Assignment05 - Amazon Fine Food Reviews Analysis_Logistic Regression

June 5, 2019

1 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.

2 [1]. Reading Data

2.1 [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.
db_path = '/home/monodeepdas112/Datasets/amazon-fine-food-reviews/database.sqlite'
# db_path = '/home/monodeepdas112/Datasets/AmazonFineFoodReviews/database.sqlite'
con = sqlite3.connect(db_path)

# 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 point
# 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 5
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 500

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negativ
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 (500000, 10)
```

Out[2]:

```
   Id   ProductId          UserId                      ProfileName  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW                       delmartian
1   2  B00813GRG4  A1D87F6ZCVE5NK                           dll pa
2   3  B000LQOCH0   ABXLMWJIXXAIN  Natalia Corres "Natalia Corres"

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     1                       1      1  1303862400
1                     0                       0      0  1346976000
2                     1                       1      1  1219017600

                 Summary                                               Text
0  Good Quality Dog Food  I have bought several of the Vitality canned d...
1      Not as Advertised  Product arrived labeled as Jumbo Salted Peanut...
2  "Delight" says it all  This is a confection that has been around a fe...
```

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  \
        0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1
        3  #oc-R11O5J5ZVQE25C  B005HG9ET0           Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBE1U   Christopher P. Presta  1348617600      1

                                                        Text  COUNT(*)
        0  Overall its just OK when considering the price...         2
        1  My wife has recurring extreme muscle spasms, u...         3
        2  This coffee is horrible and unfortunately not ...         2
        3  This will be the bottle that you grab from the...         3
        4  I didnt like this coffee. Instead of telling y...         2
```

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

```
Out[5]:               UserId   ProductId                      ProfileName        Time  \
        80638  AZY10LLTJ71NX  B006P7E5ZI  undertheshrine "undertheshrine"  1334707200

               Score                                               Text  COUNT(*)
        80638      5  I was recommended to try green tea extract to ...         5
```

```
In [6]: display['COUNT(*)'].sum()
```

```
Out[6]: 393063
```

3 [2] Exploratory Data Analysis

3.1 [2.1] Data Cleaning: Deduplication

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

```
In [7]: display= pd.read_sql_query("""
        SELECT *
        FROM Reviews
        WHERE Score != 3 AND UserId="AR5J8UI46CURR"
        ORDER BY ProductID
        """, con)
        display.head()
```

```
Out[7]:        Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
        0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
        1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
        2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
```

```
3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                      2
4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                      2

   HelpfulnessDenominator  Score        Time  \
0                       2      5  1199577600
1                       2      5  1199577600
2                       2      5  1199577600
3                       2      5  1199577600
4                       2      5  1199577600

                            Summary  \
0  LOACKER QUADRATINI VANILLA WAFERS
1  LOACKER QUADRATINI VANILLA WAFERS
2  LOACKER QUADRATINI VANILLA WAFERS
3  LOACKER QUADRATINI VANILLA WAFERS
4  LOACKER QUADRATINI VANILLA WAFERS

                                               Text
0  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
1  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
2  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
3  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
4  DELICIOUS WAFERS. I FIND THAT EUROPEAN WAFERS ...
```

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=Fals
```

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

Out[9]:
```
(348262, 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]: 69.6524

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  \
0  64422  B000MIDROQ  A161DK06JJMCYF  J. E. Stephens "Jeanne"
1  44737  B001EQ55RW  A2V0I904FH7ABY                      Ram

   HelpfulnessNumerator  HelpfulnessDenominator  Score        Time  \
0                     3                       1      5  1224892800
1                     3                       2      4  1212883200

                                   Summary  \
0            Bought This for My Son at College
1  Pure cocoa taste with crunchy almonds inside

                                                Text
0  My son loves spaghetti so I didn't hesitate or...
1  It was almost a 'love at first bite' - the per...
```

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()
```

```
(348260, 10)
```

Out[13]:
```
1    293516
0     54744
Name: Score, dtype: int64
```

4 [3] Preprocessing

4.1 [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]:
```
# 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 book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and l
==================================================
I've purchased both the Espressione Espresso (classic) and the 100% Arabica.  My vote is defin
==================================================
This is a great product. It is very healthy for all of our dogs, and it is the first food that
==================================================
I find everything I need at Amazon so I always look there first. Chocolate tennis balls for a
==================================================
```

In [15]:
```
# 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 book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and l
```

In [16]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
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 book was purchased as a birthday gift for a 4 year old boy. He squealed with delight and l
==================================================
I've purchased both the Espressione Espresso (classic) and the 100% Arabica.  My vote is defini
==================================================
This is a great product. It is very healthy for all of our dogs, and it is the first food that
==================================================
I find everything I need at Amazon so I always look there first. Chocolate tennis balls for a t
```

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)
```

```
This is a great product. It is very healthy for all of our dogs, and it is the first food that
==================================================
```

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 book was purchased as a birthday gift for a  year old boy. He squealed with delight and hu
```

In [20]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
This is a great product It is very healthy for all of our dogs and it is the first food that tl
```

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', 'ourselve
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'h
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'n
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't"
```

```
                    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
                    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn'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 stopw
    preprocessed_reviews.append(sentance.strip())
```

```
100%|| 348260/348260 [02:42<00:00, 2137.14it/s]
```

In [23]:
```
preprocessed_reviews[1500]
```

Out[23]:
```
'great product healthy dogs first food love eat helped older dog lose weight year old
```

[3.2] Preprocessing Review Summary

In [24]:
```
# ## Similartly you can do preprocessing for review summary also.
# # 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 sto
#     preprocessed_summary.append(sentance.strip())
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

In [25]:
```
# #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])
```

5.2 [4.2] Bi-Grams and n-Grams.

In [26]:
```
# #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/m

# # 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_bigr
```

5.3 [4.3] TF-IDF

In [27]:
```
# 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_na
# 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_i
```

5.4 [4.4] Word2Vec

In [28]:
```
# # 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=True
# want_to_use_google_w2v = False
# want_to_train_w2v = True

# path_to_word2vec = '/home/monodeepdas112/Datasets/GoogleNews-vectors-negative300.bi

# if want_to_train_w2v:

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

#     # min_count = 5 considers only words that occured atleast 5 times
#     w2v_model=Word2Vec(list_of_sentences,min_count=5,size=100, 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(path_to_word2vec):
#         print('Preparing to load pre-trained Word2Vec model !')
#         w2v_model=KeyedVectors.load_word2vec_format(path_to_word2vec, binary=True)
#         print('Successfully loaded model into memory !!')
#         print('Words similar to "similar" : ', w2v_model.wv.most_similar('great'))
#         print('Words similar to "worst" : ',w2v_model.wv.most_similar('worst'))
#     else:
#         print("you don't have google's word2vec file, keep want_to_train_w2v = True
```

In [29]:
```
# 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])
```

5.5 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [30]:
```
# # 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_sentences): # for each review/sentence
#     sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need
#     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]))
```

[4.4.1.2] TFIDF weighted W2v

In [31]:
```
# # 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 [32]:
```
# # 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 = tfi

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

6 [5] Assignment 5: Apply Logistic Regression

```
<li><strong>Apply Logistic Regression on these feature sets</strong>
    <ul>
        <li><font color='red'>SET 1:</font>Review text, preprocessed one converted into vectors
```

```
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vectors
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vectors
    </ul>
</li>
<br>
<li><strong>Hyper paramter tuning (find best hyper parameters corresponding the algorithm that
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>Find the best hyper paramter using k-fold cross validation or simple cross validation data
<li>Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this ta
    </ul>
</li>
<br>
<li><strong>Pertubation Test</strong>
    <ul>
<li>Get the weights W after fit your model with the data X i.e Train data.</li>
<li>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)

```
<li>Fit the model again on data X' and get the weights W'</li>
<li>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

```
<li>Now find the % change between W and W' (| (W-W') / (W) |)*100)</li>
<li>Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in t
<li> Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is suc
        <li> Print the feature names whose % change is more than a threshold x(in our example i
    </ul>
</li>
<br>
<li><strong>Sparsity</strong>
    <ul>
<li>Calculate sparsity on weight vector obtained after using L1 regularization</li>
    </ul>
</li>
<br><font color='red'>NOTE: Do sparsity and multicollinearity for any one of the vectorizers. B
<br>
<br>
<li><strong>Feature importance</strong>
    <ul>
<li>Get top 10 important features for both positive and negative classes separately.</li>
    </ul>
</li>
<br>
<li><strong>Feature engineering</strong>
    <ul>
```

```
<li>To increase the performance of your model, you can also experiment with with feature engine
        <ul>
        <li>Taking length of reviews as another feature.</li>
        <li>Considering some features from review summary as well.</li>
    </ul>
    </ul>
</li>
<br>
<li><strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='train_cv_auc.JPG' width=300px></li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and fi
<img src='train_test_auc.JPG' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.c
<img src='confusion_matrix.png' width=300px></li>
    </ul>
</li>
<br>
<li><strong>Conclusion</strong>
    <ul>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='summary.JPG' width=400px>
</li>
    </ul>
```

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.

7 Applying Logistic Regression

In [33]:
```
#Getting the necessary imports and function definations
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
from scipy.stats import uniform
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import StratifiedKFold
import pprint
from sklearn.pipeline import Pipeline
```

```
import os.path
import pickle
import pprint
from sklearn import preprocessing
import math

import warnings
warnings.filterwarnings('ignore')
```

7.0.1 [5.0.0] Splitting up the Dataset into D_train and D_test

```
In [34]: num_data_points = 50000
```

```
In [35]: Dx_train, Dx_test, Dy_train, Dy_test = train_test_split(preprocessed_reviews[:num_data
```

```
In [36]: prettytable_data = []
```

7.0.2 [5.0.1] Defining some functions to increase code reusability and readability

```
In [37]: '''Creating Custom Vectorizers for TFIDF - W2Vec and Avg - W2Vec'''
         class Tfidf_W2Vec_Vectorizer(object):
             def __init__(self, w2vec_model):
                 if(w2v_model is None):
                     raise Exception('Word 2 Vector model passed to Tfidf_W2Vec Vectorizer is N
                 self.tfidf = TfidfVectorizer(max_features=300)
                 self.dictionary = None
                 self.tfidf_feat = None

                 self.word2vec = w2vec_model

             def fit(self, X):
                 '''X : list'''
                 #Initializing the TFIDF Vectorizer
                 self.tfidf.fit_transform(X)
                 # we are converting a dictionary with word as a key, and the idf as a value
                 self.dictionary = dict(zip(self.tfidf.get_feature_names(), list(self.tfidf.idf
                 self.tfidf_feat = self.tfidf.get_feature_names()

                 return self

             def transform(self, X):
                 '''X : list'''
                 return np.array([
                         np.mean([self.word2vec[w] * self.dictionary[word]*(X.cout(word)/len(X)
                                  for w in words if w in self.word2vec and w in self.tfidf_feat
                                 [np.zeros(300)], axis=0)
                         for words in X
                     ])
```

```
class Avg_W2Vec_Vectorizer(object):
    def __init__(self, w2vec_model):
        if(w2v_model is None):
            raise Exception('Word 2 Vector model passed to Avg_W2Vec Vectorizer is No
        self.word2vec = w2vec_model

    def fit(self, X):
        return self

    def transform(self, X):
        '''X : list'''
        return np.array([
            np.mean([self.word2vec[w] for w in words if w in self.word2vec]
                    or [np.zeros(300)], axis=0)
            for words in X
        ])
```

In [38]:
```
def get_vectorizer(vectorizer, train, W2V_model=None):
    if(vectorizer=='BOW'):
        vectorizer = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=1000)
    if(vectorizer=='TFIDF'):
        vectorizer = TfidfVectorizer(ngram_range=(1,2), min_df=10, max_features=1000)
    if(vectorizer=='TFIDF-W2Vec'):
        vectorizer = Tfidf_W2Vec_Vectorizer(W2V_model)
    if(vectorizer=='Avg-W2Vec'):
        vectorizer = Avg_W2Vec_Vectorizer(W2V_model)

    vectorizer.fit(train)
    return vectorizer
```

In [39]:
```
'''Perform Simple Cross Validation'''
def perform_hyperparameter_tuning(X, Y, vectorizer, vec_name, penalty, results_path, r
    #If the pandas dataframe with the hyperparameter info exists then return it

    if(retrain==False):
        # If Cross Validation results exists then return them
        if(os.path.exists(results_path)):
            return pd.read_csv(results_path)
        else:
            # If no data exists but retrain=False then mention accordingly
            print('Retrain is set to be False but no Cross Validation Results DataFran
    else:
        # else perform hyperparameter tuning
        print('Performing Hyperparameter Tuning...\n')
        # regularization parameter
        c = [0.0001, 0.001, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000, 5000,
        c.sort()
        hyperparameters = {
```

```
    'logistic__penalty' : penalty,
    'logistic__C' : c
}

penalties = []
C_values = []

train_scores = []
test_scores = []

train_mean_score = []
test_mean_score = []

# Initializing KFold
skf = StratifiedKFold(n_splits=3)
X = np.array(X)
Y = np.array(Y)

saver = 0 # This is a counter variable that saves
for penalty in hyperparameters['logistic__penalty']:
    for reg_param in hyperparameters['logistic__C']:

        #Performing Cross Validation
        for train_index, test_index in skf.split(X, Y):
            Dx_train, Dx_cv = X[train_index], X[test_index]
            Dy_train, Dy_cv = Y[train_index], Y[test_index]

            #Initializing the Vectorizer
            vectorizer = get_vectorizer(vectorizer, Dx_train.tolist(), W2V_mod

            #Transforming the data to features
            x_train = vectorizer.transform(Dx_train.tolist())
            x_cv = vectorizer.transform(Dx_cv.tolist())

            #Initializing the LR model
            log_reg_model = LogisticRegression(penalty=penalty, C=reg_param, r

            # Fit the model
            log_reg_model.fit(x_train, Dy_train)

            #Prediction
            train_results = log_reg_model.predict_proba(x_train)
            cv_results = log_reg_model.predict_proba(x_cv)

            try:
                train_score = roc_auc_score(Dy_train, train_results[:, 1])
                test_score = roc_auc_score(Dy_cv, cv_results[:, 1])
```

```
                #storing the results to form a dataframe
                train_scores.append(train_score)
                test_scores.append(test_score)

            except Exception as e:
                print('Error Case : ', e)
                print(('Actual, Predicted'))
                [print((Dy_cv[i], cv_results[i, 1])) for i in range(len(Dy_cv

            print('CV iteration : C={0}, solver={1}, train_score={2}, test_sco
              .format(reg_param, 'saga', train_score, test_score))

        train_mean_score.append(sum(train_scores)/len(train_scores))
        test_mean_score.append(sum(test_scores)/len(test_scores))

        penalties.append(penalty)
        C_values.append(reg_param)

        print('C={0}, penalty={1}, solver="saga", train_score={2}, test_score
              .format(reg_param, penalty, sum(train_scores)/len(train_scores)

        saver += 1

        if(saver==5):
            # after every period of 100 iterations keep saving the parameters
            # so as to avoid data loss in case of system crash
            with open('saved_temp_data/{0}_penalties.pkl'.format(vec_name), '
                pickle.dump(penalties, file)
            with open('saved_temp_data/{0}_reg_params.pkl'.format(vec_name),
                pickle.dump(C_values, file)
            with open('saved_temp_data/{0}_train_mean_score.pkl'.format(vec_n
                pickle.dump(train_mean_score, file)
            with open('saved_temp_data/{0}_test_mean_score.pkl'.format(vec_na
                pickle.dump(test_mean_score, file)
            saver = 0

        train_scores = []
        test_scores = []
try:
    # Attempting to load saved data
    # Load data from the saved files
    with open('saved_temp_data/{0}_penalties.pkl'.format(vec_name), 'rb') as f
        penalties = pickle.load(file)
    with open('saved_temp_data/{0}_reg_params.pkl'.format(vec_name), 'rb') as
        C_values = pickle.load(file)
    with open('saved_temp_data/{0}_train_mean_score.pkl'.format(vec_name), 'rb
        train_mean_score = pickle.load(file)
    with open('saved_temp_data/{0}_test_mean_score.pkl'.format(vec_name), 'rb
```

```
                test_mean_score = pickle.load(file)
        except Exception as ex:
            print('Failed to load saved data from temp files')
        # Creating a DataFrame from the saved data for visualization
        results_df = pd.DataFrame({'C' : C_values, 'penalty' : penalties,
                                   'solver' : ['saga' for i in C_values], 'train_score
                                   'test_score': test_mean_score})

        try:
            # Attempting to remove the temporary files
            os.remove('saved_temp_data/{0}_penalties.pkl'.format(vec_name))
            os.remove('saved_temp_data/{0}_reg_params.pkl'.format(vec_name))
            os.remove('saved_temp_data/{0}_train_mean_score.pkl'.format(vec_name))
            os.remove('saved_temp_data/{0}_test_mean_score.pkl'.format(vec_name))
        except Exception as e:
            print('Error occurred while attempting to remove the temporary files')

        #writing the results to csv after performing hyperparameter tuning
        try:
            results_df.to_csv(results_path)
        except Exception as ex:
            print(str(ex), "\nError occured while converting DataFrame to CSV after c
        return results_df
```

```
In [40]: def analyse_results(df):
             # plotting error curves
             fig = plt.figure()
             ax = fig.gca()

             plt.plot([math.log10(i) for i in df.C.tolist()], df.test_score.tolist(), '-o', c=
             plt.plot([math.log10(i) for i in df.C.tolist()], df.train_score.tolist(), '-o', c=
             plt.grid(True)
             plt.xlabel('log10 of Hyperparameter C = 1/alpha')
             plt.ylabel('Area Under ROC Curve')
             plt.title('AUC ROC Curve for Logistic Regression')
             plt.legend(loc='best')
             plt.show()

             # return the best parameters
             mmax = 0
             ind_max = 0
             for index, row in df.iterrows():
                 if(row['test_score']>mmax):
                     mmax=row['test_score']
                     ind_max = index


             best_params = {
```

```
            'logistic__C':df.loc[ind_max, 'C'],
            'logistic__penalty':df.loc[ind_max, 'penalty'],
            'logistic__solver':df.loc[ind_max, 'solver']
        }

        return best_params
```

In [41]:
```
def retrain_with_best_params(data, labels, best_params, vec_name, model_path, word2ve
        if(os.path.exists(model_path)):
            print('Loading Model....')
            with open(model_path, 'rb') as input_file:
                clf = pickle.load(input_file)
        else:
            clf = LogisticRegression(penalty=best_params['logistic__penalty'],
                                     C = best_params['logistic__C'],
                                     solver=best_params['logistic__solver'], max_iter=100(
            print('Initializing Vectorizer')
            vectorizer = get_vectorizer(vectorizer=vec_name, train=data, W2V_model=word2ve
            print('Training Model....')
            clf.fit(vectorizer.transform(data), np.array(labels))

            print('Saving Trained Model....')
            with open(model_path,'wb') as file:
                pickle.dump(clf,file)
        return clf
```

In [42]:
```
def plot_confusion_matrix(model, data, labels, dataset_label):
        pred = model.predict(data)
        conf_mat = confusion_matrix(labels, pred)

        strings = strings = np.asarray([['TN = ', 'FP = '],
                                        ['FN = ', 'TP = ']])

        labels = (np.asarray(["{0}{1}".format(string, value)
                              for string, value in zip(strings.flatten(),
                                                       conf_mat.flatten())])
                 ).reshape(2, 2)

        fig, ax = plt.subplots()
        ax.set(xlabel='Predicted', ylabel='Actual', title='Confusion Matrix : {0}'.format
        sns.heatmap(conf_mat, annot=labels, fmt="", cmap='YlGnBu', ax=ax)
        ax.set_xlabel('Predicted')
        ax.set_ylabel('Actual')
        ax.set_xticklabels(['False', 'True'])
        ax.set_yticklabels(['False', 'True'])
        plt.show()
```

In [43]:
```
def plot_AUC_ROC(model, vectorizer, Dx_train, Dx_test, Dy_train, Dy_test):
```

```
    #predicting probability of Dx_test, Dx_train
    test_score = model.predict_proba(vectorizer.transform(Dx_test))
    train_score = model.predict_proba(vectorizer.transform(Dx_train))

    #Finding out the ROC_AUC_SCORE
    train_roc_auc_score = roc_auc_score(np.array(Dy_train), train_score[:, 1])
    print('Area Under the Curve for Train : ', train_roc_auc_score)
    test_roc_auc_score = roc_auc_score(np.array(Dy_test), test_score[:, 1])
    print('Area Under the Curve for Test : ', test_roc_auc_score)

    #Plotting with matplotlib.pyplot
    #ROC Curve for D-train
    train_fpr, train_tpr, thresholds = roc_curve(np.array(Dy_train), train_score[:, 1
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))

    # #ROC Curve for D-test
    test_fpr, test_tpr, thresholds = roc_curve(np.array(Dy_test), test_score[:, 1])
    plt.plot(test_fpr, test_tpr, label="train AUC ="+str(auc(test_fpr, test_tpr)))

    plt.legend()
    plt.xlabel("FPR : False Positive Ratio")
    plt.ylabel("TPF : True Positive Ratio")
    plt.title("Area Under ROC Curve")
    plt.show()

    plot_confusion_matrix(model, vectorizer.transform(Dx_train), np.array(Dy_train),
    plot_confusion_matrix(model, vectorizer.transform(Dx_test), np.array(Dy_test), 'Te
    return train_roc_auc_score, test_roc_auc_score
```

7.1 [5.1.0] Logistic Regression on BOW, SET 1

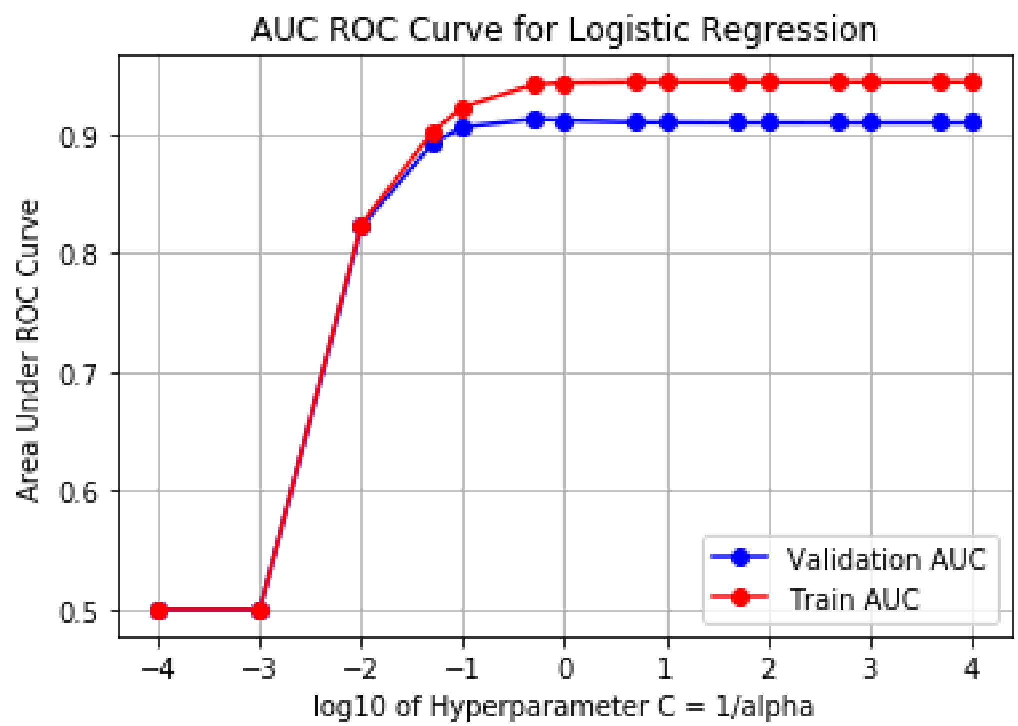
7.1.1 [5.1.0] Applying Logistic Regression with L1 regularization on BOW, SET 1

In [44]:
```
# Please write all the code with proper documentation
csv_path = 'saved_models/Assignment5/BOW_log_reg_results_l1.csv'
cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='BOW', v
                                           penalty=['l1'], results_path=csv_path, ret
# Analysing best parameters
best_parameters = analyse_results(cv_results)
pprint.pprint(best_parameters)
# retraining the model with best parameters
model_path = 'saved_models/Assignment5/{0}_log_reg_l1.pkl'.format('BOW')
log_reg = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'BOW', model_p

print('Retraining Vectorizer with Dx_train')
vectorizer_obj = get_vectorizer(W2V_model = None, train=Dx_train, vectorizer='BOW')

# plotting AUC ROC
```

```
train_score, test_score = plot_AUC_ROC(log_reg, vectorizer_obj, Dx_train, Dx_test, Dy_

# appending the data results
# prettytable_data.append(['BOW', 'LogisticRegression', 'L1', best_parameters['logist
```

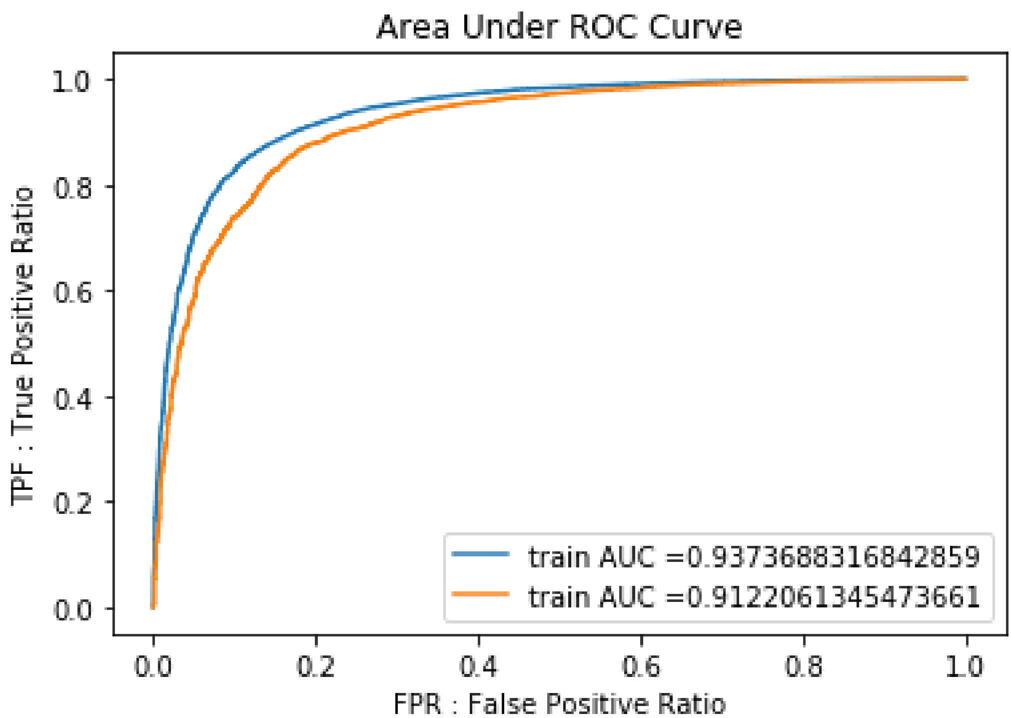
```
Performing Hyperparameter Tuning...

CV iteration : C=0.0001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.0001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.0001, solver=saga, train_score=0.5, test_score=0.5
C=0.0001, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=0.001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.001, solver=saga, train_score=0.5, test_score=0.5
C=0.001, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=0.01, solver=saga, train_score=0.8244418426264546, test_score=0.8206521350463
CV iteration : C=0.01, solver=saga, train_score=0.8247320630638473, test_score=0.8115282727301
CV iteration : C=0.01, solver=saga, train_score=0.8239450915217509, test_score=0.8342770795735
C=0.01, penalty=l1, solver="saga", train_score=0.8243729990706843, test_score=0.822152495783359
CV iteration : C=0.05, solver=saga, train_score=0.9019193121821065, test_score=0.8900017362818
CV iteration : C=0.05, solver=saga, train_score=0.9036854023941612, test_score=0.8929765991173
CV iteration : C=0.05, solver=saga, train_score=0.9015880912165825, test_score=0.8944236887819
C=0.05, penalty=l1, solver="saga", train_score=0.9023976019309501, test_score=0.892467341393748
CV iteration : C=0.1, solver=saga, train_score=0.9216839057870274, test_score=0.904673406082098
CV iteration : C=0.1, solver=saga, train_score=0.9250289150594848, test_score=0.906702048050575
CV iteration : C=0.1, solver=saga, train_score=0.9226615530566592, test_score=0.907432924244072
C=0.1, penalty=l1, solver="saga", train_score=0.9231247913010572, test_score=0.906269459458915
CV iteration : C=0.5, solver=saga, train_score=0.9386440024187235, test_score=0.913281425031369
CV iteration : C=0.5, solver=saga, train_score=0.9453715094188645, test_score=0.913670300587791
CV iteration : C=0.5, solver=saga, train_score=0.9414061945871383, test_score=0.912077270783072
C=0.5, penalty=l1, solver="saga", train_score=0.9418072354749087, test_score=0.913009665467411
CV iteration : C=1, solver=saga, train_score=0.9399987308443226, test_score=0.912962159020207
CV iteration : C=1, solver=saga, train_score=0.9469477978390353, test_score=0.9117054942359779
CV iteration : C=1, solver=saga, train_score=0.9427849911503553, test_score=0.9107327775422377
C=1, penalty=l1, solver="saga", train_score=0.9432438399445711, test_score=0.911800143599474
CV iteration : C=5, solver=saga, train_score=0.9406982466599663, test_score=0.9121590846875297
CV iteration : C=5, solver=saga, train_score=0.9476213150152841, test_score=0.9093168463540602
CV iteration : C=5, solver=saga, train_score=0.9434832554898338, test_score=0.9093009338641114
C=5, penalty=l1, solver="saga", train_score=0.9439342723883614, test_score=0.910258954968567
CV iteration : C=10, solver=saga, train_score=0.9407587294022335, test_score=0.912048412837887
CV iteration : C=10, solver=saga, train_score=0.9476632440498435, test_score=0.908975100123439
CV iteration : C=10, solver=saga, train_score=0.9435407456468976, test_score=0.909113628731798
C=10, penalty=l1, solver="saga", train_score=0.9439875730329915, test_score=0.9100457138977083
CV iteration : C=50, solver=saga, train_score=0.9407979911581177, test_score=0.911949581938260
CV iteration : C=50, solver=saga, train_score=0.9476800684230258, test_score=0.908692500024326
CV iteration : C=50, solver=saga, train_score=0.9435776899971288, test_score=0.908945611045034
C=50, penalty=l1, solver="saga", train_score=0.9440185831927574, test_score=0.9098625643358739
CV iteration : C=100, solver=saga, train_score=0.9408030619186761, test_score=0.911937506513988
```

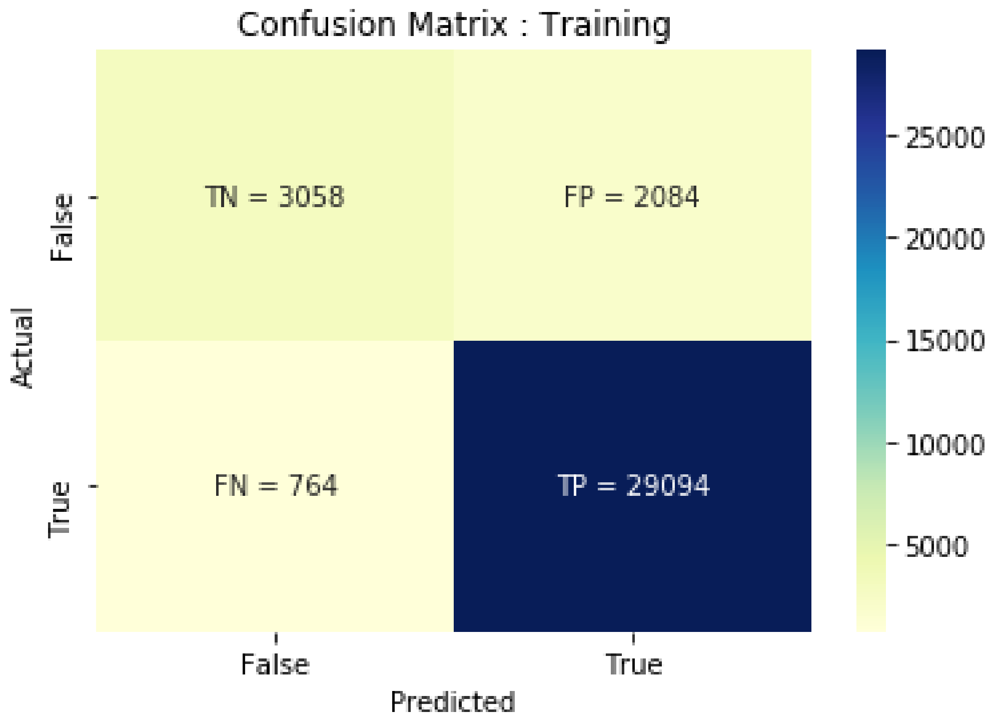
```
CV iteration : C=100, solver=saga, train_score=0.9476835270920771, test_score=0.908653694534674
CV iteration : C=100, solver=saga, train_score=0.943581661404869, test_score=0.908924799363666
C=100, penalty=l1, solver="saga", train_score=0.9440227501385409, test_score=0.909838666804109
CV iteration : C=500, solver=saga, train_score=0.9408071507689529, test_score=0.911930237811998
CV iteration : C=500, solver=saga, train_score=0.947682427938777, test_score=0.908625850716571
CV iteration : C=500, solver=saga, train_score=0.9435838742556762, test_score=0.908908150018572
C=500, penalty=l1, solver="saga", train_score=0.9440244843211354, test_score=0.909821412849047
CV iteration : C=1000, solver=saga, train_score=0.9408063007570675, test_score=0.911929710244
CV iteration : C=1000, solver=saga, train_score=0.9476834831259451, test_score=0.908621161231
CV iteration : C=1000, solver=saga, train_score=0.943584929389836, test_score=0.908905746415935
C=1000, penalty=l1, solver="saga", train_score=0.9440249044242828, test_score=0.909818872630756
CV iteration : C=5000, solver=saga, train_score=0.9408073412888583, test_score=0.911923965625
CV iteration : C=5000, solver=saga, train_score=0.9476831167415117, test_score=0.908619754385
CV iteration : C=5000, solver=saga, train_score=0.9435839035649584, test_score=0.908905453293
C=5000, penalty=l1, solver="saga", train_score=0.9440247871984427, test_score=0.909816391101712
CV iteration : C=10000, solver=saga, train_score=0.9408075611195184, test_score=0.911920448511
CV iteration : C=10000, solver=saga, train_score=0.9476830141538704, test_score=0.908618288921
CV iteration : C=10000, solver=saga, train_score=0.9435859991786366, test_score=0.908904046306
C=10000, penalty=l1, solver="saga", train_score=0.9440255248173418, test_score=0.909814261246
```

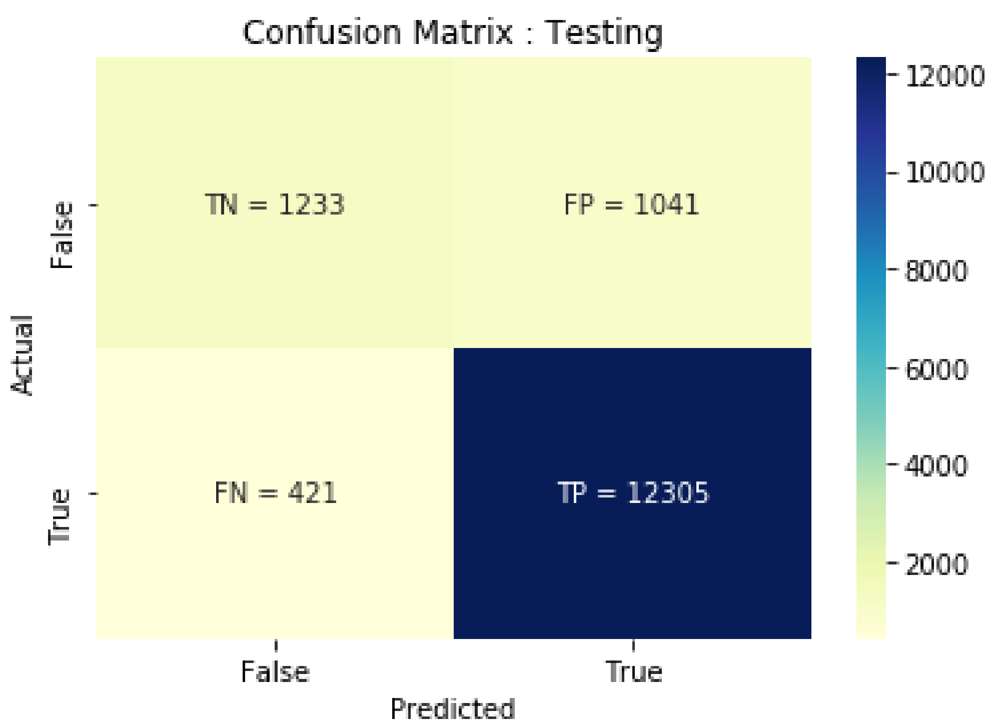



```
{'logistic__C': 0.5, 'logistic__penalty': 'l1', 'logistic__solver': 'saga'}
Initializing Vectorizer
Training Model...
```

```
Saving Trained Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.9373688316842859
Area Under the Curve for Test :  0.9122061345473661
```




Confusion Matrix : Training
TN = 3058
FP = 2084
FN = 764
TP = 29094
Actual
False
True
Predicted
False
True
25000
20000
15000
10000
5000


Confusion Matrix : Testing
TN = 1233
FP = 1041
FN = 421
TP = 12305
Actual
False
True
Predicted
False
True
12000
10000
8000
6000
4000
2000

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

```
In [45]: # # Please write all the code with proper documentation
         # # fetching the coeff and calculating sparsity
         weights = log_reg.coef_
         elements = (weights.shape[0]*weights.shape[1])
         zeroes = elements - np.count_nonzero(weights)
         sparsity = zeroes/elements
         print('Sparsity of the weight vector by L1 reguralization is : ', sparsity)
```

```
Sparsity of the weight vector by L1 reguralization is :  0.137
```

7.1.2 [5.1.0.2] Feature Importance on BOW, SET 1

```
In [46]: # Please write all the code with proper documentation
         feature_names = vectorizer_obj.get_feature_names()
         weights = np.reshape(log_reg.coef_,(log_reg.coef_.shape[1], log_reg.coef_.shape[0]))

         # making a list of feature names along with their feature weights
         features_with_weights = [(feature_names[i],weights[i]) for i in range(len(feature_name
         features_with_weights.sort(key=lambda x : abs(x[1][0]), reverse=True)
```

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

```
In [47]: positive_weights = [i for i in features_with_weights if i[1][0]>=0]
         print('Top 10 features of positive class with the feature names : ')
         for i in positive_weights[:10]:
             print(i[0],' : ',i[1][0])
```

```
Top 10 features of positive class with the feature names :
excellent  :  1.9753053108460117
delicious  :  1.8589530909799676
pleased  :  1.8324072393701578
loves  :  1.6731939586153268
perfect  :  1.5793338333615692
awesome  :  1.5033011438383237
keeps  :  1.4020567949287117
best  :  1.4003388186598489
wonderful  :  1.3736884238797988
amazing  :  1.3533707686471173
```

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

```
In [48]: negative_weights = [i for i in features_with_weights if i[1][0]<0]
         print('Top 10 features of negative class with the feature names : ')
         for i in negative_weights[:10]:
             print(i[0],' : ',i[1][0])
```

```
Top 10 features of negative class with the feature names :
terrible  :  -1.906277472024034
awful  :  -1.7446309138069729
not buy  :  -1.5747627365624441
unfortunately  :  -1.511689812131779
not good  :  -1.4807794459805517
disappointed  :  -1.4336116407383688
horrible  :  -1.3764205956238866
money  :  -1.225686396423233
bland  :  -1.0690434464749512
ended  :  -0.929604547728539
```

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

In [49]:
```
# 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 mat
# 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+
# Now find the % change between W and W' (| (W-W') / (W) |)*100)
# Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden r
# Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there
# Print the feature names whose % change is more than a threshold x(in our example it
```

In [50]:
```
def pertubation_test(model, features, labels, features_with_weights):

    # saving initial model weights
    W = model.coef_

    # change the data slightly
    print('Adding noise to features...')
    noise = np.random.normal(0,0.1,features.shape[0]*features.shape[1])
    noise = np.reshape(noise, features.shape)
    features = features + noise

    # retraining the model with the new features formed by adding errors
    print('Retraining model...')
    model.fit(features, labels)

    # getting the model weights after retraining
    print('Calculating model weights percentage change...')
    _W = model.coef_
    epsilon_val = np.random.normal(0, 0.000005, model.coef_.shape[0]*model.coef_.shape
    epsilon_val = np.reshape(epsilon_val, _W.shape)
    _W = _W + epsilon_val
    W = W + epsilon_val

    percent_change = abs((W-_W)/W)*100
```

```
#     0 : feature name
#     1 : percent change
    feature_percent_change = [(features_with_weights[i][0], percent_change[0][i])for i

    # sorting the data according to the weight values
    feature_percent_change.sort(key=lambda x: x[1])

    # calculating percentile values and displaying
    print('Percentile values of some intervals')
    for i in range(0, 100, 10):
        index = int((i/100)*(percent_change.shape[1]+1))
        print('{0} th percentile : '.format(i),feature_percent_change[index])

    # printing those feature names with percent change difference above threshold
    print('Percent Change above threshold = 2.5')
    for i in range(1, 100):
        indexi = int((i/100)*(percent_change.shape[1]+1))
        indexi_ = int(((i-1)/100)*(percent_change.shape[1]+1))
        difference = abs(feature_percent_change[indexi][1]-feature_percent_change[inde
        if(difference > 2.5):   # taking threshold of 2.5
            print('\n{0} th percentile : '.format(i-1),feature_percent_change[indexi_]
            print('{0} th percentile : '.format(i),feature_percent_change[indexi])
```

```
In [51]: pertubation_test(log_reg, vectorizer_obj.transform(Dx_train), Dy_train, features_with
```

```
Adding noise to features...
Retraining model...
Calculating model weights percentage change...
Percentile values of some intervals
0 th percentile :  ('keeps', 0.0)
10 th percentile :  ('spice', 7.426537586476137)
20 th percentile :  ('not eat', 21.118749524096533)
30 th percentile :  ('tea bags', 31.57523264918191)
40 th percentile :  ('bite', 41.16511261235236)
50 th percentile :  ('yes', 52.37539559627573)
60 th percentile :  ('least', 66.50152315798424)
70 th percentile :  ('apple', 91.66743791896737)
80 th percentile :  ('house', 113.70426985541637)
90 th percentile :  ('food', 299628.3570150673)
Percent Change above threshold = 2.5

63 th percentile :  ('worked', 72.13567526873742)
64 th percentile :  ('milk', 75.26485866456457)

65 th percentile :  ('fun', 76.49652010742746)
66 th percentile :  ('touch', 79.67994081218664)

66 th percentile :  ('touch', 79.67994081218664)
```

```
67 th percentile :  ('opened', 83.96846207049516)

67 th percentile :  ('opened', 83.96846207049516)
68 th percentile :  ('cheaper', 86.85848087514947)

68 th percentile :  ('cheaper', 86.85848087514947)
69 th percentile :  ('serving', 90.12445915407207)

70 th percentile :  ('apple', 91.66743791896737)
71 th percentile :  ('varieties', 95.16785731890529)

71 th percentile :  ('varieties', 95.16785731890529)
72 th percentile :  ('hooked', 98.91544871834057)

79 th percentile :  ('easy make', 102.28756771874497)
80 th percentile :  ('house', 113.70426985541637)

80 th percentile :  ('house', 113.70426985541637)
81 th percentile :  ('mean', 124.1882682730498)

81 th percentile :  ('mean', 124.1882682730498)
82 th percentile :  ('room', 142.97938447766813)

82 th percentile :  ('room', 142.97938447766813)
83 th percentile :  ('soft', 165.07956862597618)

83 th percentile :  ('soft', 165.07956862597618)
84 th percentile :  ('cocoa', 217.64991389179227)

84 th percentile :  ('cocoa', 217.64991389179227)
85 th percentile :  ('dogs love', 306.09064033396805)

85 th percentile :  ('dogs love', 306.09064033396805)
86 th percentile :  ('ok', 368.8947319406195)

86 th percentile :  ('ok', 368.8947319406195)
87 th percentile :  ('children', 425.15531719705876)

87 th percentile :  ('children', 425.15531719705876)
88 th percentile :  ('excellent', 985.4557058363861)

88 th percentile :  ('excellent', 985.4557058363861)
89 th percentile :  ('due', 35268.29847831288)

89 th percentile :  ('due', 35268.29847831288)
90 th percentile :  ('food', 299628.3570150673)

90 th percentile :  ('food', 299628.3570150673)
```

```
91 th percentile :  ('best', 511176.085627835)

91 th percentile :  ('best', 511176.085627835)
92 th percentile :  ('daughter', 746075.4445921665)

92 th percentile :  ('daughter', 746075.4445921665)
93 th percentile :  ('always', 987063.9099715654)

93 th percentile :  ('always', 987063.9099715654)
94 th percentile :  ('though', 1521587.0195252018)

94 th percentile :  ('though', 1521587.0195252018)
95 th percentile :  ('crazy', 2276999.184031091)

95 th percentile :  ('crazy', 2276999.184031091)
96 th percentile :  ('flavors', 2903741.576993312)

96 th percentile :  ('flavors', 2903741.576993312)
97 th percentile :  ('thin', 4680354.412447514)

97 th percentile :  ('thin', 4680354.412447514)
98 th percentile :  ('bags', 7390505.314619637)

98 th percentile :  ('bags', 7390505.314619637)
99 th percentile :  ('eggs', 23824140.225007847)
```

These are some of the features showing multicolinearity and hence we cant give feature importance or interpretability.

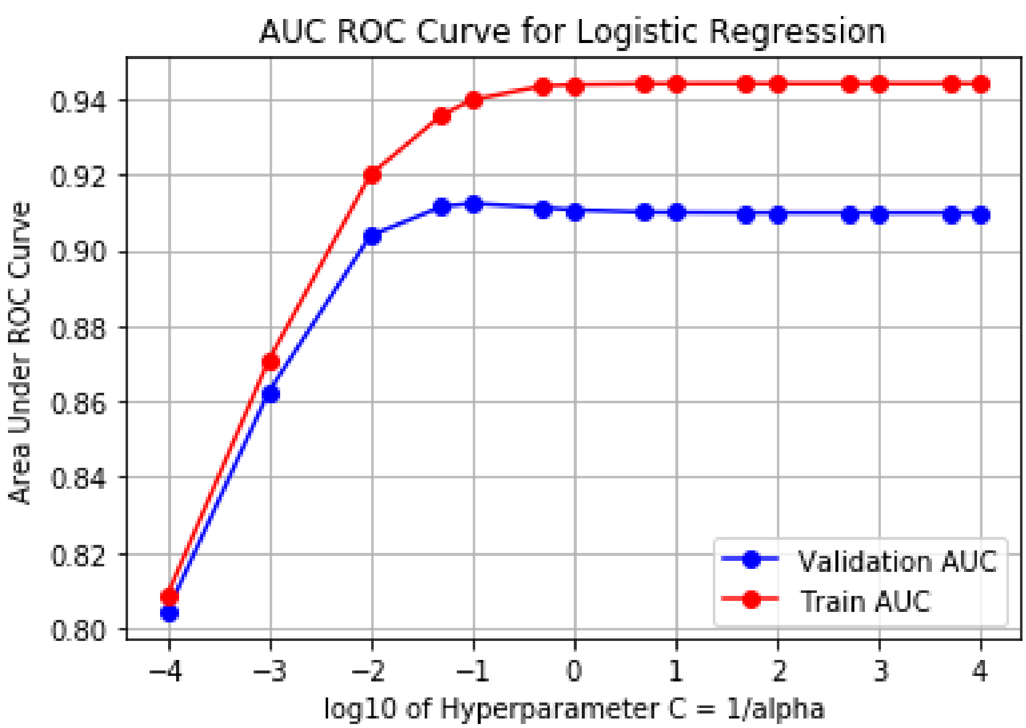
7.1.3 [5.1.2] Applying Logistic Regression with L2 regularization on BOW, SET 1

In [52]:
```
# Please write all the code with proper documentation
csv_path = 'saved_models/Assignment5/BOW_log_reg_results_l2.csv'
cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='BOW', v
                                           penalty=['l2'], results_path=csv_path, ret
# Analysing best parameters
best_parameters = analyse_results(cv_results)

# retraining the model with best parameters
model_path = 'saved_models/Assignment5/{0}_log_reg_l2.pkl'.format('BOW')
log_reg = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'BOW', model_p

print('Retraining Vectorizer with Dx_train')
vectorizer_obj = get_vectorizer(W2V_model=None, train=Dx_train, vectorizer='BOW')

# plotting AUC ROC
train_score, test_score = plot_AUC_ROC(log_reg, vectorizer_obj, Dx_train, Dx_test, Dy_
```

```
    # appending the data results
    prettytable_data.append(['BOW', 'LogisticRegression', 'L2', best_parameters['logistic_
```

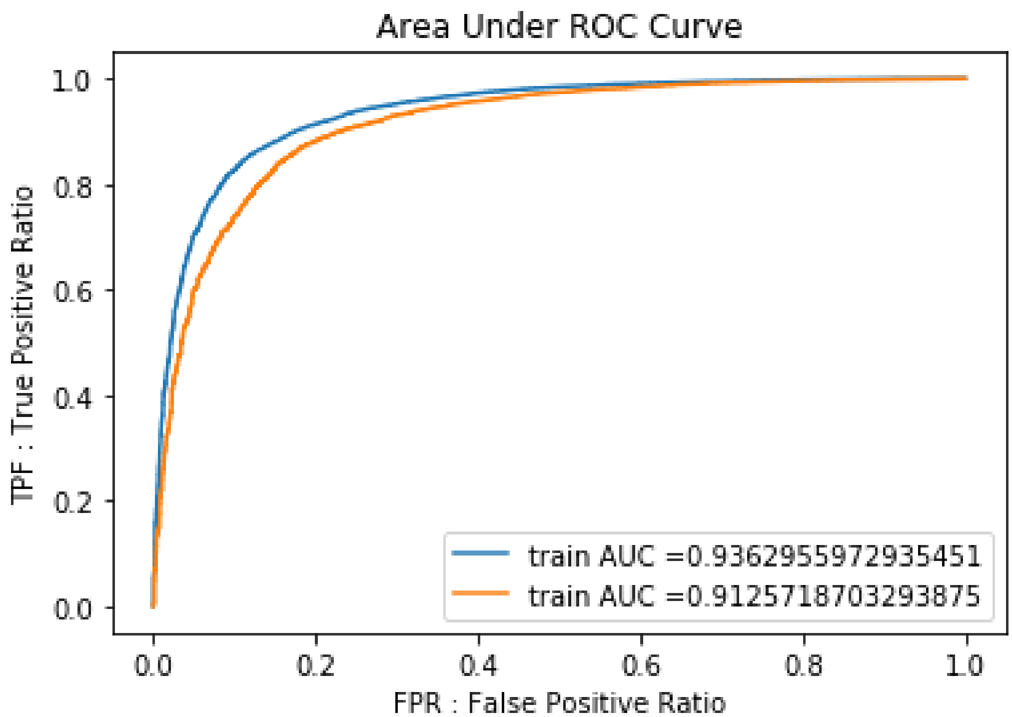
```
Performing Hyperparameter Tuning...

CV iteration : C=0.0001, solver=saga, train_score=0.8159612740447112, test_score=0.81061690059
CV iteration : C=0.0001, solver=saga, train_score=0.7929317408214105, test_score=0.78313024540
CV iteration : C=0.0001, solver=saga, train_score=0.8173374823748631, test_score=0.81930219546
C=0.0001, penalty=l2, solver="saga", train_score=0.8087434990803283, test_score=0.804349780490
CV iteration : C=0.001, solver=saga, train_score=0.8712384409375105, test_score=0.860243025533
CV iteration : C=0.001, solver=saga, train_score=0.8698574061095923, test_score=0.858890929726
CV iteration : C=0.001, solver=saga, train_score=0.8717808662792136, test_score=0.868504117312
C=0.001, penalty=l2, solver="saga", train_score=0.8709589044421054, test_score=0.862546024191
CV iteration : C=0.01, solver=saga, train_score=0.9184452432602117, test_score=0.901969654107
CV iteration : C=0.01, solver=saga, train_score=0.9219529198934143, test_score=0.904486559407
CV iteration : C=0.01, solver=saga, train_score=0.9194468596921284, test_score=0.904919928374
C=0.01, penalty=l2, solver="saga", train_score=0.9199483409485848, test_score=0.903792047296
CV iteration : C=0.05, solver=saga, train_score=0.9332984388798955, test_score=0.911337604125
CV iteration : C=0.05, solver=saga, train_score=0.9385647314827107, test_score=0.912887068639
CV iteration : C=0.05, solver=saga, train_score=0.935156004809536, test_score=0.910200350245
C=0.05, penalty=l2, solver="saga", train_score=0.935673058390714, test_score=0.9114750076702455
CV iteration : C=0.1, solver=saga, train_score=0.9370112468296754, test_score=0.912798730462
CV iteration : C=0.1, solver=saga, train_score=0.9420622519394194, test_score=0.912720767771
CV iteration : C=0.1, solver=saga, train_score=0.9402620774466128, test_score=0.911532708224
C=0.1, penalty=l2, solver="saga", train_score=0.9397785254052359, test_score=0.912350735486383
CV iteration : C=0.5, solver=saga, train_score=0.9402942272175564, test_score=0.912590106991
CV iteration : C=0.5, solver=saga, train_score=0.9468901142738392, test_score=0.911219839429
CV iteration : C=0.5, solver=saga, train_score=0.9430591941401132, test_score=0.909814249588
C=0.5, penalty=l2, solver="saga", train_score=0.9434145118771696, test_score=0.911208065336560
CV iteration : C=1, solver=saga, train_score=0.9405709940185543, test_score=0.9123163582958926
CV iteration : C=1, solver=saga, train_score=0.9473473620467349, test_score=0.9101417854112697
CV iteration : C=1, solver=saga, train_score=0.9433365918416323, test_score=0.9094168344107726
C=1, penalty=l2, solver="saga", train_score=0.9437516493023071, test_score=0.9106249927059783
CV iteration : C=5, solver=saga, train_score=0.9407608544319473, test_score=0.9120502886319494
CV iteration : C=5, solver=saga, train_score=0.947633200526304, test_score=0.9089810205984463
CV iteration : C=5, solver=saga, train_score=0.943541214595413, test_score=0.9090575251287862
C=5, penalty=l2, solver="saga", train_score=0.9439784231845548, test_score=0.9100296114530607
CV iteration : C=10, solver=saga, train_score=0.9407854901212498, test_score=0.912016465720273
CV iteration : C=10, solver=saga, train_score=0.947660518149659, test_score=0.908812785318535
CV iteration : C=10, solver=saga, train_score=0.943565292170752, test_score=0.9090156086437771
C=10, penalty=l2, solver="saga", train_score=0.944003766813887, test_score=0.909948286560862
CV iteration : C=50, solver=saga, train_score=0.9408044688349003, test_score=0.911985104788304
CV iteration : C=50, solver=saga, train_score=0.9476796287617055, test_score=0.908674211032224
CV iteration : C=50, solver=saga, train_score=0.943583024286492, test_score=0.9089759198880412
C=50, penalty=l2, solver="saga", train_score=0.9440223739610326, test_score=0.9098784119028567
CV iteration : C=100, solver=saga, train_score=0.9408070921474436, test_score=0.911981763530131
CV iteration : C=100, solver=saga, train_score=0.9476793796202909, test_score=0.908655628947300
```

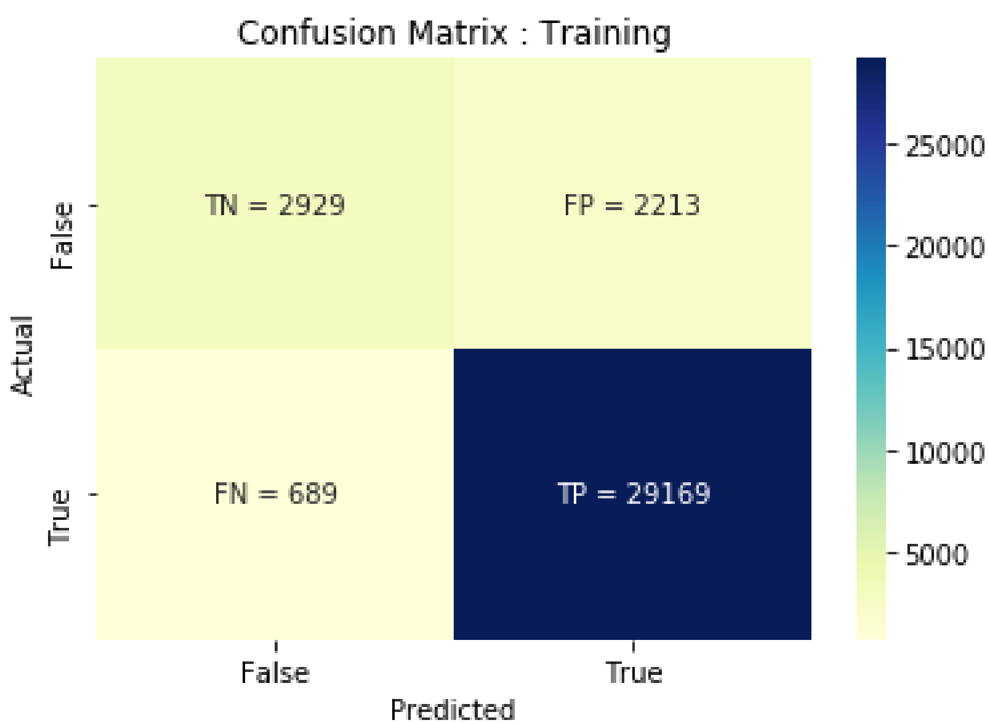
```
CV iteration : C=100, solver=saga, train_score=0.943586585364281, test_score=0.90896935394913O8
C=100, penalty=l2, solver="saga", train_score=0.9440243523773385, test_score=0.909868915475521
CV iteration : C=500, solver=saga, train_score=0.9408093930416855, test_score=0.91198246695290
CV iteration : C=500, solver=saga, train_score=0.9476820029328341, test_score=0.90863997779059
CV iteration : C=500, solver=saga, train_score=0.9435885637408306, test_score=0.90896261213685
C=500, penalty=l2, solver="saga", train_score=0.94402665323845, test_score=0.9098616856267867
CV iteration : C=1000, solver=saga, train_score=0.9408086749281961, test_score=0.91198281866429
CV iteration : C=1000, solver=saga, train_score=0.947682296040381, test_score=0.90864091568762
CV iteration : C=1000, solver=saga, train_score=0.9435890619986281, test_score=0.90896466399276
C=1000, penalty=l2, solver="saga", train_score=0.9440266776557351, test_score=0.90986279944822
CV iteration : C=5000, solver=saga, train_score=0.9408084404421586, test_score=0.91197965326181
CV iteration : C=5000, solver=saga, train_score=0.9476819296559474, test_score=0.90863558139826
CV iteration : C=5000, solver=saga, train_score=0.9435888421790115, test_score=0.90896032578312
C=5000, penalty=l2, solver="saga", train_score=0.9440264040923725, test_score=0.90985852014773
CV iteration : C=10000, solver=saga, train_score=0.9408089094142333, test_score=0.91197988773606
CV iteration : C=10000, solver=saga, train_score=0.9476817098252874, test_score=0.90863903989393
CV iteration : C=10000, solver=saga, train_score=0.9435891352718337, test_score=0.90896231901445
C=10000, penalty=l2, solver="saga", train_score=0.9440265848371182, test_score=0.90986041554807
```

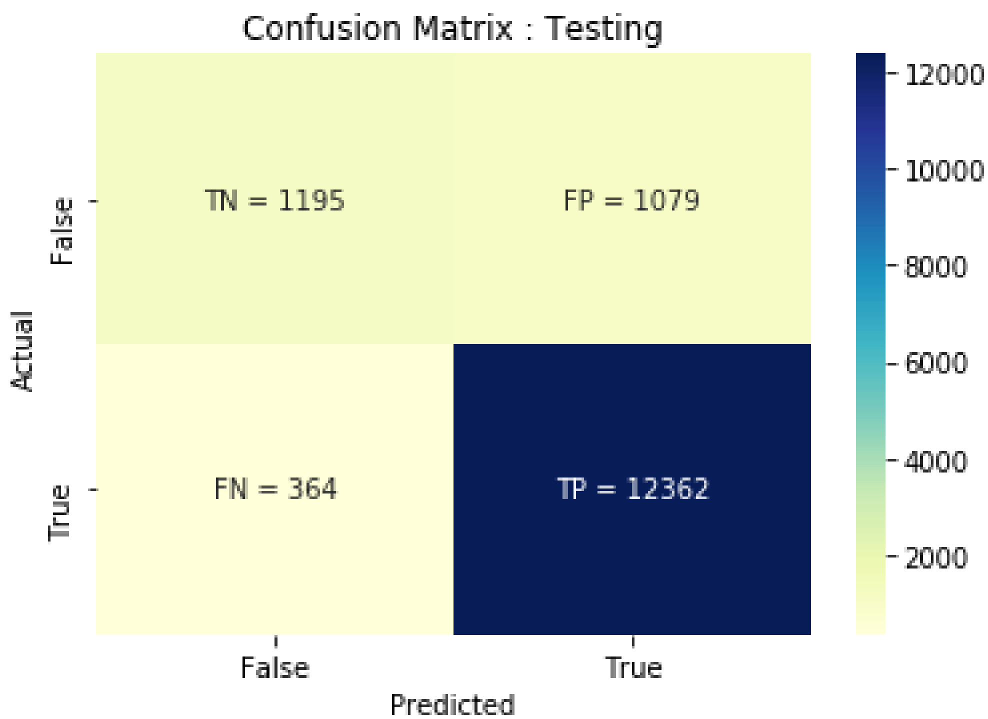



```
Initializing Vectorizer
Training Model...
Saving Trained Model...
Retraining Vectorizer with Dx_train
```

```
Area Under the Curve for Train :  0.9362955972935451
Area Under the Curve for Test :  0.9125718703293875
```







7.2 [5.2] Logistic Regression on TFIDF, SET 2

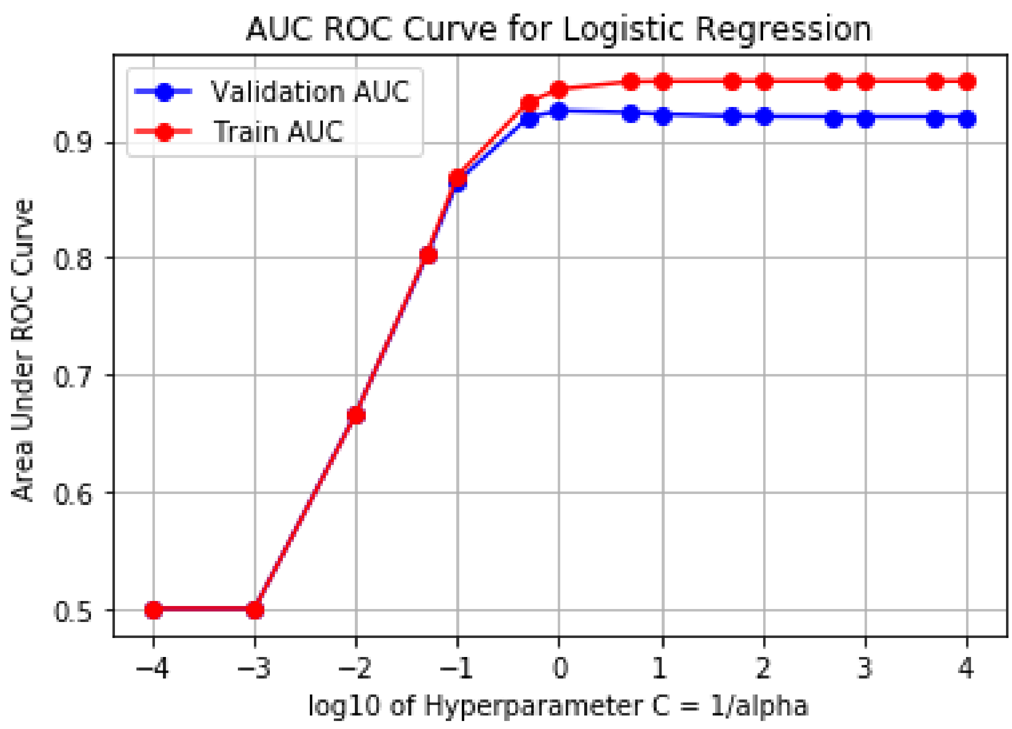
7.2.1 [5.2.1] Applying Logistic Regression with L1 regularization on TFIDF, SET 2

In [53]:
```
# Please write all the code with proper documentation
csv_path = 'saved_models/Assignment5/TFIDF_log_reg_results_l1.csv'
cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='TFIDF'
                                           penalty=['l1'], results_path=csv_path, ret
# Analysing best parameters
best_parameters = analyse_results(cv_results)

# retraining the model with best parameters
model_path = 'saved_models/Assignment5/{0}_log_reg_l1.pkl'.format('TFIDF')
log_reg = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'TFIDF', model

print('Retraining Vectorizer with Dx_train')
vectorizer_obj = get_vectorizer(W2V_model=None, train=Dx_train, vectorizer='TFIDF')

# plotting AUC ROC
train_score, test_score = plot_AUC_ROC(log_reg, vectorizer_obj, Dx_train, Dx_test, Dy_
```

```
        # appending the data results
        prettytable_data.append(['TFIDF', 'LogisticRegression', 'L1', best_parameters['logist
```

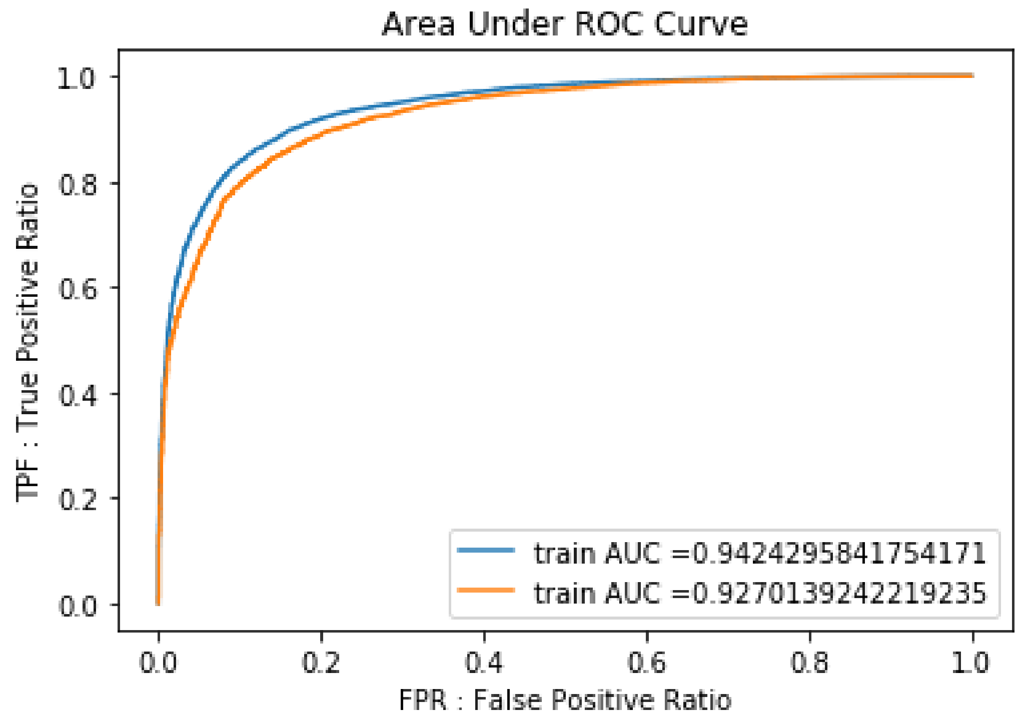
```
Performing Hyperparameter Tuning...

CV iteration : C=0.0001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.0001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.0001, solver=saga, train_score=0.5, test_score=0.5
C=0.0001, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=0.001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.001, solver=saga, train_score=0.5, test_score=0.5
C=0.001, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=0.01, solver=saga, train_score=0.6665617195095608, test_score=0.6681858644614
CV iteration : C=0.01, solver=saga, train_score=0.6685603685768778, test_score=0.6626467618342
CV iteration : C=0.01, solver=saga, train_score=0.6652055427721493, test_score=0.6706821095986
C=0.01, penalty=l1, solver="saga", train_score=0.6667758769528627, test_score=0.6671715786314
CV iteration : C=0.05, solver=saga, train_score=0.8041778010895981, test_score=0.8017712419902
CV iteration : C=0.05, solver=saga, train_score=0.8037825822012787, test_score=0.7927278922722
CV iteration : C=0.05, solver=saga, train_score=0.8036081719437248, test_score=0.8119039651705
C=0.05, penalty=l1, solver="saga", train_score=0.8038561850782006, test_score=0.8021343664776
CV iteration : C=0.1, solver=saga, train_score=0.8679350602643772, test_score=0.8642287948222
CV iteration : C=0.1, solver=saga, train_score=0.869509099084127, test_score=0.86148573909979
CV iteration : C=0.1, solver=saga, train_score=0.873027924946197, test_score=0.8696334588052
C=0.1, penalty=l1, solver="saga", train_score=0.8701573614315671, test_score=0.8651159975757
CV iteration : C=0.5, solver=saga, train_score=0.93207651015603, test_score=0.9207017732467453
CV iteration : C=0.5, solver=saga, train_score=0.9340002555897808, test_score=0.9200461539128
CV iteration : C=0.5, solver=saga, train_score=0.9337349955526095, test_score=0.9185998569094
C=0.5, penalty=l1, solver="saga", train_score=0.9332705870994734, test_score=0.9197825946896
CV iteration : C=1, solver=saga, train_score=0.9422949280377007, test_score=0.9272606630392717
CV iteration : C=1, solver=saga, train_score=0.9450826812423188, test_score=0.9251741352384211
CV iteration : C=1, solver=saga, train_score=0.9442919059134525, test_score=0.9246334271480938
C=1, penalty=l1, solver="saga", train_score=0.9438898383978239, test_score=0.9256894084752622
CV iteration : C=5, solver=saga, train_score=0.948529721251792, test_score=0.926619053542314
CV iteration : C=5, solver=saga, train_score=0.9518872608718718, test_score=0.9226894701479684
CV iteration : C=5, solver=saga, train_score=0.9507355003170677, test_score=0.9230398679120688
C=5, penalty=l1, solver="saga", train_score=0.9503841608135772, test_score=0.9241161305341171
CV iteration : C=10, solver=saga, train_score=0.9488839417220125, test_score=0.925450961408937
CV iteration : C=10, solver=saga, train_score=0.9522993554271939, test_score=0.9211303335712
CV iteration : C=10, solver=saga, train_score=0.9510467648941859, test_score=0.9216273703039
C=10, penalty=l1, solver="saga", train_score=0.9507433540144641, test_score=0.9227362217613897
CV iteration : C=50, solver=saga, train_score=0.9489942527472237, test_score=0.9242263609794
CV iteration : C=50, solver=saga, train_score=0.9524707060990112, test_score=0.9189472316855
CV iteration : C=50, solver=saga, train_score=0.9511242146724377, test_score=0.9201934161454
C=50, penalty=l1, solver="saga", train_score=0.9508630578395575, test_score=0.921122336270148
CV iteration : C=100, solver=saga, train_score=0.9489952786236373, test_score=0.9240633427517
CV iteration : C=100, solver=saga, train_score=0.9524842037015381, test_score=0.9184874862847
CV iteration : C=100, solver=saga, train_score=0.9511243612188487, test_score=0.9200013624323
```

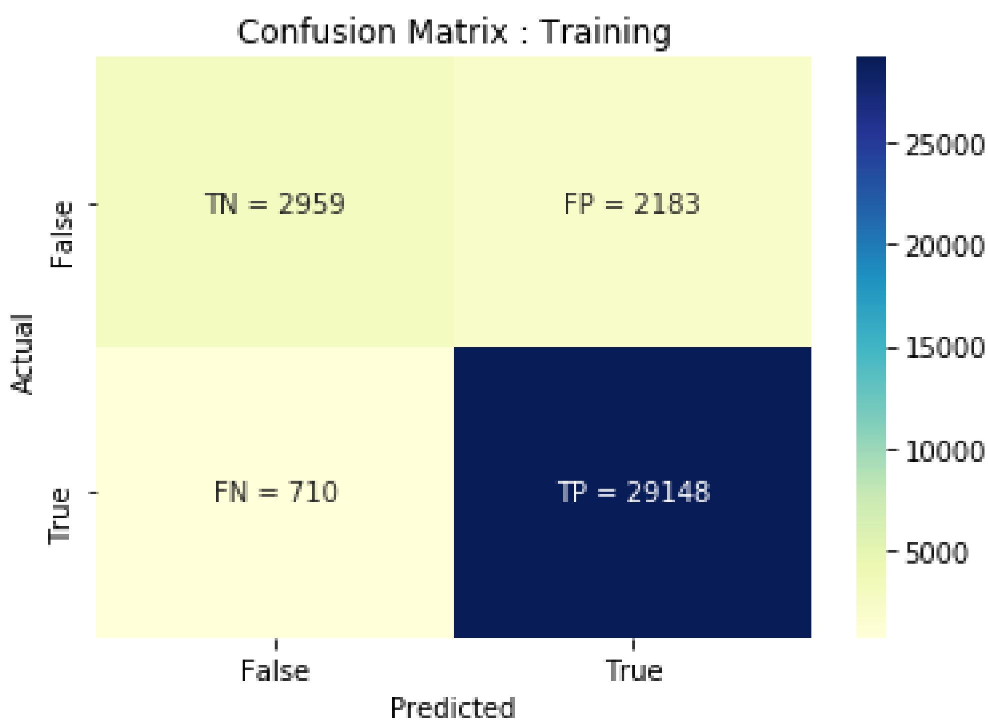
```
C=100, penalty=l1, solver="saga", train_score=0.9508679478480081, test_score=0.920850730489606
CV iteration : C=500, solver=saga, train_score=0.9489961872570323, test_score=0.92393086479616
CV iteration : C=500, solver=saga, train_score=0.9524897580895484, test_score=0.91808981794363
CV iteration : C=500, solver=saga, train_score=0.9511237896878456, test_score=0.91985521166711
C=500, penalty=l1, solver="saga", train_score=0.9508699116781422, test_score=0.920625298135636
CV iteration : C=1000, solver=saga, train_score=0.9489954544881652, test_score=0.92391339646639
CV iteration : C=1000, solver=saga, train_score=0.9524900218863405, test_score=0.91803975768961
CV iteration : C=1000, solver=saga, train_score=0.9511221776773238, test_score=0.91983727258401
C=1000, penalty=l1, solver="saga", train_score=0.9508692180172765, test_score=0.92059680891253
CV iteration : C=5000, solver=saga, train_score=0.9489940182611863, test_score=0.92389932800850
CV iteration : C=5000, solver=saga, train_score=0.9524907839659621, test_score=0.91799708337470
CV iteration : C=5000, solver=saga, train_score=0.9511221630226827, test_score=0.91982214747473
C=5000, penalty=l1, solver="saga", train_score=0.9508689884166103, test_score=0.92057285295264
CV iteration : C=10000, solver=saga, train_score=0.9489939449842996, test_score=0.92389756945l5
CV iteration : C=10000, solver=saga, train_score=0.952490476203038, test_score=0.91799086980687
CV iteration : C=10000, solver=saga, train_score=0.9511216501102439, test_score=0.91982021286
C=10000, penalty=l1, solver="saga", train_score=0.9508686904325271, test_score=0.92056955070872
```

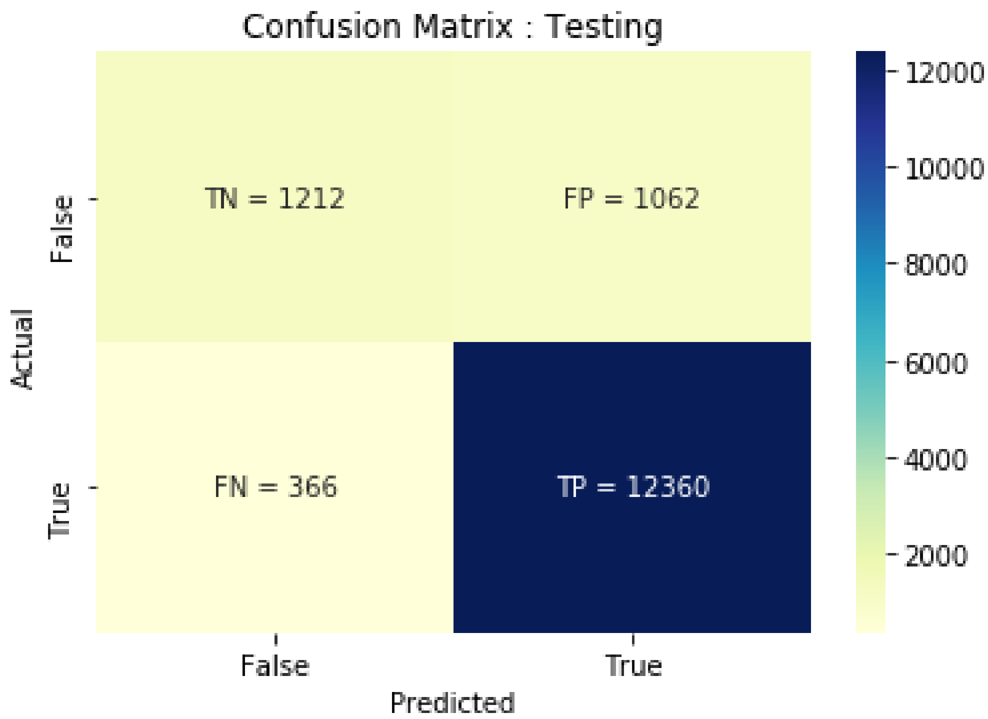



```
Initializing Vectorizer
Training Model...
Saving Trained Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.9424295841754171
```

```
Area Under the Curve for Test :  0.9270139242219235
```





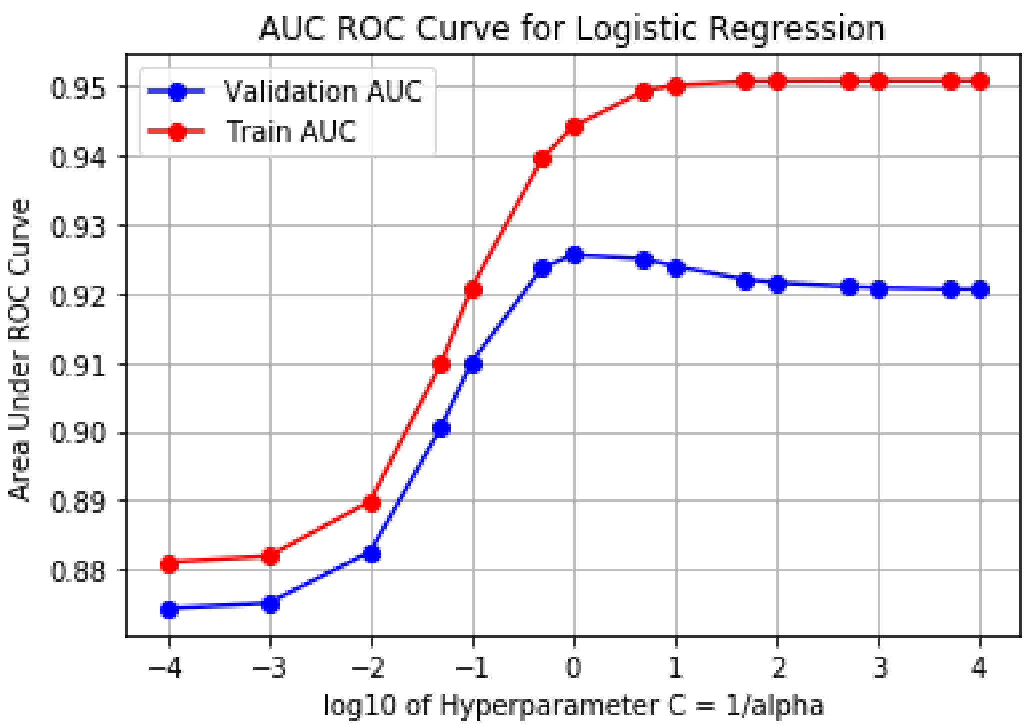


7.2.2 [5.2.2] Applying Logistic Regression with L2 regularization on TFIDF, SET 2

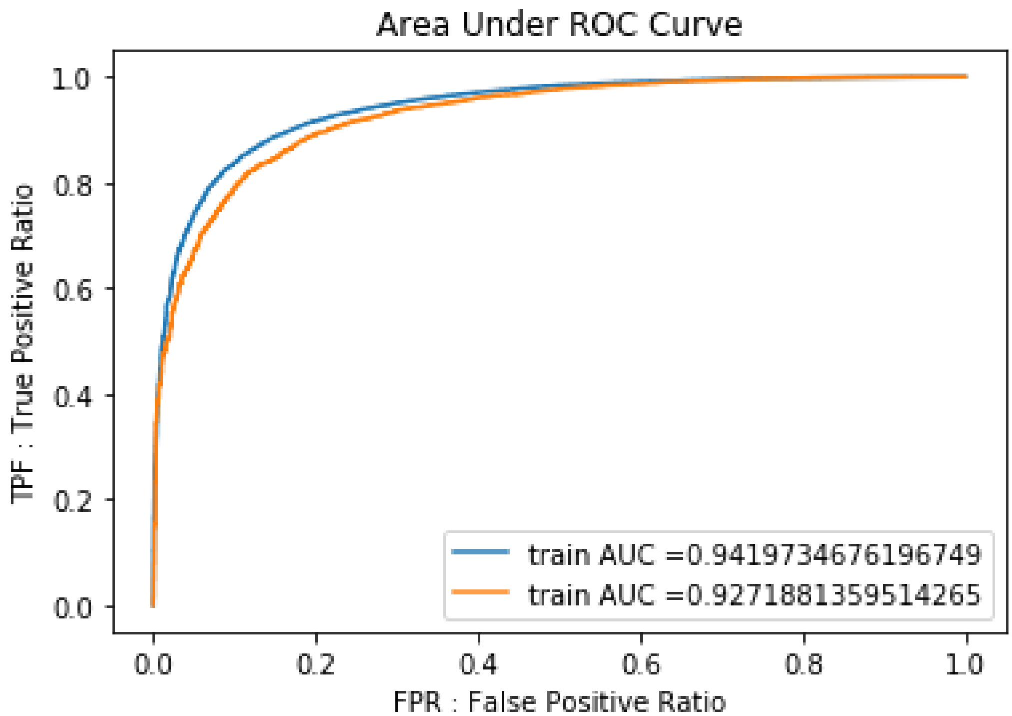
In [54]:
```
# Please write all the code with proper documentation
csv_path = 'saved_models/Assignment5/TFIDF_log_reg_results_l2.csv'
cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='TFIDF'
                                           penalty=['l2'], results_path=csv_path, ret
# Analysing best parameters
best_parameters = analyse_results(cv_results)

# retraining the model with best parameters
model_path = 'saved_models/Assignment5/{0}_log_reg_l2.pkl'.format('TFIDF')
log_reg = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'TFIDF', model

print('Retraining Vectorizer with Dx_train')
vectorizer_obj = get_vectorizer(W2V_model=None, train=Dx_train, vectorizer='TFIDF')

# plotting AUC ROC
train_score, test_score = plot_AUC_ROC(log_reg, vectorizer_obj, Dx_train, Dx_test, Dy_

# appending the data results
prettytable_data.append(['TFIDF', 'LogisticRegression', 'L2', best_parameters['logist
```

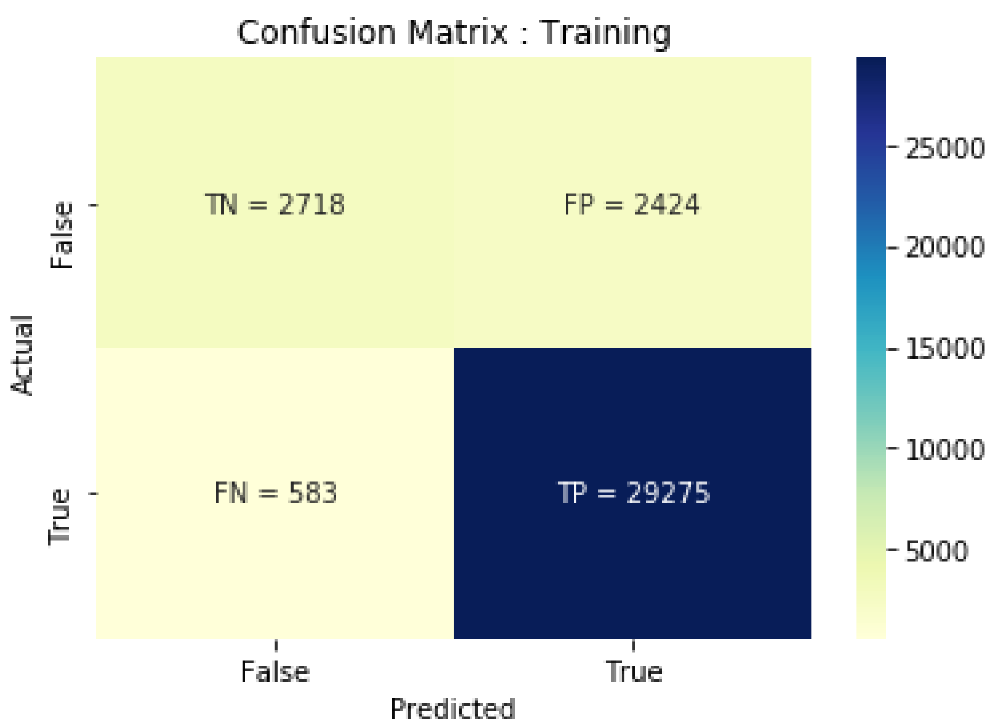
```
Performing Hyperparameter Tuning...

CV iteration : C=0.0001, solver=saga, train_score=0.8798079749873743, test_score=0.87284214806
CV iteration : C=0.0001, solver=saga, train_score=0.8810617498461918, test_score=0.87175436336
CV iteration : C=0.0001, solver=saga, train_score=0.8821356891391876, test_score=0.87826719947
C=0.0001, penalty=l2, solver="saga", train_score=0.8810018046575845, test_score=0.87428790363
CV iteration : C=0.001, solver=saga, train_score=0.8806614308865595, test_score=0.87364199837
CV iteration : C=0.001, solver=saga, train_score=0.8819202618505578, test_score=0.87257396812
CV iteration : C=0.001, solver=saga, train_score=0.8830282300558249, test_score=0.87906038834
C=0.001, penalty=l2, solver="saga", train_score=0.8818699742643141, test_score=0.87509211828
CV iteration : C=0.01, solver=saga, train_score=0.8885272796659278, test_score=0.88121117912
CV iteration : C=0.01, solver=saga, train_score=0.8899809392162362, test_score=0.88026929602
CV iteration : C=0.01, solver=saga, train_score=0.8910345001319504, test_score=0.88606888326
C=0.01, penalty=l2, solver="saga", train_score=0.8898475730047047, test_score=0.88251645280
CV iteration : C=0.05, solver=saga, train_score=0.9084322278195993, test_score=0.89998195720
CV iteration : C=0.05, solver=saga, train_score=0.9101220148095519, test_score=0.89901088206
CV iteration : C=0.05, solver=saga, train_score=0.9108941766676777, test_score=0.90263292977
C=0.05, penalty=l2, solver="saga", train_score=0.9098161397656096, test_score=0.90054192301
CV iteration : C=0.1, solver=saga, train_score=0.9194533500287392, test_score=0.91003955463
CV iteration : C=0.1, solver=saga, train_score=0.9213040237510908, test_score=0.90891730221
CV iteration : C=0.1, solver=saga, train_score=0.9217368964940353, test_score=0.91107678584
C=0.1, penalty=l2, solver="saga", train_score=0.9208314234246218, test_score=0.91001121423
CV iteration : C=0.5, solver=saga, train_score=0.9381807972349406, test_score=0.92490909139
CV iteration : C=0.5, solver=saga, train_score=0.9406171438017866, test_score=0.92294114895
CV iteration : C=0.5, solver=saga, train_score=0.940288338563477, test_score=0.92339255263
C=0.5, penalty=l2, solver="saga", train_score=0.9396954265334014, test_score=0.92374759766
CV iteration : C=1, solver=saga, train_score=0.9426562417105522, test_score=0.9271182492369914
CV iteration : C=1, solver=saga, train_score=0.9453269424163844, test_score=0.9247991229724865
CV iteration : C=1, solver=saga, train_score=0.944764261632942, test_score=0.9249878999125791
C=1, penalty=l2, solver="saga", train_score=0.9442491485866262, test_score=0.9256350907073522
CV iteration : C=5, solver=saga, train_score=0.9476390846603046, test_score=0.9269605652986774
CV iteration : C=5, solver=saga, train_score=0.950794072310218, test_score=0.9240444089554629
CV iteration : C=5, solver=saga, train_score=0.9498137527007039, test_score=0.9241161542733007
C=5, penalty=l2, solver="saga", train_score=0.9494156365570755, test_score=0.9250403761758137
CV iteration : C=10, solver=saga, train_score=0.9483883847927599, test_score=0.92607659734708
CV iteration : C=10, solver=saga, train_score=0.9516691448909742, test_score=0.92292227377659
CV iteration : C=10, solver=saga, train_score=0.9505813921111839, test_score=0.92296940131768
C=10, penalty=l2, solver="saga", train_score=0.9502129739316393, test_score=0.9239894241471255
CV iteration : C=50, solver=saga, train_score=0.9489332863775044, test_score=0.92456658312739
CV iteration : C=50, solver=saga, train_score=0.9523606588705922, test_score=0.92066381772627
CV iteration : C=50, solver=saga, train_score=0.9510810714090181, test_score=0.92089608885778
C=50, penalty=l2, solver="saga", train_score=0.9507916722190383, test_score=0.9220421632371544
CV iteration : C=100, solver=saga, train_score=0.9489697489563174, test_score=0.92425021873517
CV iteration : C=100, solver=saga, train_score=0.9524236623377612, test_score=0.92005013997527
CV iteration : C=100, solver=saga, train_score=0.9511069661598545, test_score=0.92042926232614
C=100, penalty=l2, solver="saga", train_score=0.950833459151311, test_score=0.9215765403455288
CV iteration : C=500, solver=saga, train_score=0.9489920251298686, test_score=0.92397154607987
CV iteration : C=500, solver=saga, train_score=0.9524771544650392, test_score=0.91908082339387
```

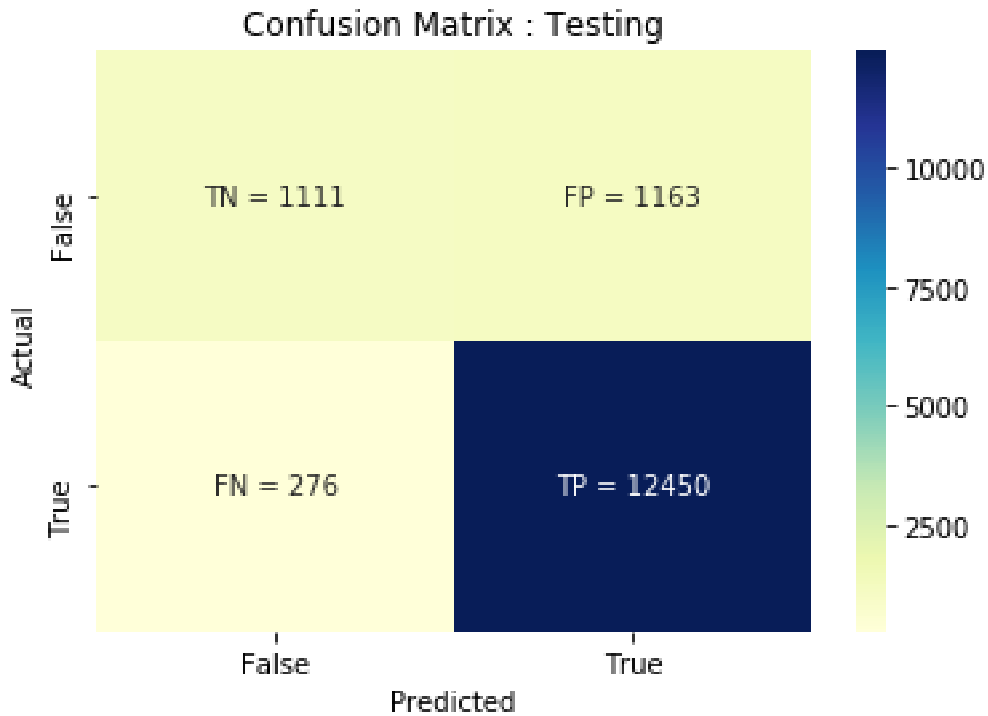
```
CV iteration : C=500, solver=saga, train_score=0.9511223535330171, test_score=0.91996618775958
C=500, penalty=l2, solver="saga", train_score=0.950863844375975, test_score=0.9210061857444461
CV iteration : C=1000, solver=saga, train_score=0.9489951027591093, test_score=0.92393637494122
CV iteration : C=1000, solver=saga, train_score=0.952486387352761, test_score=0.91871656763451
CV iteration : C=1000, solver=saga, train_score=0.9511232767754069, test_score=0.91989736264993
C=1000, penalty=l2, solver="saga", train_score=0.9508682556290924, test_score=0.920850101741889
CV iteration : C=5000, solver=saga, train_score=0.9489947510300534, test_score=0.92390542433920
CV iteration : C=5000, solver=saga, train_score=0.9524904029261512, test_score=0.91816877714992
CV iteration : C=5000, solver=saga, train_score=0.9511224854247872, test_score=0.91983680358837
C=5000, penalty=l2, solver="saga", train_score=0.9508692131269972, test_score=0.920637001692500
CV iteration : C=10000, solver=saga, train_score=0.9489939449842997, test_score=0.923899562482
CV iteration : C=10000, solver=saga, train_score=0.9524901391293592, test_score=0.9180761011995
CV iteration : C=10000, solver=saga, train_score=0.9511222949144527, test_score=0.9198268374311
C=10000, penalty=l2, solver="saga", train_score=0.9508687930093705, test_score=0.920600833704
```




```
Initializing Vectorizer
Training Model...
Saving Trained Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.9419734676196749
Area Under the Curve for Test :  0.9271881359514265
```


Area Under ROC Curve
1.0
0.8
0.6
0.4
0.2
0.0
TPF : True Positive Ratio
train AUC =0.9419734676196749
train AUC =0.9271881359514265
0.0
0.2
0.4
0.6
0.8
1.0
FPR : False Positive Ratio


Confusion Matrix : Training
TN = 2718
FP = 2424
FN = 583
TP = 29275
False
True
Actual
False
True
Predicted
25000
20000
15000
10000
5000



7.2.3 [5.2.3] Feature Importance on TFIDF, SET 2

```
In [55]: # Please write all the code with proper documentation
         feature_names = vectorizer_obj.get_feature_names()
         weights = np.reshape(log_reg.coef_,(log_reg.coef_.shape[1], log_reg.coef_.shape[0]))

         # making a list of feature names along with their feature weights
         features_with_weights = [(feature_names[i],weights[i]) for i in range(len(feature_name
         features_with_weights.sort(key=lambda x : abs(x[1][0]), reverse=True)
```

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

```
In [56]: positive_weights = [i for i in features_with_weights if i[1][0]>=0]
         print('Top 10 features of positive class with the feature names : ')
         for i in positive_weights[:10]:
             print(i[0],' : ',i[1][0])
```

```
Top 10 features of positive class with the feature names :
great  :  8.250425983875289
best  :  7.047466209780987
delicious  :  6.5610791490084805
excellent  :  5.676299237656143
loves  :  5.533156858346458
love  :  5.46371508484913
```

```
perfect  :  5.178975977470662
good  :  4.664923465749984
wonderful  :  4.549403455960735
pleased  :  4.0260402569712666
```

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

```
In [57]: # Please write all the code with proper documentation
         negative_weights = [i for i in features_with_weights if i[1][0]<0]
         print('Top 10 features of negative class with the feature names : ')
         for i in negative_weights[:10]:
             print(i[0],' : ',i[1][0])
```

```
Top 10 features of negative class with the feature names :
not  :  -5.3249053217795
disappointed  :  -5.027833945426516
awful  :  -4.979721775746406
terrible  :  -4.863041733252396
money  :  -4.474683323856104
unfortunately  :  -4.223315676440612
not buy  :  -4.206004932916333
not good  :  -4.107539907827749
horrible  :  -4.097944215892398
bland  :  -3.255097383219475
```

7.3 Preparing/Training Google Word2Vec

```
In [58]: is_your_ram_gt_16g=True
         want_to_use_google_w2v = False
         want_to_train_w2v = True

         path_to_word2vec = '/home/monodeepdas112/Datasets/GoogleNews-vectors-negative300.bin.

         if want_to_train_w2v:

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

             # min_count = 5 considers only words that occured atleast 5 times
             w2v_model=Word2Vec(list_of_sentences,min_count=5,size=300, 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(path_to_word2vec):
            print('Preparing to load pre-trained Word2Vec model !')
            w2v_model=KeyedVectors.load_word2vec_format(path_to_word2vec, binary=True)
            print('Successfully loaded model into memory !!')
            print('Words similar to "similar" : ', w2v_model.wv.most_similar('great'))
            print('Words similar to "worst" : ',w2v_model.wv.most_similar('worst'))
        else:
            print("you don't have google's word2vec file, keep want_to_train_w2v = True, t
```

```
[('terrific', 0.78861403465271), ('fantastic', 0.7743387222290039), ('excellent', 0.75111579895
==================================================
[('nastiest', 0.7911990284919739), ('greatest', 0.6833856105804443), ('disgusting', 0.63364648
```

7.4 [5.3] Logistic Regression on AVG W2V, SET 3

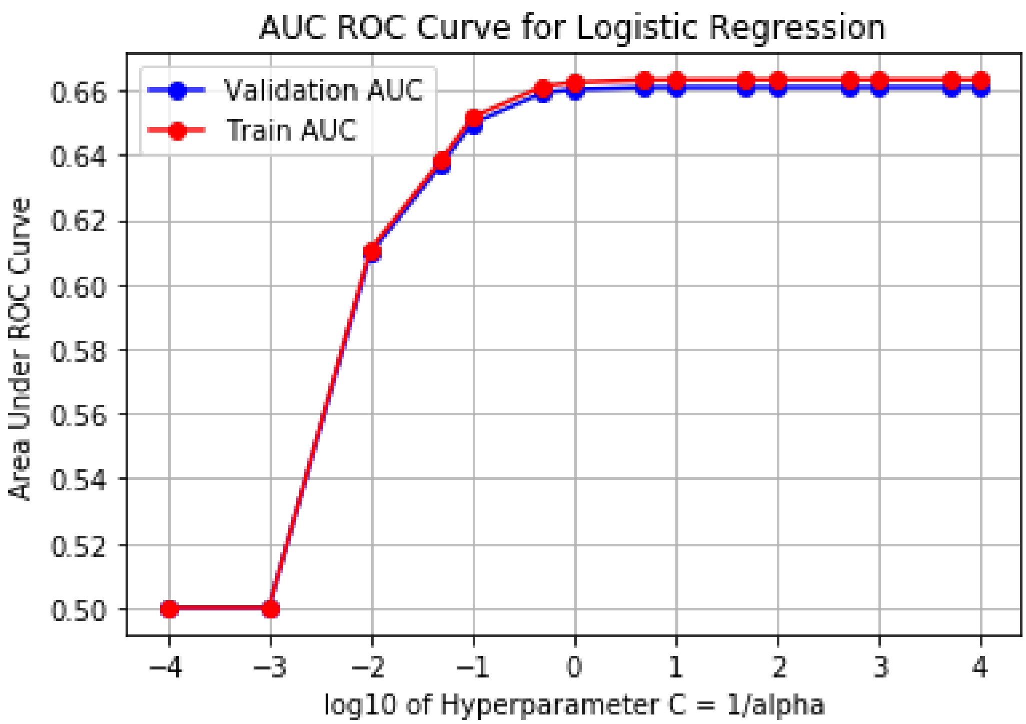
7.4.1 [5.3.1] Applying Logistic Regression with L1 regularization on AVG W2V SET 3

In [59]:
```
# Please write all the code with proper documentation
csv_path = 'saved_models/Assignment5/Avg-W2Vec_log_reg_results_l1.csv'
cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='Avg-W2V
                                            penalty=['l1'], results_path=csv_path, ret
# Analysing best parameters
best_parameters = analyse_results(cv_results)

# retraining the model with best parameters
model_path = 'saved_models/Assignment5/{0}_log_reg_l1.pkl'.format('Avg-W2Vec')
log_reg = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'Avg-W2Vec', m

print('Retraining Vectorizer with Dx_train')
vectorizer_obj = get_vectorizer(W2V_model=w2v_model, train=Dx_train, vectorizer='Avg-W

# plotting AUC ROC
train_score, test_score = plot_AUC_ROC(log_reg, vectorizer_obj, Dx_train, Dx_test, Dy_

# appending the data results
prettytable_data.append(['Avg Word2Vec', 'LogisticRegression', 'L1', best_parameters[
```

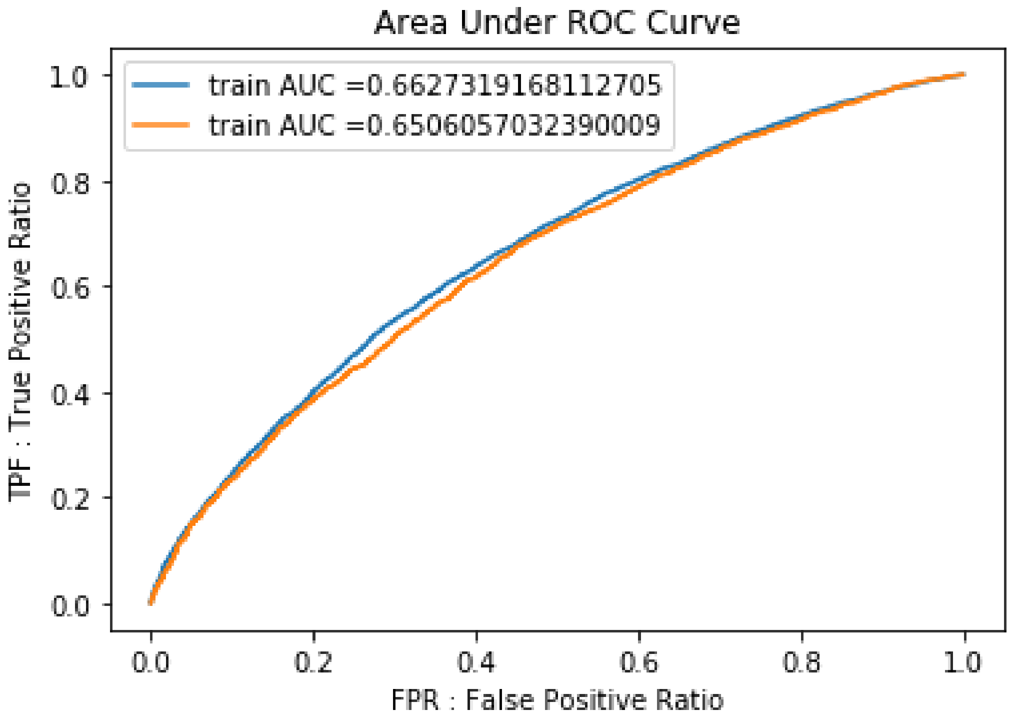
```
Performing Hyperparameter Tuning...

CV iteration : C=0.0001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.0001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.0001, solver=saga, train_score=0.5, test_score=0.5
C=0.0001, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=0.001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.001, solver=saga, train_score=0.5, test_score=0.5
```

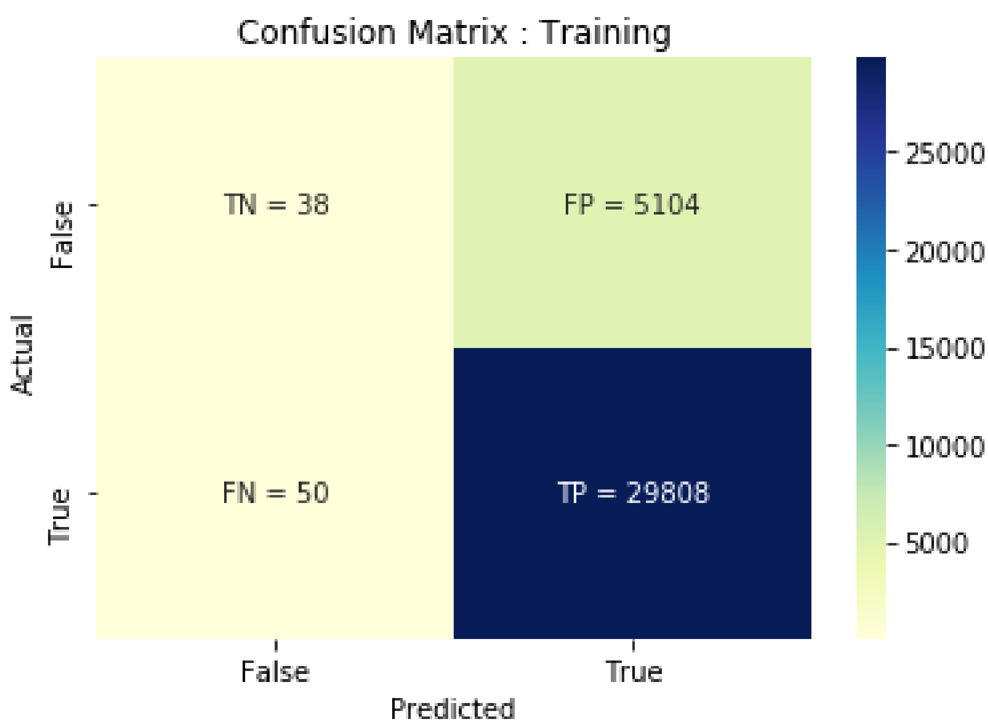
```
C=0.001, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=0.01, solver=saga, train_score=0.6014673916388728, test_score=0.6049558948059
CV iteration : C=0.01, solver=saga, train_score=0.6142678012273586, test_score=0.6107555569519
CV iteration : C=0.01, solver=saga, train_score=0.616261173724205, test_score=0.6125077442904
C=0.01, penalty=l1, solver="saga", train_score=0.6106654555301455, test_score=0.6094063986827
CV iteration : C=0.05, solver=saga, train_score=0.6390860232545665, test_score=0.6371251181603
CV iteration : C=0.05, solver=saga, train_score=0.6383272117822198, test_score=0.6327745069270
CV iteration : C=0.05, solver=saga, train_score=0.6381474845425776, test_score=0.6412530730939
C=0.05, penalty=l1, solver="saga", train_score=0.638520239859788, test_score=0.6370508993937
CV iteration : C=0.1, solver=saga, train_score=0.6533847766388596, test_score=0.6497456950819
CV iteration : C=0.1, solver=saga, train_score=0.6500071225133854, test_score=0.6477035415343
CV iteration : C=0.1, solver=saga, train_score=0.6513110305718088, test_score=0.6511968358271
C=0.1, penalty=l1, solver="saga", train_score=0.6515676432413513, test_score=0.6495486908144
CV iteration : C=0.5, solver=saga, train_score=0.6621460689734817, test_score=0.6583827888391
CV iteration : C=0.5, solver=saga, train_score=0.6614628206266815, test_score=0.6588098250810
CV iteration : C=0.5, solver=saga, train_score=0.6599744001005425, test_score=0.6604277838173
C=0.5, penalty=l1, solver="saga", train_score=0.6611944299002352, test_score=0.6592067992458
CV iteration : C=1, solver=saga, train_score=0.6633257535721749, test_score=0.659254798603612
CV iteration : C=1, solver=saga, train_score=0.6628535573143962, test_score=0.659396479673837
CV iteration : C=1, solver=saga, train_score=0.6606331555275957, test_score=0.6616021488911067
C=1, penalty=l1, solver="saga", train_score=0.6622708221380557, test_score=0.660084475722852
CV iteration : C=5, solver=saga, train_score=0.663963995255175, test_score=0.6596195819300538
CV iteration : C=5, solver=saga, train_score=0.6637815358073369, test_score=0.6594682874152624
CV iteration : C=5, solver=saga, train_score=0.6613421763736469, test_score=0.6631163013034327
C=5, penalty=l1, solver="saga", train_score=0.663029235812053, test_score=0.660734723549583
CV iteration : C=10, solver=saga, train_score=0.6639943905077706, test_score=0.6596101443411
CV iteration : C=10, solver=saga, train_score=0.6638383986714022, test_score=0.6594237373062
CV iteration : C=10, solver=saga, train_score=0.661417750357837, test_score=0.6632802152783771
C=10, penalty=l1, solver="saga", train_score=0.6630835131790033, test_score=0.6607713656419514
CV iteration : C=50, solver=saga, train_score=0.6640088113990698, test_score=0.6595895106065
CV iteration : C=50, solver=saga, train_score=0.6638804596043576, test_score=0.659398414086463
CV iteration : C=50, solver=saga, train_score=0.6614706536122342, test_score=0.6633522061085
C=50, penalty=l1, solver="saga", train_score=0.6631199748718872, test_score=0.660780043600513
CV iteration : C=100, solver=saga, train_score=0.6640104381459541, test_score=0.6595755593881
CV iteration : C=100, solver=saga, train_score=0.6638910554421718, test_score=0.6593993519834
CV iteration : C=100, solver=saga, train_score=0.6614777904224534, test_score=0.6633750696458
C=100, penalty=l1, solver="saga", train_score=0.6631264280035264, test_score=0.6607833270058
CV iteration : C=500, solver=saga, train_score=0.6640097493432193, test_score=0.6595722181299
CV iteration : C=500, solver=saga, train_score=0.6638894433506648, test_score=0.6593977106636
CV iteration : C=500, solver=saga, train_score=0.6614800618918251, test_score=0.6633857392965
C=500, penalty=l1, solver="saga", train_score=0.6631264181952364, test_score=0.6607852226967
CV iteration : C=1000, solver=saga, train_score=0.66401057004435, test_score=0.6595721595114
CV iteration : C=1000, solver=saga, train_score=0.663888710581798, test_score=0.6593950142097
CV iteration : C=1000, solver=saga, train_score=0.6614787576287666, test_score=0.6633857392965
C=1000, penalty=l1, solver="saga", train_score=0.6631260127516382, test_score=0.6607843043392
CV iteration : C=5000, solver=saga, train_score=0.6640112002255756, test_score=0.6595725698413
CV iteration : C=5000, solver=saga, train_score=0.6638894726614195, test_score=0.6593946624983
CV iteration : C=5000, solver=saga, train_score=0.6614792119226409, test_score=0.6633884360214
```

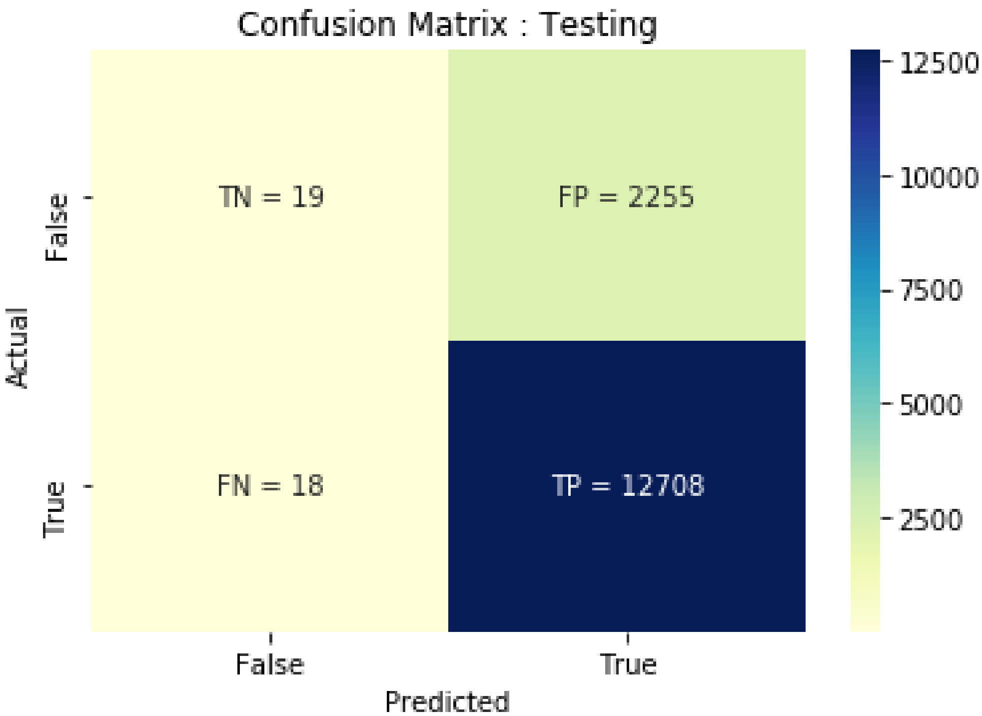
```
C=5000, penalty=l1, solver="saga", train_score=0.6631266282698787, test_score=0.660785222787070
CV iteration : C=10000, solver=saga, train_score=0.6640112002255756, test_score=0.659572745697
CV iteration : C=10000, solver=saga, train_score=0.6638902933625503, test_score=0.659394545261
CV iteration : C=10000, solver=saga, train_score=0.6614793438144109, test_score=0.663388436021
C=10000, penalty=l1, solver="saga", train_score=0.6631269458008456, test_score=0.6607852423265
```




```
Initializing Vectorizer
Training Model...
Saving Trained Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.6627319168112705
Area Under the Curve for Test :  0.6506057032390009
```


Area Under ROC Curve
train AUC =0.6627319168112705
train AUC =0.6506057032390009
TPF : True Positive Ratio
FPR : False Positive Ratio
1.0
0.8
0.6
0.4
0.2
0.0


Confusion Matrix : Training
TN = 38
FP = 5104
FN = 50
TP = 29808
Actual
False
True
Predicted
25000
20000
15000
10000
5000



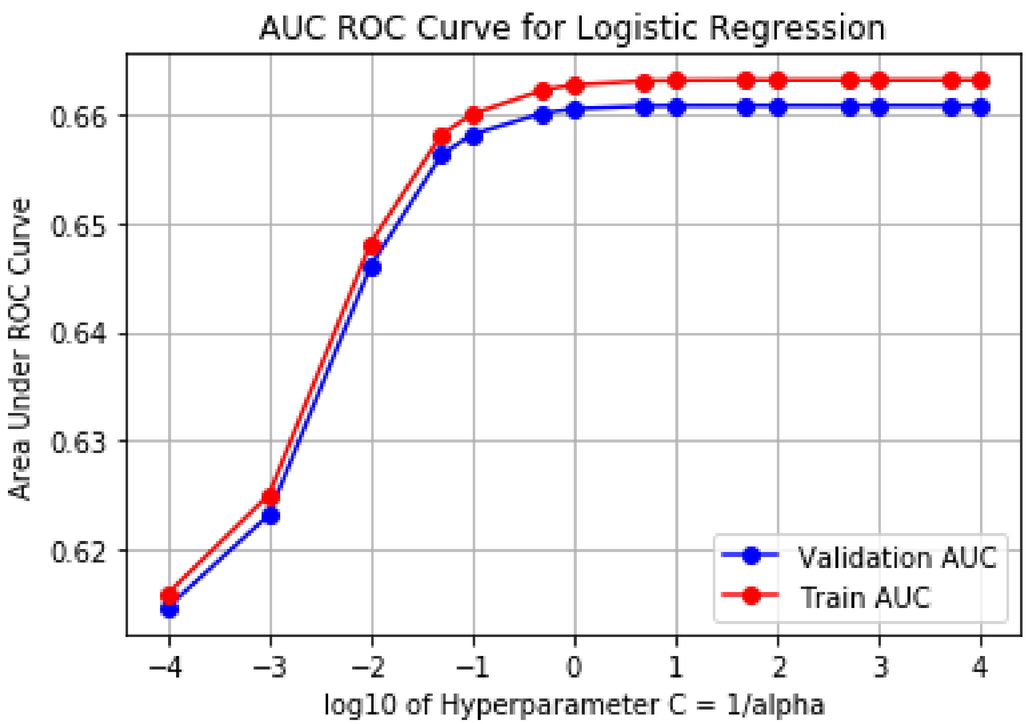
7.4.2 [5.3.2] Applying Logistic Regression with L2 regularization on AVG W2V, SET 3

In [60]:
```
# Please write all the code with proper documentation
csv_path = 'saved_models/Assignment5/Avg-W2Vec_log_reg_results_l2.csv'
cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='Avg-W2V
                                           penalty=['l2'], results_path=csv_path, ret
# Analysing best parameters
best_parameters = analyse_results(cv_results)

# retraining the model with best parameters
model_path = 'saved_models/Assignment5/{0}_log_reg_l2.pkl'.format('Avg-W2Vec')
log_reg = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'Avg-W2Vec', m

print('Retraining Vectorizer with Dx_train')
vectorizer_obj = get_vectorizer(W2V_model=w2v_model, train=Dx_train, vectorizer='Avg-W

# plotting AUC ROC
train_score, test_score = plot_AUC_ROC(log_reg, vectorizer_obj, Dx_train, Dx_test, Dy_

# appending the data results
prettytable_data.append(['Avg-Word2Vec', 'LogisticRegression', 'L2', best_parameters[
```

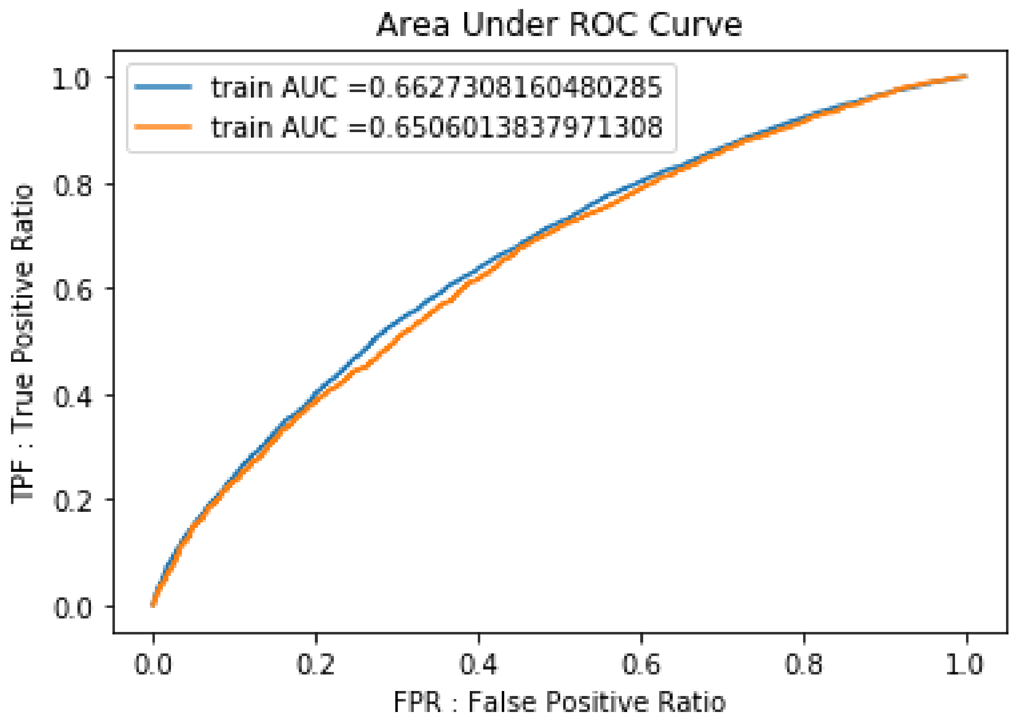
```
Performing Hyperparameter Tuning...
```

```
CV iteration : C=0.0001, solver=saga, train_score=0.6147310283942073, test_score=0.61650850010
CV iteration : C=0.0001, solver=saga, train_score=0.616703715460573, test_score=0.612442657854
CV iteration : C=0.0001, solver=saga, train_score=0.6162938535738742, test_score=0.61498421712
C=0.0001, penalty=l2, solver="saga", train_score=0.6159095324762182, test_score=0.61464512502
CV iteration : C=0.001, solver=saga, train_score=0.6243115270111794, test_score=0.62478456212
CV iteration : C=0.001, solver=saga, train_score=0.6253406422631185, test_score=0.62119159583
CV iteration : C=0.001, solver=saga, train_score=0.6254126600389391, test_score=0.623550481048
C=0.001, penalty=l2, solver="saga", train_score=0.625021609771079, test_score=0.62317554633429
CV iteration : C=0.01, solver=saga, train_score=0.6490087835538528, test_score=0.64678364040
CV iteration : C=0.01, solver=saga, train_score=0.6473274453889347, test_score=0.64505837881
CV iteration : C=0.01, solver=saga, train_score=0.6479435699010554, test_score=0.64612230304
C=0.01, penalty=l2, solver="saga", train_score=0.6480932662812809, test_score=0.64598810741928
CV iteration : C=0.05, solver=saga, train_score=0.6593885717953746, test_score=0.65576464927
CV iteration : C=0.05, solver=saga, train_score=0.6576928273945348, test_score=0.65624350433
CV iteration : C=0.05, solver=saga, train_score=0.657269871429558, test_score=0.6567580688354
C=0.05, penalty=l2, solver="saga", train_score=0.6581170902064891, test_score=0.6562554074810
CV iteration : C=0.1, solver=saga, train_score=0.661128560780393, test_score=0.657414586010492
CV iteration : C=0.1, solver=saga, train_score=0.6599546943665022, test_score=0.6579490114623
CV iteration : C=0.1, solver=saga, train_score=0.6589422737273587, test_score=0.6590352185238
C=0.1, penalty=l2, solver="saga", train_score=0.6600085096247513, test_score=0.65813293866557
CV iteration : C=0.5, solver=saga, train_score=0.663154153758943, test_score=0.659198583400324
CV iteration : C=0.5, solver=saga, train_score=0.6627829623617668, test_score=0.6592481160872
CV iteration : C=0.5, solver=saga, train_score=0.660721332503138, test_score=0.661865958936618
C=0.5, penalty=l2, solver="saga", train_score=0.662219482874616, test_score=0.6601042194747365
CV iteration : C=1, solver=saga, train_score=0.6636040591877931, test_score=0.6595050412551595
CV iteration : C=1, solver=saga, train_score=0.6633454210885604, test_score=0.6593875110334793
CV iteration : C=1, solver=saga, train_score=0.6610751981219549, test_score=0.6625516305571293
C=1, penalty=l2, solver="saga", train_score=0.6626748927994361, test_score=0.6604813942819227
CV iteration : C=5, solver=saga, train_score=0.6639661642510208, test_score=0.659624681745159
CV iteration : C=5, solver=saga, train_score=0.6638100405162561, test_score=0.6594032794273108
CV iteration : C=5, solver=saga, train_score=0.6613922219730282, test_score=0.6632366573086403
C=5, penalty=l2, solver="saga", train_score=0.6630561422467683, test_score=0.6607548728270367
CV iteration : C=10, solver=saga, train_score=0.664000633698516, test_score=0.6596116684238558
CV iteration : C=10, solver=saga, train_score=0.6638514566126088, test_score=0.659405155221372
CV iteration : C=10, solver=saga, train_score=0.6614448321346031, test_score=0.663310582745838
C=10, penalty=l2, solver="saga", train_score=0.663098974148576, test_score=0.6607758021303555
CV iteration : C=50, solver=saga, train_score=0.6640082691501084, test_score=0.659583941842880
CV iteration : C=50, solver=saga, train_score=0.663887362287083, test_score=0.6593909109102162
CV iteration : C=50, solver=saga, train_score=0.6614730716280169, test_score=0.663370848685123
C=50, penalty=l2, solver="saga", train_score=0.663122901021736, test_score=0.660781900479407
CV iteration : C=100, solver=saga, train_score=0.664009734687842, test_score=0.659579545450548
CV iteration : C=100, solver=saga, train_score=0.6638882709204779, test_score=0.65939319703422
CV iteration : C=100, solver=saga, train_score=0.6614753870613118, test_score=0.66337213842312
C=100, penalty=l2, solver="saga", train_score=0.6631244642232105, test_score=0.660781626969300
CV iteration : C=500, solver=saga, train_score=0.6640081812178442, test_score=0.65958142124461
CV iteration : C=500, solver=saga, train_score=0.6638886666156659, test_score=0.65939495559116
CV iteration : C=500, solver=saga, train_score=0.6614789041751776, test_score=0.66337905610876
C=500, penalty=l2, solver="saga", train_score=0.6631252506695625, test_score=0.660785144314844
```

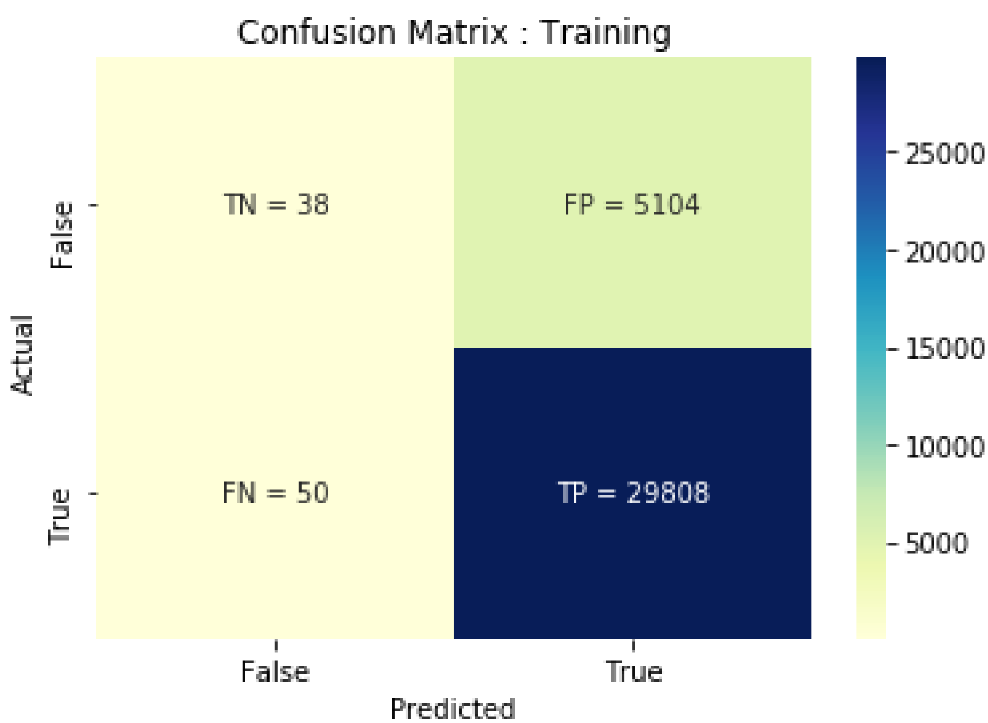
```
CV iteration : C=1000, solver=saga, train_score=0.6640073605167134, test_score=0.659580131636
CV iteration : C=1000, solver=saga, train_score=0.6638887252371752, test_score=0.659393548745
CV iteration : C=1000, solver=saga, train_score=0.6614791826133586, test_score=0.663380287222
C=1000, penalty=l2, solver="saga", train_score=0.6631250894557491, test_score=0.66078465586803
CV iteration : C=5000, solver=saga, train_score=0.6640075803473735, test_score=0.659582417760
CV iteration : C=5000, solver=saga, train_score=0.6638895019721742, test_score=0.659391731570
CV iteration : C=5000, solver=saga, train_score=0.6614785964277143, test_score=0.663379994100
C=5000, penalty=l2, solver="saga", train_score=0.6631252262490873, test_score=0.66078471447678
CV iteration : C=10000, solver=saga, train_score=0.6640077122457695, test_score=0.659582241904
CV iteration : C=10000, solver=saga, train_score=0.6638896631813247, test_score=0.659392024662
CV iteration : C=10000, solver=saga, train_score=0.6614784938452265, test_score=0.663379876851
C=10000, penalty=l2, solver="saga", train_score=0.6631252897574402, test_score=0.66078471447728
```



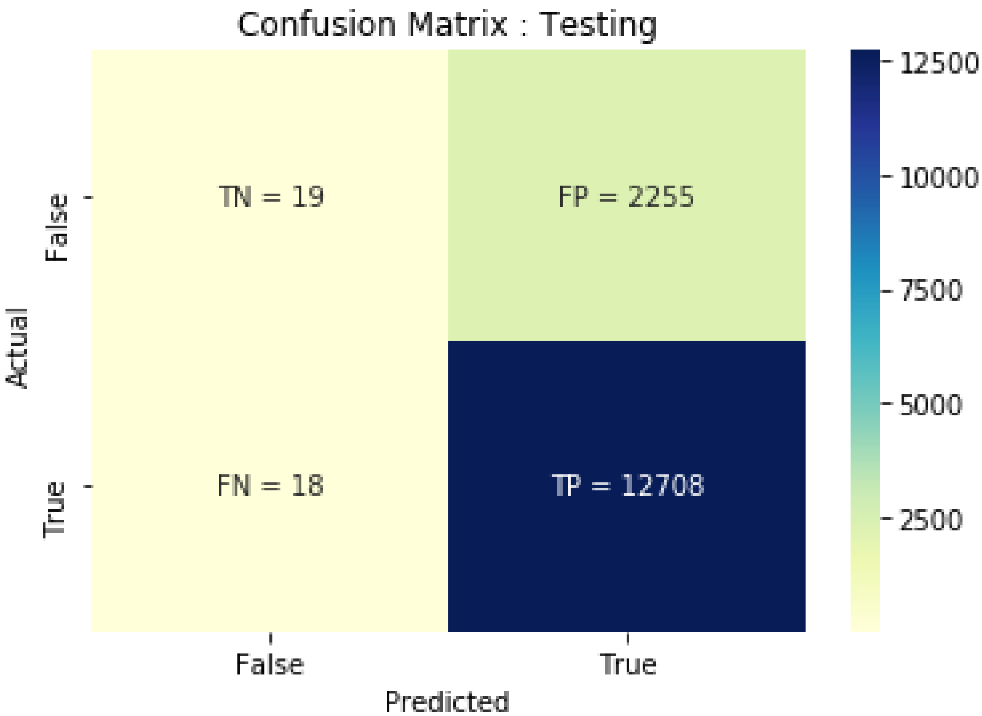

```
Initializing Vectorizer
Training Model...
Saving Trained Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.6627308160480285
Area Under the Curve for Test :  0.6506013837971308
```



Area Under ROC Curve
train AUC =0.6627308160480285
train AUC =0.6506013837971308
TPF : True Positive Ratio
FPR : False Positive Ratio
0.0
0.2
0.4
0.6
0.8
1.0



Confusion Matrix : Training
TN = 38
FP = 5104
FN = 50
TP = 29808
Actual
False
True
Predicted
25000
20000
15000
10000
5000



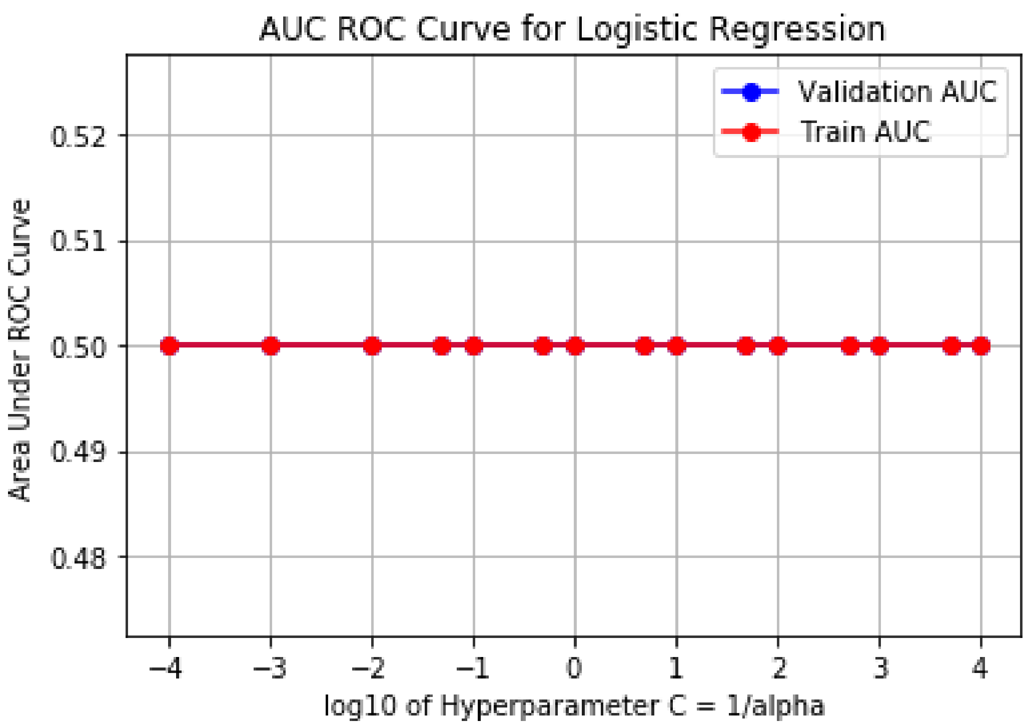
7.5 [5.4] Logistic Regression on TFIDF W2V, SET 4

7.5.1 [5.4.1] Applying Logistic Regression with L1 regularization on TFIDF W2V, SET 4

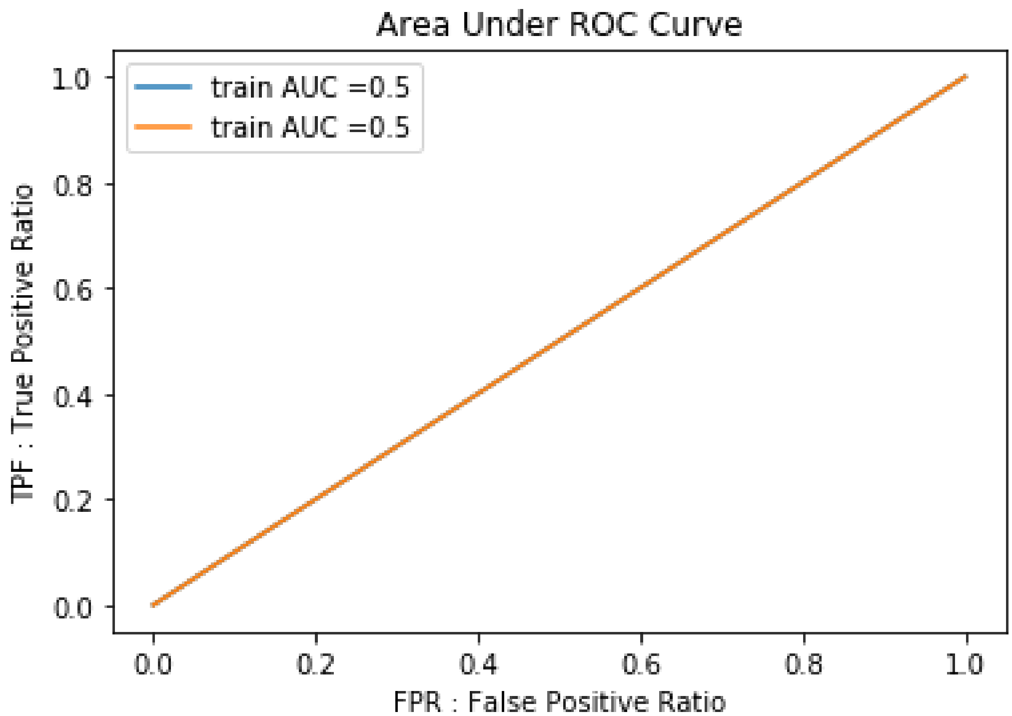
```
In [61]: # Please write all the code with proper documentation
         csv_path = 'saved_models/Assignment5/TFIDF-W2Vec_log_reg_results_l1.csv'
         cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='TFIDF-W
                                                     penalty=['l1'], results_path=csv_path, ret
         # Analysing best parameters
         best_parameters = analyse_results(cv_results)

         # retraining the model with best parameters
         model_path = 'saved_models/Assignment5/{0}_log_reg_l1.pkl'.format('TFIDF-W2Vec')
         log_reg = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'TFIDF-W2Vec'

         print('Retraining Vectorizer with Dx_train')
         vectorizer_obj = get_vectorizer(W2V_model=w2v_model, train=Dx_train, vectorizer='TFIDF

         # plotting AUC ROC
         train_score, test_score = plot_AUC_ROC(log_reg, vectorizer_obj, Dx_train, Dx_test, Dy_

         # appending the data results
         prettytable_data.append(['TFIDF-Word2Vec', 'LogisticRegression', 'L1', best_parameters
```

```
Performing Hyperparameter Tuning...

CV iteration : C=0.0001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.0001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.0001, solver=saga, train_score=0.5, test_score=0.5
C=0.0001, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=0.001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.001, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.001, solver=saga, train_score=0.5, test_score=0.5
C=0.001, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=0.01, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.01, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.01, solver=saga, train_score=0.5, test_score=0.5
C=0.01, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=0.05, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.05, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.05, solver=saga, train_score=0.5, test_score=0.5
C=0.05, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=0.1, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.1, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.1, solver=saga, train_score=0.5, test_score=0.5
C=0.1, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=0.5, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.5, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=0.5, solver=saga, train_score=0.5, test_score=0.5
C=0.5, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=1, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=1, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=1, solver=saga, train_score=0.5, test_score=0.5
C=1, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=5, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=5, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=5, solver=saga, train_score=0.5, test_score=0.5
C=5, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=10, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=10, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=10, solver=saga, train_score=0.5, test_score=0.5
C=10, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=50, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=50, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=50, solver=saga, train_score=0.5, test_score=0.5
C=50, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=100, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=100, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=100, solver=saga, train_score=0.5, test_score=0.5
C=100, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=500, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=500, solver=saga, train_score=0.5, test_score=0.5
```

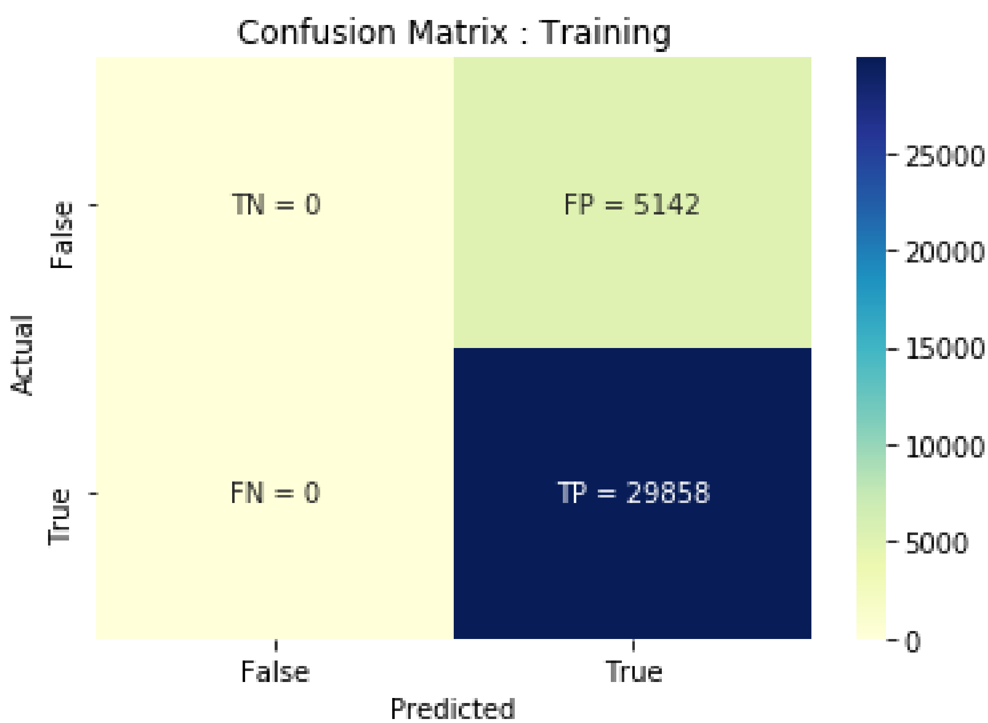
```
CV iteration : C=500, solver=saga, train_score=0.5, test_score=0.5
C=500, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=1000, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=1000, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=1000, solver=saga, train_score=0.5, test_score=0.5
C=1000, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=5000, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=5000, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=5000, solver=saga, train_score=0.5, test_score=0.5
C=5000, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
CV iteration : C=10000, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=10000, solver=saga, train_score=0.5, test_score=0.5
CV iteration : C=10000, solver=saga, train_score=0.5, test_score=0.5
C=10000, penalty=l1, solver="saga", train_score=0.5, test_score=0.5
```



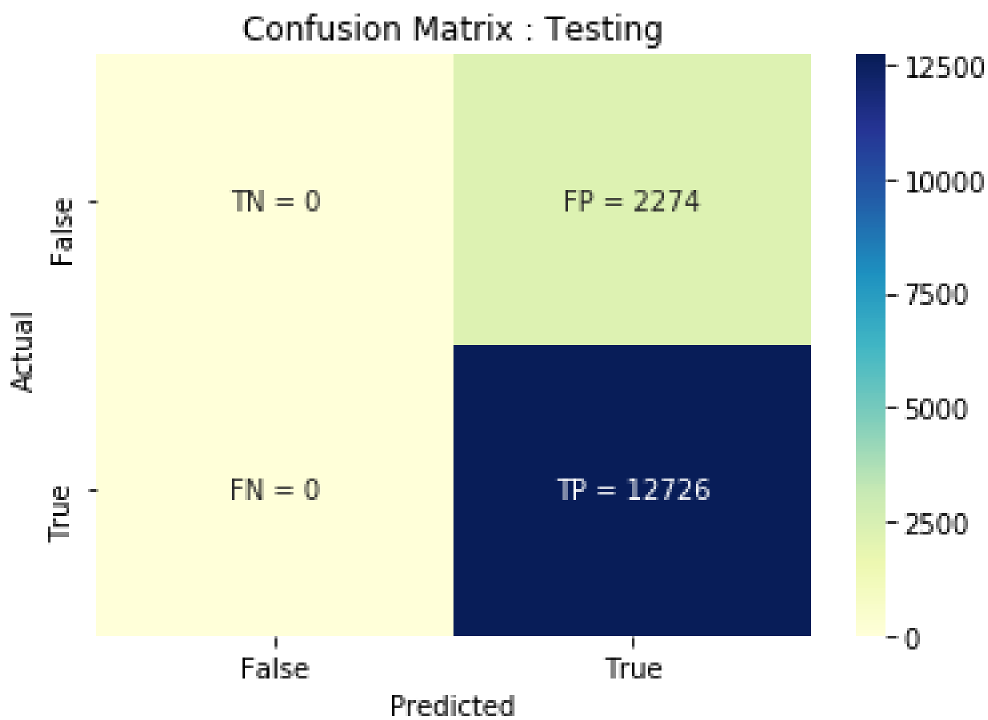

```
Initializing Vectorizer
Training Model...
Saving Trained Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.5
Area Under the Curve for Test :  0.5
```



Area Under ROC Curve
train AUC =0.5
train AUC =0.5
TPF : True Positive Ratio
FPR : False Positive Ratio
1.0
0.8
0.6
0.4
0.2
0.0



Confusion Matrix : Training
TN = 0
FP = 5142
FN = 0
TP = 29858
Actual
False
True
Predicted
25000
20000
15000
10000
5000
0



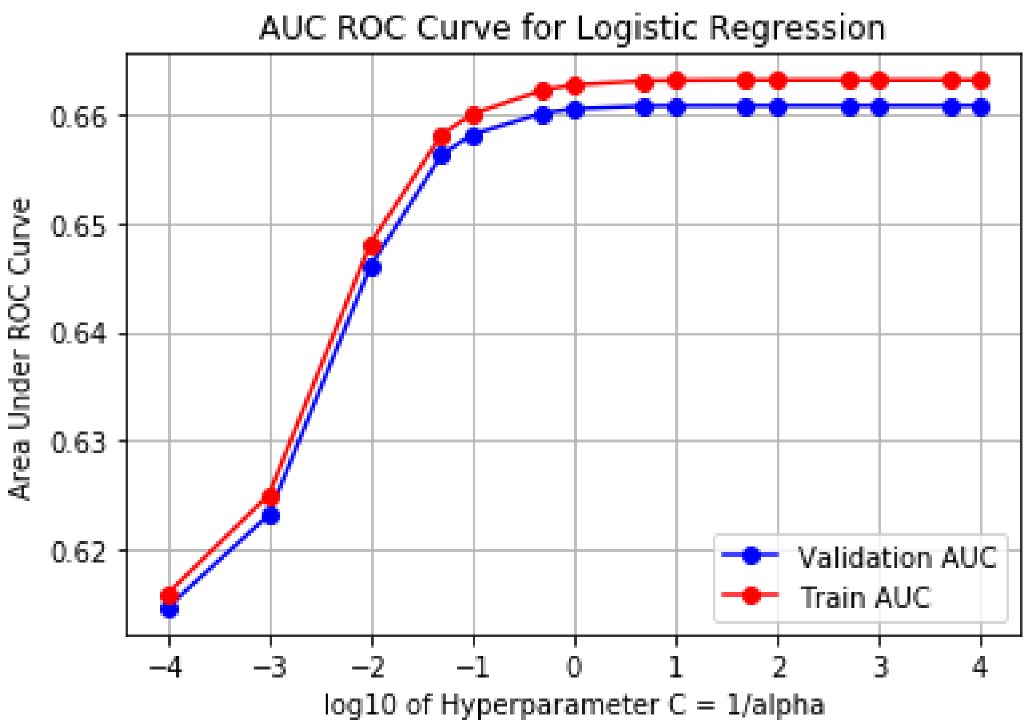
7.5.2 [5.4.2] Applying Logistic Regression with L2 regularization on TFIDF W2V, SET 4

In [62]:
```
# Please write all the code with proper documentation
csv_path = 'saved_models/Assignment5/Avg-W2Vec_log_reg_results_l2.csv'
cv_results = perform_hyperparameter_tuning(X=Dx_train, Y=Dy_train, vectorizer='Avg-W2V
                                           penalty=['l2'], results_path=csv_path, reti
# Analysing best parameters
best_parameters = analyse_results(cv_results)

# retraining the model with best parameters
model_path = 'saved_models/Assignment5/{0}_log_reg_l2.pkl'.format('Avg-W2Vec')
log_reg = retrain_with_best_params(Dx_train, Dy_train, best_parameters, 'Avg-W2Vec', r

print('Retraining Vectorizer with Dx_train')
vectorizer_obj = get_vectorizer(W2V_model=w2v_model, train=Dx_train, vectorizer='Avg-V

# plotting AUC ROC
train_score, test_score = plot_AUC_ROC(log_reg, vectorizer_obj, Dx_train, Dx_test, Dy_

# appending the data results
prettytable_data.append(['TFIDF-Word2Vec', 'LogisticRegression', 'L1', best_parameter
```

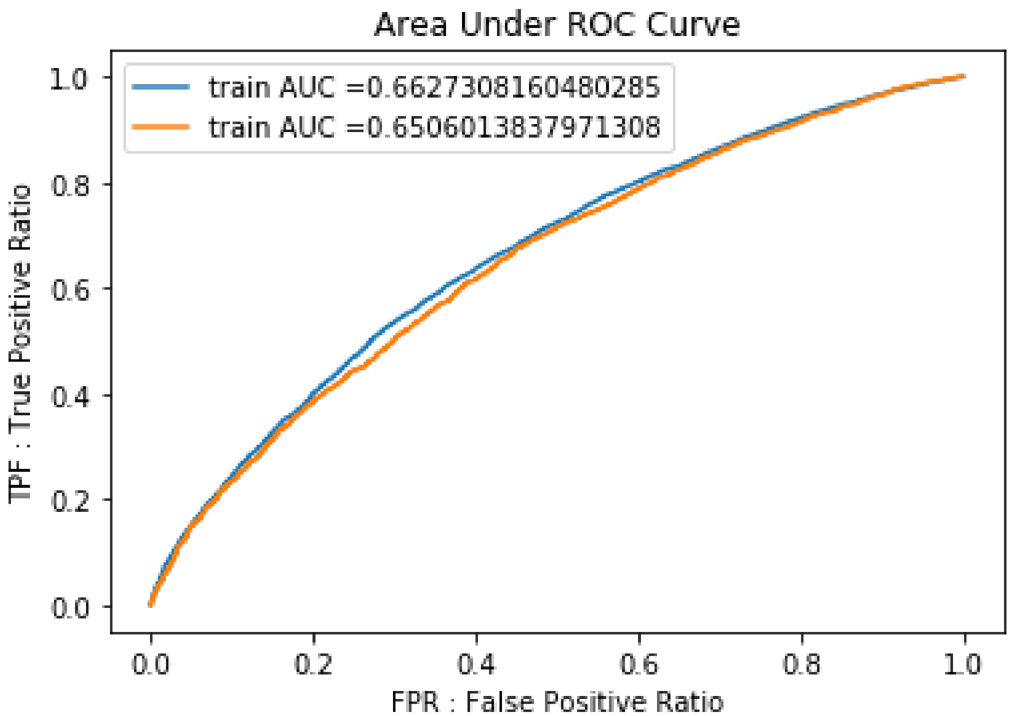
```
Performing Hyperparameter Tuning...
```

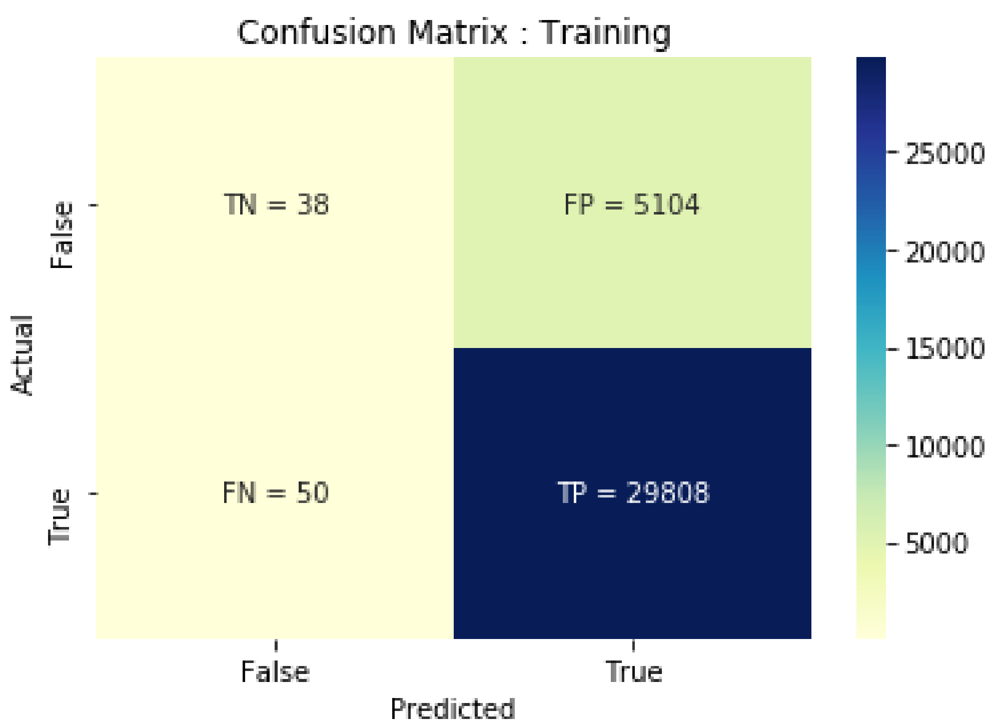
```
CV iteration : C=0.0001, solver=saga, train_score=0.6147316439200554, test_score=0.616510434514
CV iteration : C=0.0001, solver=saga, train_score=0.6167041551218931, test_score=0.612441778576
CV iteration : C=0.0001, solver=saga, train_score=0.6162951285276506, test_score=0.614985213740
C=0.0001, penalty=l2, solver="saga", train_score=0.6159103091898663, test_score=0.614645808943
CV iteration : C=0.001, solver=saga, train_score=0.624311336491274, test_score=0.6247843276468
CV iteration : C=0.001, solver=saga, train_score=0.6253401732910437, test_score=0.621191419977
CV iteration : C=0.001, solver=saga, train_score=0.6254125721110925, test_score=0.623550656922
C=0.001, penalty=l2, solver="saga", train_score=0.6250213606311368, test_score=0.6231754681820
CV iteration : C=0.01, solver=saga, train_score=0.6490099120179077, test_score=0.6467838162584
CV iteration : C=0.01, solver=saga, train_score=0.6473275186658213, test_score=0.6450580271031
CV iteration : C=0.01, solver=saga, train_score=0.647943379390721, test_score=0.64612236166504
C=0.01, penalty=l2, solver="saga", train_score=0.64809360335815, test_score=0.6459880683422098
CV iteration : C=0.05, solver=saga, train_score=0.6593886450722612, test_score=0.6557645320403
CV iteration : C=0.05, solver=saga, train_score=0.6576939558585897, test_score=0.6562437974231
CV iteration : C=0.05, solver=saga, train_score=0.6572695929913768, test_score=0.6567597689446
C=0.05, penalty=l2, solver="saga", train_score=0.6581173979740759, test_score=0.65625603280271
CV iteration : C=0.1, solver=saga, train_score=0.6611285754357703, test_score=0.65741446877336
CV iteration : C=0.1, solver=saga, train_score=0.6599532727949007, test_score=0.65795288028764
CV iteration : C=0.1, solver=saga, train_score=0.6589434754079295, test_score=0.65903697725746
C=0.1, penalty=l2, solver="saga", train_score=0.6600084412128668, test_score=0.658134775439490
CV iteration : C=0.5, solver=saga, train_score=0.6631543589342257, test_score=0.65919922820453
CV iteration : C=0.5, solver=saga, train_score=0.6627829623617668, test_score=0.65924799885013
CV iteration : C=0.5, solver=saga, train_score=0.6607212445752915, test_score=0.66186613480998
C=0.5, penalty=l2, solver="saga", train_score=0.6622195219570947, test_score=0.660104453954884
CV iteration : C=1, solver=saga, train_score=0.6636037367694918, test_score=0.6595052757294172
CV iteration : C=1, solver=saga, train_score=0.663345406433183, test_score=0.6593882730748168
CV iteration : C=1, solver=saga, train_score=0.6610757696529581, test_score=0.6625511029370383
C=1, penalty=l2, solver="saga", train_score=0.6626749709518777, test_score=0.6604815505804241
CV iteration : C=5, solver=saga, train_score=0.6639657978665874, test_score=0.6596247403637234
CV iteration : C=5, solver=saga, train_score=0.6638098646517281, test_score=0.6594037483758262
CV iteration : C=5, solver=saga, train_score=0.6613922512823104, test_score=0.663235250321731
C=5, penalty=l2, solver="saga", train_score=0.6630559712668753, test_score=0.6607545796870936
CV iteration : C=10, solver=saga, train_score=0.6640005164554973, test_score=0.659611668423855
CV iteration : C=10, solver=saga, train_score=0.6638511341943074, test_score=0.659404393180034
CV iteration : C=10, solver=saga, train_score=0.6614454036656063, test_score=0.663310641370292
C=10, penalty=l2, solver="saga", train_score=0.6630990181051369, test_score=0.6607755676580611
CV iteration : C=50, solver=saga, train_score=0.6640082838054856, test_score=0.659583824605752
CV iteration : C=50, solver=saga, train_score=0.6638883735081191, test_score=0.659390852291651
CV iteration : C=50, solver=saga, train_score=0.661472954390888, test_score=0.6633709073095784
C=50, penalty=l2, solver="saga", train_score=0.6631232039014976, test_score=0.6607818614023274
CV iteration : C=100, solver=saga, train_score=0.6640098958969927, test_score=0.65957960406911
CV iteration : C=100, solver=saga, train_score=0.6638891795538727, test_score=0.65939243499289
CV iteration : C=100, solver=saga, train_score=0.6614752844788241, test_score=0.66337213842312
C=100, penalty=l2, solver="saga", train_score=0.6631247866432298, test_score=0.660781392495042
CV iteration : C=500, solver=saga, train_score=0.6640079613871842, test_score=0.65958171433743
CV iteration : C=500, solver=saga, train_score=0.6638886519602887, test_score=0.65939448664264
CV iteration : C=500, solver=saga, train_score=0.661478962793742, test_score=0.663379056108762
C=500, penalty=l2, solver="saga", train_score=0.6631251920470715, test_score=0.660785085696280
```

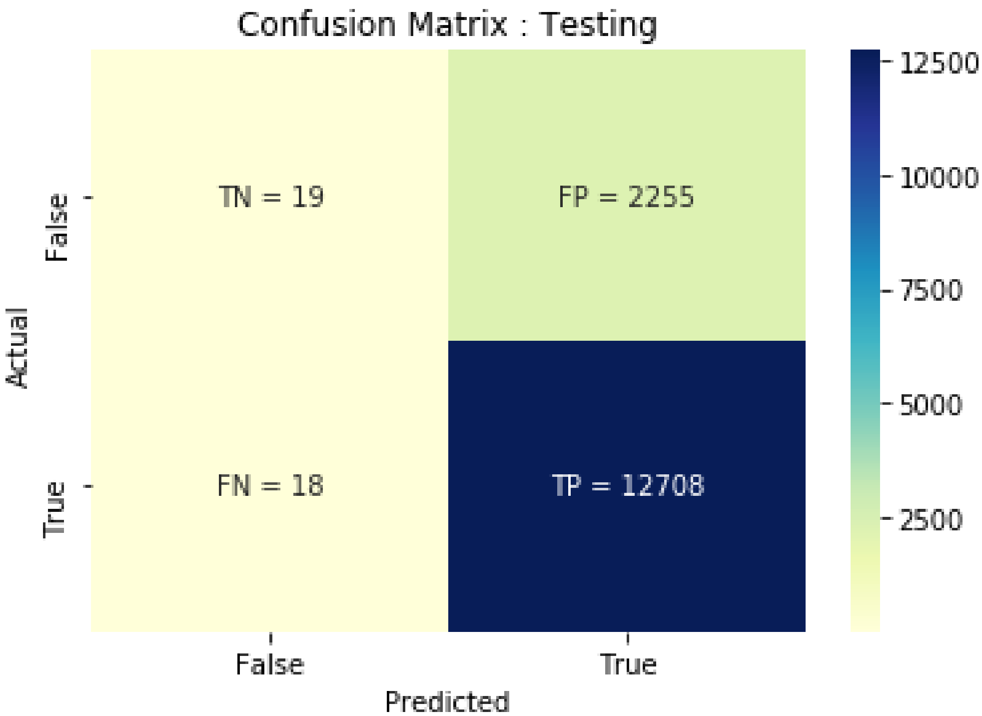
```
CV iteration : C=1000, solver=saga, train_score=0.6640075510366188, test_score=0.6595804833475
CV iteration : C=1000, solver=saga, train_score=0.66388875454793, test_score=0.659393548745615
CV iteration : C=1000, solver=saga, train_score=0.6614794903608219, test_score=0.6633802285978
C=1000, penalty=l2, solver="saga", train_score=0.6631252653151235, test_score=0.66078475356368
CV iteration : C=5000, solver=saga, train_score=0.6640076243135056, test_score=0.6595823591416
CV iteration : C=5000, solver=saga, train_score=0.6638895899044381, test_score=0.6593920832815
CV iteration : C=5000, solver=saga, train_score=0.6614784498813032, test_score=0.6633799354755
C=5000, penalty=l2, solver="saga", train_score=0.6631252213664157, test_score=0.66078479263290
CV iteration : C=10000, solver=saga, train_score=0.6640076096581282, test_score=0.659582593615
CV iteration : C=10000, solver=saga, train_score=0.6638897804243435, test_score=0.659391966044
CV iteration : C=10000, solver=saga, train_score=0.6614786110823554, test_score=0.663379700977
C=10000, penalty=l2, solver="saga", train_score=0.6631253337216091, test_score=0.6607847535460
```




```
Loading Model...
Retraining Vectorizer with Dx_train
Area Under the Curve for Train :  0.6627308160480285
Area Under the Curve for Test :  0.6506013837971308
```


Area Under ROC Curve
train AUC =0.6627308160480285
train AUC =0.6506013837971308
TPF : True Positive Ratio
FPR : False Positive Ratio
0.0
0.2
0.4
0.6
0.8
1.0


Confusion Matrix : Training
TN = 38
FP = 5104
FN = 50
TP = 29808
Actual
Predicted
False
True
25000
20000
15000
10000
5000



8 [6] Conclusions

```
In [63]: from prettytable import PrettyTable
```

```
In [64]: # Please compare all your models using Prettytable library
         x = PrettyTable()

         x.field_names = ["Vectorizer", "Model", "Penalty", "Hyper parameter: 1/C", "Train AUC
         [x.add_row(i) for i in prettytable_data]
         print(x)
```

```
+----------------+--------------------+---------+----------------------+--------------------+--
|   Vectorizer   |       Model        | Penalty | Hyper parameter: 1/C |     Train AUC      |
+----------------+--------------------+---------+----------------------+--------------------+--
|      BOW       | LogisticRegression |    L2   |         0.1          | 0.9362955972935451 | 0
|     TFIDF      | LogisticRegression |    L1   |         1.0          | 0.9424295841754171 | 0
|     TFIDF      | LogisticRegression |    L2   |         1.0          | 0.9419734676196749 | 0
|  Avg Word2Vec  | LogisticRegression |    L1   |       10000.0        | 0.6627319168112705 | 0
|  Avg-Word2Vec  | LogisticRegression |    L2   |        500.0         | 0.6627308160480285 | 0
| TFIDF-Word2Vec | LogisticRegression |    L1   |        0.0001        |        0.5         |
| TFIDF-Word2Vec | LogisticRegression |    L1   |        500.0         | 0.6627308160480285 | 0
+----------------+--------------------+---------+----------------------+--------------------+--
```