabut' 'floppy' 'foie' 'frappucinno' 'handfull'
 'java' 'jiggled' 'kathryn' 'kelp' 'metabolisma' 'miscarriage'
 'naturallygreen' 'neillthere' 'nonetheless' 'othersin' 'oxidationburns'
 'phytochemical' 'problemsfor' 'regulationexperts' 'repulsed'
 'requirements' 'rinsing' 'satisifed' 'sp' 'temporary' 'thermogenesis'
 'trade' 'washer' 'wiggled']
```

[5.1.3] Feature Importance on BOW, SET 1

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

In [55]:

```
# Please write all the code with proper documentation
weights = w[0,:]
top10_positive_indices = weights.argsort()[-10:]
rank = np.array(range(1,11))
top10_positive_features = np.take(count_vect.get_feature_names(),top10_positive_indices)
prob = np.take(weights,top10_positive_indices)
top10_positive_features_details = pd.DataFrame(data = {'Rank' : rank,'Feature' : top10_positive_fea
tures,'Probability' : prob})
print(top10_positive_features_details)
```

```
     Feature  Probability  Rank
0      great     1.242577     1
1     hooked     1.251379     2
2       beat     1.287772     3
3       best     1.290808     4
4    awesome     1.366603     5
5    pleased     1.416479     6
6  excellent     1.417061     7
7    perfect     1.434910     8
8      loves     1.449791     9
9  delicious     1.511119    10
```

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

In [56]:

```
# Please write all the code with proper documentation
weights = w[0,:]
top10_neg_indices = weights.argsort()[:10]
rank = np.array(range(1,11))
top10_negative_features = np.take(count_vect.get_feature_names(),top10_neg_indices)
prob = np.take(weights,top10_neg_indices)
top10_negative_features_details = pd.DataFrame(data = {'Rank' : rank,'Feature' : top10_negative_fea
tures,'Probability' : prob})
print(top10_negative_features_details)
```

```
          Feature  Probability  Rank
0           worst    -2.206581     1
1    disappointing    -1.762562     2
2         terrible    -1.612921     3
3     disappointed    -1.461853     4
4            awful    -1.446757     5
5        tasteless    -1.319497     6
6            threw    -1.296867     7
7   disappointment    -1.295737     8
8              rip    -1.288476     9
9    unfortunately    -1.269733    10
```

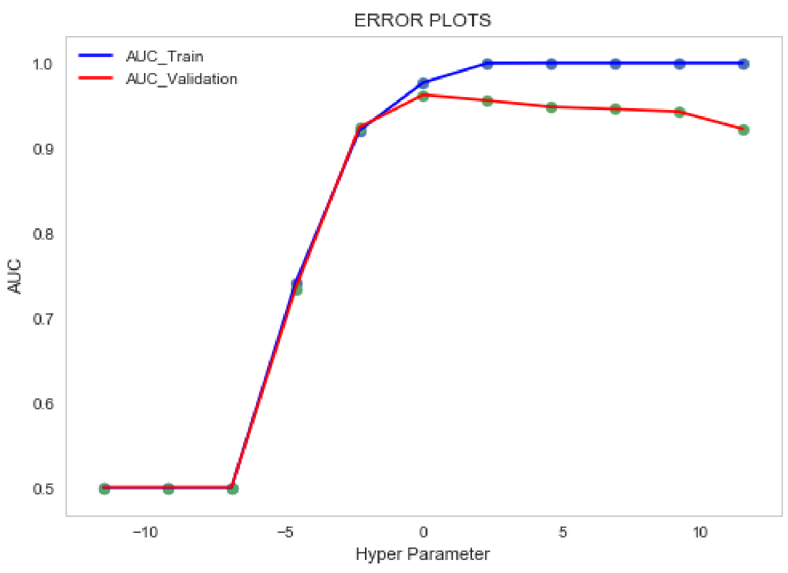
[5.2] Logistic Regression on TFIDF, SET 2

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

[illegible]

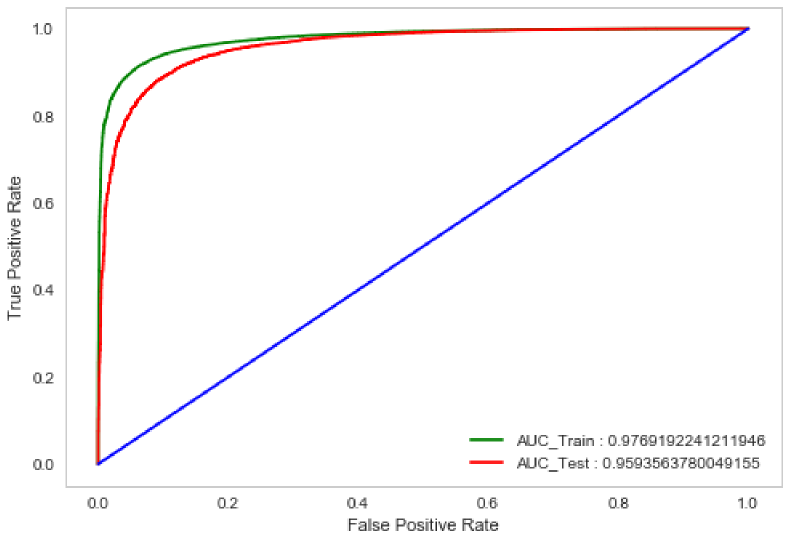
In [57]:

```
# Please write all the code with proper documentation
# dictionary C
hyper_parameter = {"C" : [10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4,10**5]}
# C range
C = hyper_parameter.get("C")
# AUC
train_auc,cv_auc = get_AUC(data_train_tf_idf,scores_train,data_cv_tf_idf,scores_cv,C,'l1')


# plot auc
plot_AUC_Curves(train_auc,cv_auc,C)
```



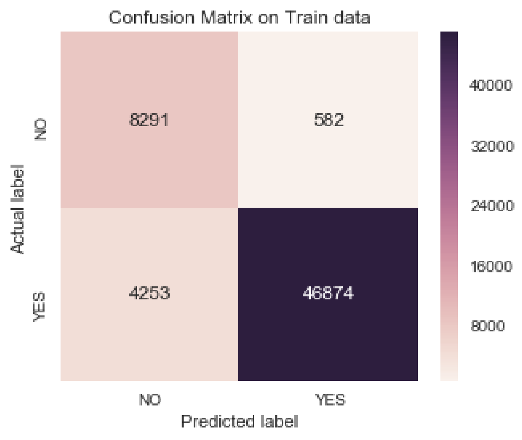
In [58]:

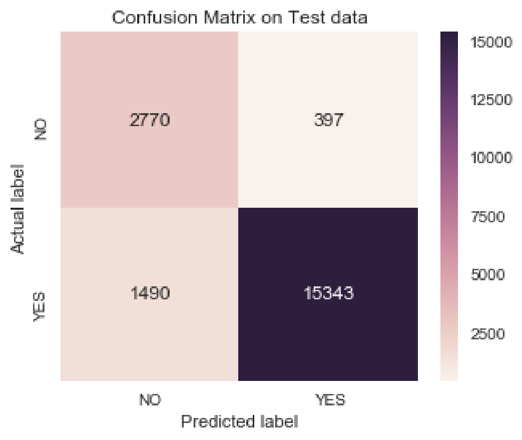
```
# optimal alpha
optimal_C_tfidf_L1 = C [cv_auc.index(max(cv_auc))]
# roc
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test,w = apply_roc_curve(data_train_tf_idf,s
cores_train,data_test_tf_idf,scores_test,optimal_C_tfidf_L1,'l1')
# auc
auc_tfidf_L1 = auc(fpr_test,tpr_test)
# plot roc
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)
```



In [59]:

```
# plot confusion matrix
plot_Confusion_Matrix(scores_train,pred_train,"Confusion Matrix on Train data")
plot_Confusion_Matrix(scores_train,pred_train,"Confusion Matrix on Train data")
plot_Confusion_Matrix(scores_test,pred_test,"Confusion Matrix on Test data")
print("C =",optimal_C_tfidf_L1)
print("AUC =",auc_tfidf_L1)
```



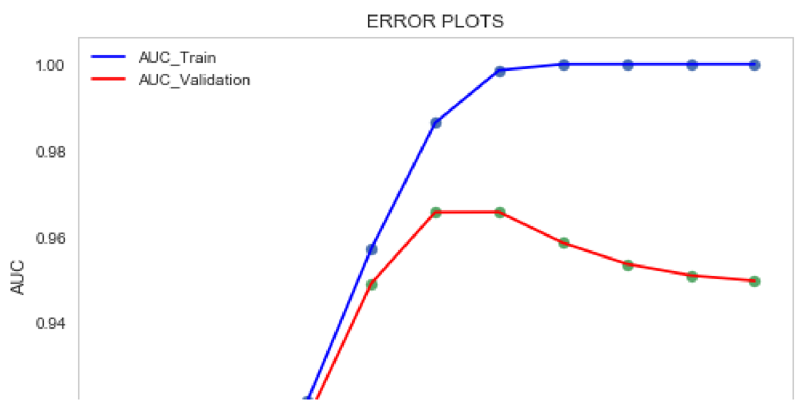


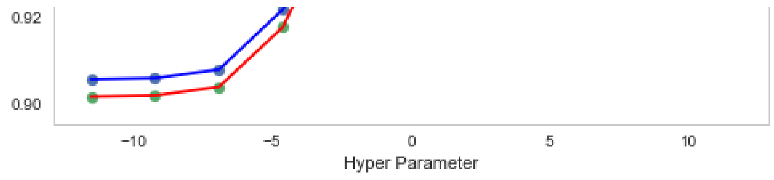
```
C = 1
AUC = 0.9593563780049155
```

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

In [60]:

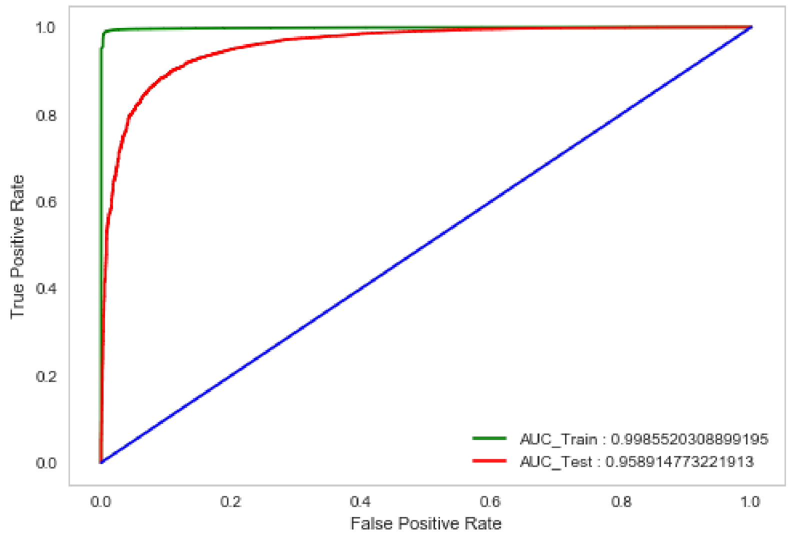
```
# Please write all the code with proper documentation
# dictionary C
hyper_parameter = {"C" : [10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4,10**5]}
# AUC
# C range
C = hyper_parameter.get("C")

train_auc,cv_auc = get_AUC(data_train_tf_idf,scores_train,data_cv_tf_idf,scores_cv,C,'l2')

# plot auc
plot_AUC_Curves(train_auc,cv_auc,C)
```





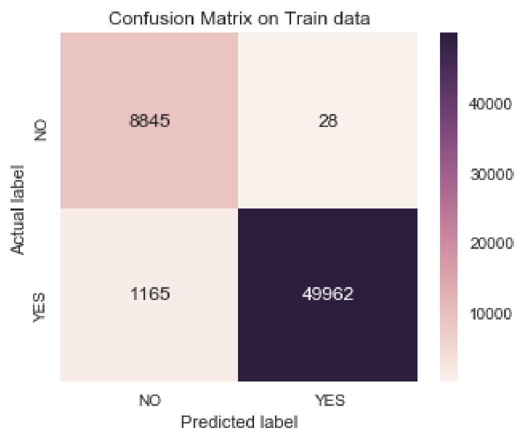
In [61]:

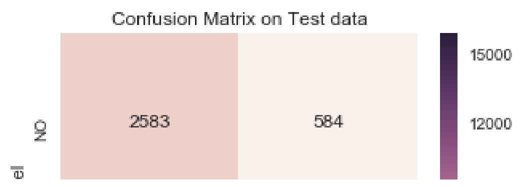
```
# Please write all the code with proper documentation
# optimal alpha
optimal_C_tfidf_L2 = C [cv_auc.index(max(cv_auc))]
# roc
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test,w = apply_roc_curve(data_train_tf_idf,s
cores_train,data_test_tf_idf,scores_test,optimal_C_tfidf_L2,'l2')
# auc
auc_tfidf_L2 = auc(fpr_test,tpr_test)
# plot roc
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)
```

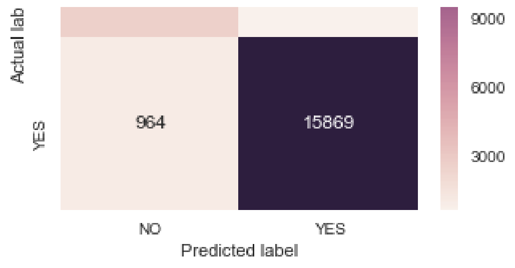


In [62]:

```
# plot confusion matrix
plot_Confusion_Matrix(scores_train,pred_train,"Confusion Matrix on Train data")
plot_Confusion_Matrix(scores_test,pred_test,"Confusion Matrix on Test data")
print("C =",optimal_C_tfidf_L2)
print("AUC =",auc_tfidf_L2)
```







```
C = 10
AUC = 0.958914773221913
```

[5.2.3] Feature Importance on TFIDF, SET 2

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

In [63]:

```
# Please write all the code with proper documentation
weights = w[0,:]
top10_positive_indices = weights.argsort()[-10:]
rank = np.array(range(1,11))
top10_positive_features = np.take(tf_idf_vect.get_feature_names(),top10_positive_indices)
prob = np.take(weights,top10_positive_indices)
top10_positive_features_details = pd.DataFrame(data = {'Rank' : rank,'Feature' : top10_positive_fea
tures,'Probability' : prob})
print(top10_positive_features_details)
```

```
            Feature  Probability  Rank
0  not disappointed    11.057132     1
1              love    11.150720     2
2         wonderful    11.157929     3
3         excellent    11.181810     4
4              good    11.674952     5
5           perfect    12.356331     6
6             loves    12.785028     7
7         delicious    14.271161     8
8              best    14.562056     9
9             great    19.078165    10
```

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

In [64]:

```
# Please write all the code with proper documentation
weights = w[0,:]
top10_negative_indices = weights.argsort()[:10]
rank = np.array(range(1,11))
top10_negative_features = np.take(tf_idf_vect.get_feature_names(),top10_negative_indices)
prob = np.take(weights,top10_negative_indices)
top10_negative_features_details = pd.DataFrame(data = {'Rank' : rank,'Feature' : top10_negative_fea
tures,'Probability' : prob})
print(top10_negative_features_details)
```

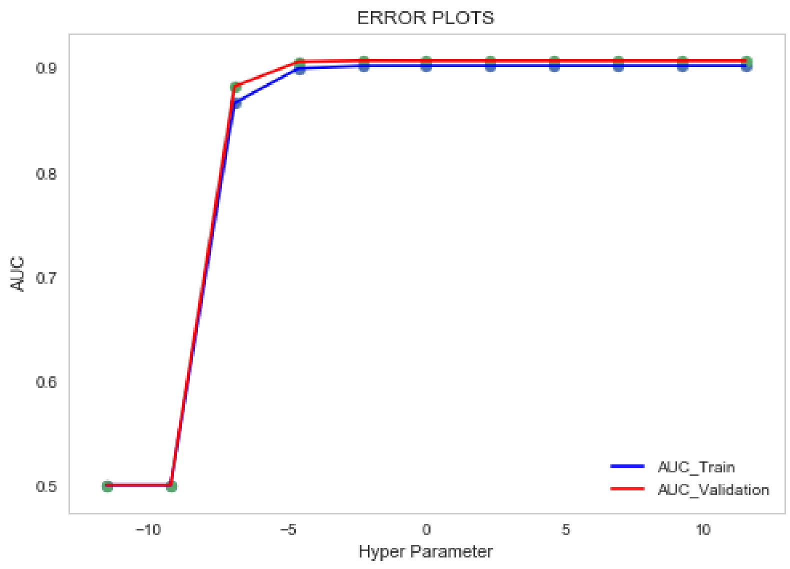
```
         Feature  Probability  Rank
0          worst   -15.195919     1
1   disappointed   -14.656599     2
2      not worth   -14.027856     3
3  not recommend   -12.447222     4
4  disappointing   -11.886147     5
5       not good   -11.189009     6
6          awful   -10.883979     7
7       terrible   -10.653285     8
8        not buy   -10.404479     9
9       not work   -10.339158    10
```

[5.3] Logistic Regression on AVG W2V, SET 3

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

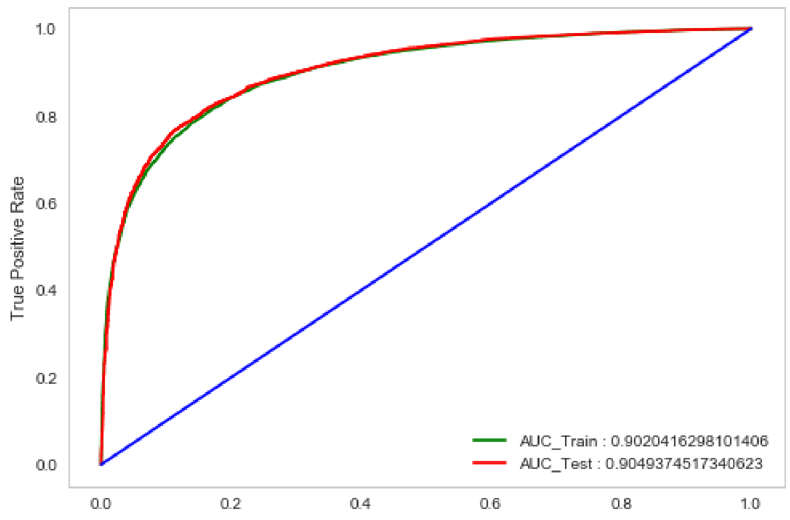
In [65]:

```
# Please write all the code with proper documentation
# dictionary C
hyper_parameter = {"C" : [10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4,10**5]}
# C range
C = hyper_parameter.get("C")

# AUC
train_auc,cv_auc = get_AUC(data_train_avg_w2v,scores_train,data_cv_avg_w2v,scores_cv,C,'l1')

# plot auc
plot_AUC_Curves(train_auc,cv_auc,C)
```



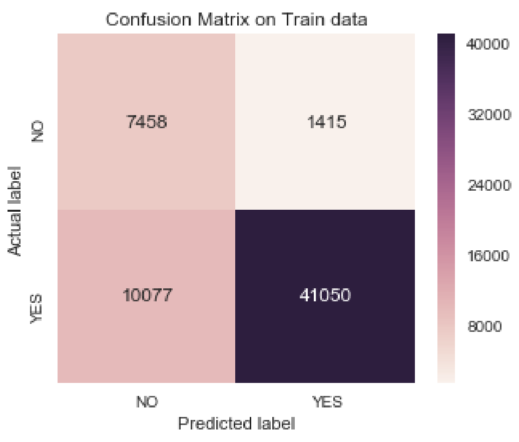
In [66]:

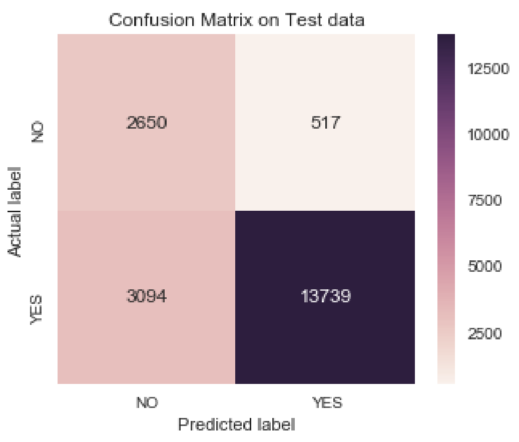
```
# Please write all the code with proper documentation
# optimal alpha
optimal_C_avgw2v_L1 = C [cv_auc.index(max(cv_auc))]
#optimal_C_avgw2v_L1 = 0.1
# roc
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test,w = apply_roc_curve(data_train_avg_w2v,
scores_train,data_test_avg_w2v,scores_test,optimal_C_avgw2v_L1,'l1')
# auc
auc_avgw2v_L1 = auc(fpr_test,tpr_test)
# plot roc
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)
```



In [67]:

```
# plot confusion matrix
plot_Confusion_Matrix(scores_train,pred_train,"Confusion Matrix on Train data")
plot_Confusion_Matrix(scores_test,pred_test,"Confusion Matrix on Test data")
print("C =",optimal_C_avgw2v_L1)
print("AUC =",auc_avgw2v_L1)
```



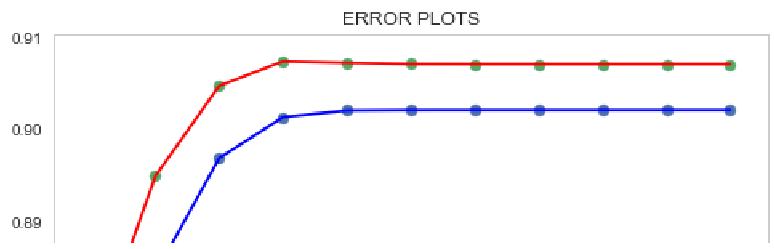


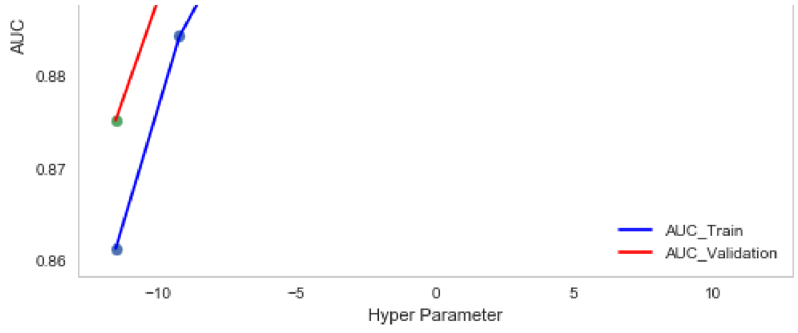
```
C = 0.1
AUC = 0.9049374517340623
```

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

In [68]:

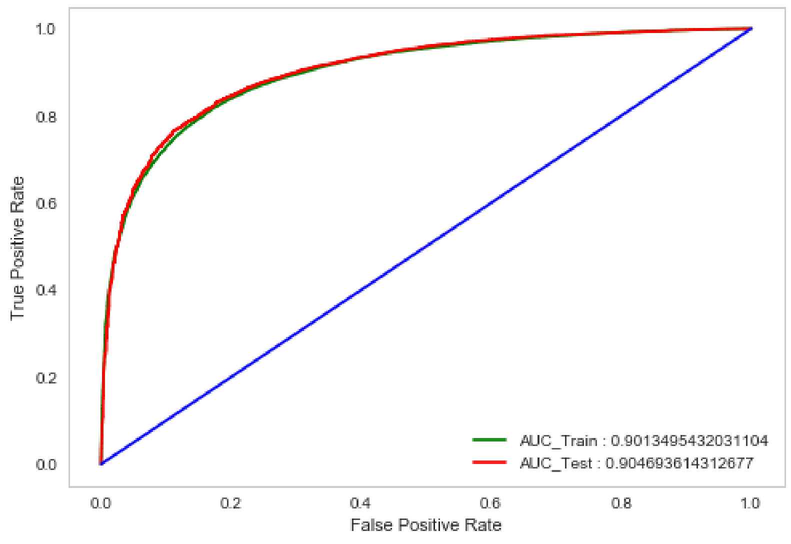
```
# Please write all the code with proper documentation
# dictionary C
hyper_parameter = {"C" : [10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4,10**5]}
# C range
C = hyper_parameter.get("C")
# AUC
train_auc,cv_auc = get_AUC(data_train_avg_w2v,scores_train,data_cv_avg_w2v,scores_cv,C,'l2')

# plot auc
plot_AUC_Curves(train_auc,cv_auc,C)
```





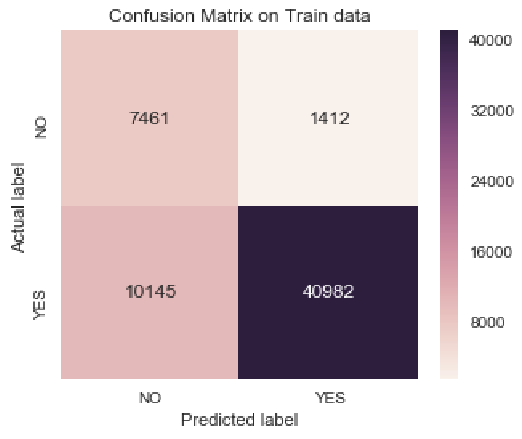
In [69]:

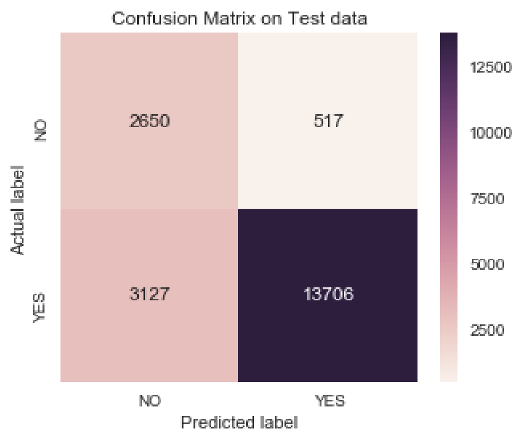
```
# Please write all the code with proper documentation
# optimal alpha
optimal_C_avgw2v_L2 = C [cv_auc.index(max(cv_auc))]
#optimal_C_avgw2v_L2 = 1
# roc
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test,w = apply_roc_curve(data_train_avg_w2v,
scores_train,data_test_avg_w2v,scores_test,optimal_C_avgw2v_L2,'l2')
# auc
auc_avgw2v_L2 = auc(fpr_test,tpr_test)
# plot roc
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)
```



In [70]:

```
# plot confusion matrix
plot_Confusion_Matrix(scores_train,pred_train,"Confusion Matrix on Train data")
plot_Confusion_Matrix(scores_test,pred_test,"Confusion Matrix on Test data")
print("C =",optimal_C_avgw2v_L2)
print("AUC =",auc_avgw2v_L2)
```





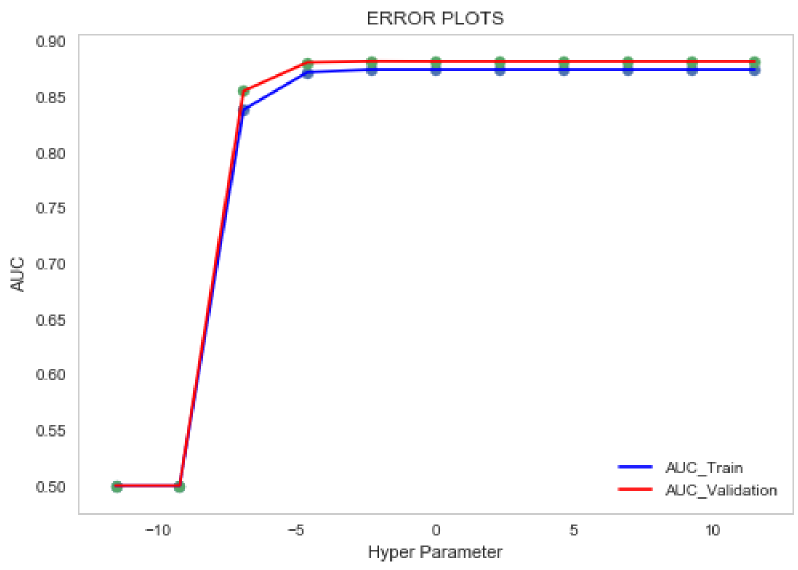
```
C = 0.01
AUC = 0.904693614312677
```

[5.4] Logistic Regression on TFIDF W2V, SET 4

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

In [71]:

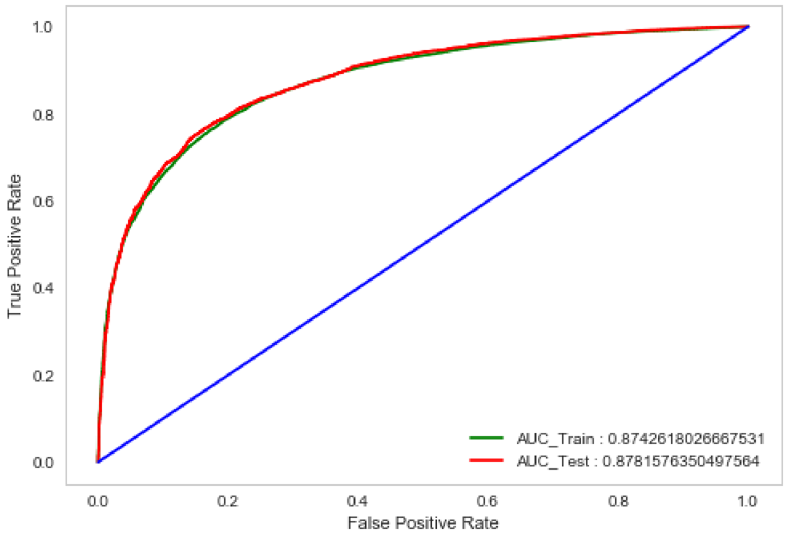
```
# Please write all the code with proper documentation
# dictionary C
hyper_parameter = {"C" : [10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4,10**5]}
# C range
C = hyper_parameter.get("C")
# AUC
train_auc,cv_auc = get_AUC(data_train_tf_idf_w2v,scores_train,data_cv_tf_idf_w2v,scores_cv,C,'l1')

# plot auc
plot_AUC_Curves(train_auc,cv_auc,C)
```



In [72]:

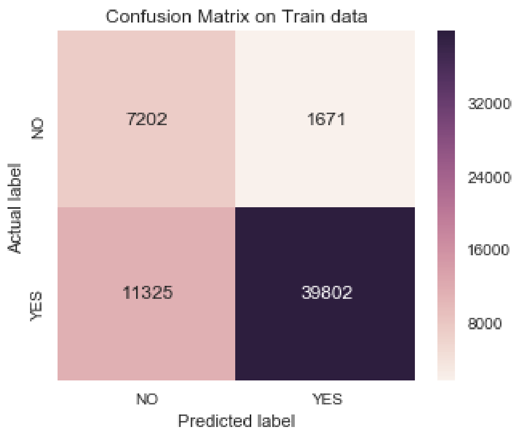
```
# Please write all the code with proper documentation
# optimal alpha
optimal_C_tfidfw2v_L1 = C [cv_auc.index(max(cv_auc))]
# roc
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test,w =
apply_roc_curve(data_train_tf_idf_w2v,scores_train,data_test_tf_idf_w2v,scores_test,optimal_C_tfidf
w2v_L1,'l1')
# auc
auc_tfidfw2v_L1 = auc(fpr_test,tpr_test)
# plot roc
```

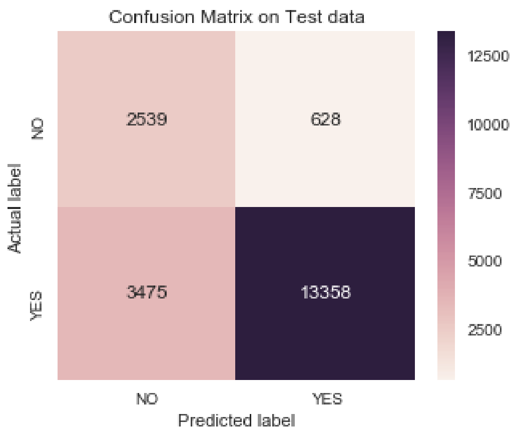
```
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)
```



In [73]:

```
# plot confusion matrix
plot_Confusion_Matrix(scores_train,pred_train,"Confusion Matrix on Train data")
plot_Confusion_Matrix(scores_test,pred_test,"Confusion Matrix on Test data")
print("C =",optimal_C_tfidfw2v_L1)
print("AUC =",auc_tfidfw2v_L1)
```



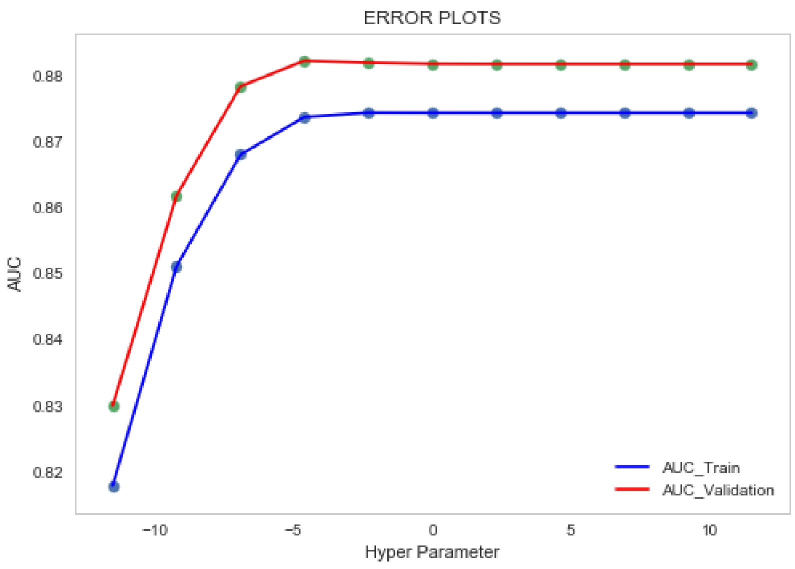


```
C = 0.1
AUC = 0.8781576350497564
```

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

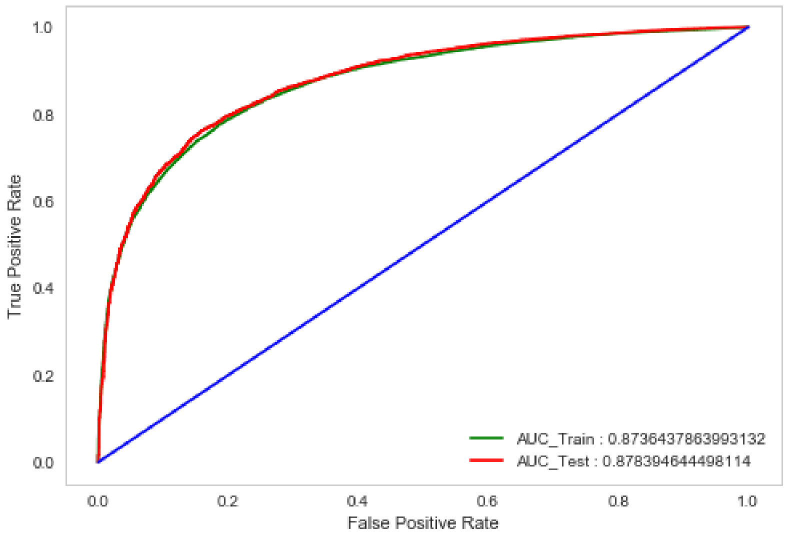
In [74]:

```
# Please write all the code with proper documentation
# dictionary C
hyper_parameter = {"C" : [10**-5,10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4,10**5]}
# C range
C = hyper_parameter.get("C")

# AUC
train_auc,cv_auc = get_AUC(data_train_tf_idf_w2v,scores_train,data_cv_tf_idf_w2v,scores_cv,C,'l2')

# plot auc
plot_AUC_Curves(train_auc,cv_auc,C)
```



In [75]:

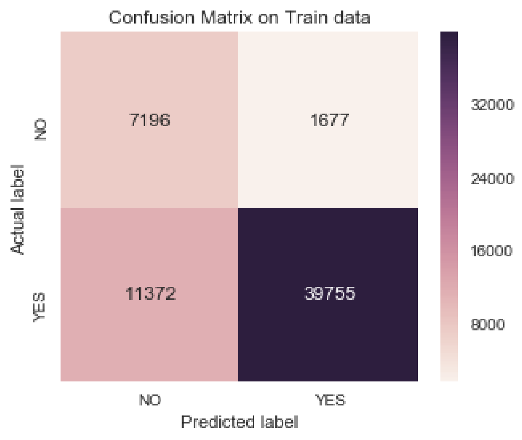
```
# Please write all the code with proper documentation
# optimal alpha
optimal_C_tfidfw2v_L2 = C [cv_auc.index(max(cv_auc))]
# roc
fpr_train,tpr_train,fpr_test,tpr_test,pred_train,pred_test,w =
apply_roc_curve(data_train_tf_idf_w2v,scores_train,data_test_tf_idf_w2v,scores_test,optimal_C_tfidf
w2v_L2,'l2')
# auc
auc_tfidfw2v_L2 = auc(fpr_test,tpr_test)
# plot roc
plot_roc_curve(fpr_train,tpr_train,fpr_test,tpr_test)
```

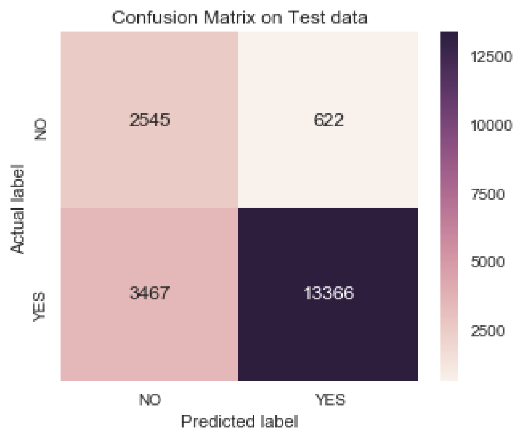


In [76]:

```
# plot confusion matrix
```

```
plot_Confusion_Matrix(scores_train,pred_train,"Confusion Matrix on Train data")
plot_Confusion_Matrix(scores_test,pred_test,"Confusion Matrix on Test data")
print("C =",optimal_C_tfidfw2v_L2)
print("AUC =",auc_tfidfw2v_L2)
```





```
C = 0.01
AUC = 0.878394644498114
```

[6] Conclusions

In [77]:

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

from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ["Vectorizer","Regularization","Hyper parameter","AUC"]
table.add_row(["BOW","L1",optimal_C_bow_L1,round(auc_bow_L1,2)])
table.add_row(["BOW","L2",optimal_C_bow_L2,round(auc_bow_L2,2)])
table.add_row(["TFIDF","L1",optimal_C_tfidf_L1,round(auc_tfidf_L1,2)])
table.add_row(["TFIDF","L2",optimal_C_tfidf_L2,round(auc_tfidf_L2,2)])
table.add_row(["AVG W2V","L1",optimal_C_avgw2v_L1,round(auc_avgw2v_L1,2)])
table.add_row(["AVG W2V","L2",optimal_C_avgw2v_L2,round(auc_avgw2v_L2,2)])
table.add_row(["TFIDF W2V","L1",optimal_C_tfidfw2v_L1,round(auc_tfidfw2v_L1,2)])
table.add_row(["TFIDF W2V","L2",optimal_C_tfidfw2v_L2,round(auc_tfidfw2v_L2,2)])

print(table.get_string(title="Results"))
```

```
+--------------------------------------------------------+
|                        Results                         |
+------------+----------------+-----------------+------+
| Vectorizer | Regularization | Hyper parameter | AUC  |
+------------+----------------+-----------------+------+
|    BOW     |       L1       |       0.1       | 0.94 |
|    BOW     |       L2       |       0.1       | 0.94 |
|   TFIDF    |       L1       |        1        | 0.96 |
|   TFIDF    |       L2       |        10       | 0.96 |
|  AVG W2V   |       L1       |       0.1       | 0.9  |
|  AVG W2V   |       L2       |       0.01      | 0.9  |
| TFIDF W2V  |       L1       |       0.1       | 0.88 |
| TFIDF W2V  |       L2       |       0.01      | 0.88 |
+------------+----------------+-----------------+------+
```

In [ ]: