Amazon Fine Food Reviews Analysis

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

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/ (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 the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

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 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 id 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 os
import re
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sb
import pickle
import math

from sklearn import metrics
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.metrics import roc_curve,auc

from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.stem import PorterStemmer

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

from sklearn.preprocessing import StandardScaler
#TSNE
from sklearn.manifold import TSNE
from bs4 import BeautifulSoup
```

In [2]:

```
# Temporarily Suppressing Warnings
def fxn():
    warnings.warn("deprecated", DeprecationWarning)

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    fxn()
```

[1]. Reading Data

In [3]:

```
# using the SQLite Table to read data.
# con = sqlite3.connect('./amazon-fine-food-reviews/database.sqlite')

con = sqlite3.connect('D:/Appliedai/Data/amazon-fine-food-reviews/database.sqlite')

#filetering only positve and negative reviews
#reviews not taking in to consideration with score = 3

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 50
000""", con)

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

#changing reviews with score less than 3 to be positive and vice versa
actual_score = filtered_data['Score']
positivenegative = actual_score.map(partition)
filtered_data['Score']=positivenegative
print('Number of data point in our data',filtered_data.shape)
filtered_data.head(5)
```

```
Number of data point in our data (50000, 10)
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Sc
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	

Exploratory Data Analysis

[2] 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 [4]:

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

In [5]:

```
display.head()
```

Out[5]:

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

As can be seen above the same user has multiple reviews of the with the 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 [6]:

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

In [7]:

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

Out[7]:

```
(46072, 10)
```

In [8]:

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

Out[8]:

```
92.144
```

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

In [9]:

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

Out[9]:

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

In [10]:
```
final = final[final.HelpfulnessNumerator <= final.HelpfulnessDenominator]
```

In [11]:
```
final.shape
final['Score'].value_counts()
```

Out[11]:
```
1    38479
0     7592
Name: Score, dtype: int64
```

Text Preprocessing.

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 [12]:
```
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 [13]:

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

In [14]:

```
# 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):
for sentance in 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)
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

In [15]:

```
# Add pre processed reviews in to final df
# final['preprocessed_reviews'] = preprocessed_reviews
```

In [16]:

```
preprocessed_reviews[100]
```

Out[16]:

```
'fyi customers item beef ocean fish formula red bag haste purchased thinking version chicken rice formula woops went bought bag chicken rice mix beef fish not wreak havoc pup digestive system say started feeding pup starting stinky farts never also roshan right fish breath ick overall dog no issues formula stinky stick chicken rice bag done'
```

[3.2] Preprocess Summary

In [17]:

```
##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):
for sentance in (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)
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopw
ords)
    preprocessed_summary.append(sentance.strip())
```

```
C:\Users\Saraswathi\AppData\Local\Continuum\anaconda3\lib\site-packages\bs4\__init_
_.py:273: UserWarning: "b'...'" looks like a filename, not markup. You should probab
ly open this file and pass the filehandle into Beautiful Soup.
  ' Beautiful Soup.' % markup)
```

In [18]:

```
preprocessed_summary[100]
```

Out[18]:

```
'wrong bag pictured'
```

Featurization

BAG OF WORDS, Bi-Grams and n-Grams, TF-IDF, Word2Vec, Converting text into vectors using wAvg W2V, TFIDF-W2V, Avg W2v, TFIDF weighted W2v

In [19]:

```
#storing label i.e positive and negative in another variable for tsne plot
labels = final['Score']
```

BAG OF WORDS

In [20]:

```
#BOW
count_vect = CountVectorizer()
count_vect.fit(preprocessed_reviews)
print('some feature names are',count_vect.get_feature_names()[:10])
print('='*50)

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

```
some feature names are ['aa', 'aaa', 'aaaa', 'aaaaa', 'aaaaaaaaaaaa', 'aaaaaaaaaaaaa
aa', 'aaaaaaahhhhhh', 'aaaaaawwwwwwwwww', 'aaaaah', 'aaaand']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (46071, 39364)
the number of unique words 39364
```

Bi-Grams and n-Grams.

In [21]:
```
#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))
count_vect = CountVectorizer(ngram_range=(1,2),min_df=10,max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams",final_bigram_c
ounts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer (46071, 5000)
the number of unique words including both unigrams and bigrams 5000
```

TF-IDF

In [22]:
```
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_name
s()[:10])
print('='*50)
final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf
.get_shape()[1])
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able buy', 'ab
le chew', 'able drink', 'able eat', 'able enjoy', 'able feed', 'able figure', 'able
find']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (46071, 27311)
the number of unique words including both unigrams and bigrams  27311
```

Word2Vec

In [23]:
```
# Train your own Word2Vec model using your own text corpus
# i = 0
list_of_sentance = []
for sentance in preprocessed_reviews:
#     list_of_sentance.append(sentance)
    list_of_sentance.append(sentance.split())
# print((list_of_sentance))
```

```
In [24]: # Using Google News Word2Vectors
is_your_ram_gt_16gb = False
want_to_use_google_w2v = True
want_to_train_w2v = True

# print(list_of_sentance)

if want_to_train_w2v:
   # min_count = 5 considers only words that occured atleast 5 times
    w2v_model = Word2Vec(list_of_sentance,min_count = 5 ,size = 50 ,workers = 4)
    print(type(w2v_model))
    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_16gb :
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative30
0.bin',binary = True)
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True,
 to train your own w2v ")
```

```
<class 'gensim.models.word2vec.Word2Vec'>
[('fantastic', 0.8399447798728943), ('awesome', 0.8316543698310852), ('terrific', 0.
822209894657135), ('good', 0.7968133091926575), ('excellent', 0.7788823843002319),
('wonderful', 0.7613141536712646), ('amazing', 0.7610692977905273), ('perfect', 0.75
5174994468689), ('nice', 0.7039507031440735), ('decent', 0.6919985413551331)]
==================================================
[('best', 0.725080132484436), ('greatest', 0.720651388168335), ('tastiest', 0.713737
6070022583), ('closest', 0.682880163192749), ('experienced', 0.658531904220581), ('d
isgusting', 0.6545877456665039), ('awful', 0.6536005735397339), ('nastiest', 0.62769
65737342834), ('eaten', 0.6232050657272339), ('softest', 0.600709080696106)]
```

```
In [25]: print(type(w2v_model))
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])
```

```
<class 'gensim.models.word2vec.Word2Vec'>
number of words that occured minimum 5 times  12798
sample words  ['dogs', 'loves', 'chicken', 'product', 'china', 'wont', 'buying', 'an
ymore', 'hard', 'find', 'products', 'made', 'usa', 'one', 'isnt', 'bad', 'good', 'ta
ke', 'chances', 'till', 'know', 'going', 'imports', 'love', 'saw', 'pet', 'store',
'tag', 'attached', 'regarding', 'satisfied', 'safe', 'available', 'victor', 'traps',
'unreal', 'course', 'total', 'fly', 'pretty', 'stinky', 'right', 'nearby', 'used',
'bait', 'seasons', 'ca', 'not', 'beat', 'great']
```

Converting text into vectors using wAvg W2V, TFIDF-W2V

Avg W2v

In [26]:

```python
#average word2vec
#compute average word2 vec for each review
sent_vectors = [];
# for sent in tqdm(list_of_sentance):
for sent in (list_of_sentance):

    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use google's w2v
    cnt_words = 0;
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec  /=cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
46071
50
```

TFIDF weighted W2v

In [27]:

```python
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(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 [28]:

```python
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names()

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

tfidf_sent_vectors = [] ; # the tfidf-w2v for each sentence/review is stored in this list
row = 0
# for sent in tqdm(list_of_sentance):
for sent in (list_of_sentance):
    sent_vec = np.zeros(50)
    weight_sum = 0; # as word vectors are of zero length
    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
```

[5] Assignment 8: Decision Trees

1. **Apply Decision Trees on these feature sets**

    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
    - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **The hyper paramter tuning (best `depth` in range [4,6, 8, 9,10,12,14,17] , and the best `min_samples_split` in range [2,10,20,30,40,50])**

    - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Graphviz**

    - Visualize your decision tree with Graphviz. It helps you to understand how a decision is being made, given a new vector.
    - Since feature names are not obtained from word2vec related models, visualize only BOW & TFIDF decision trees using Graphviz
    - Make sure to print the words in each node of the decision tree instead of printing its index.
    - Just for visualization purpose, limit max_depth to 2 or 3 and either embed the generated images of graphviz in your notebook, or directly upload them as .png files.

4. **Feature importance**

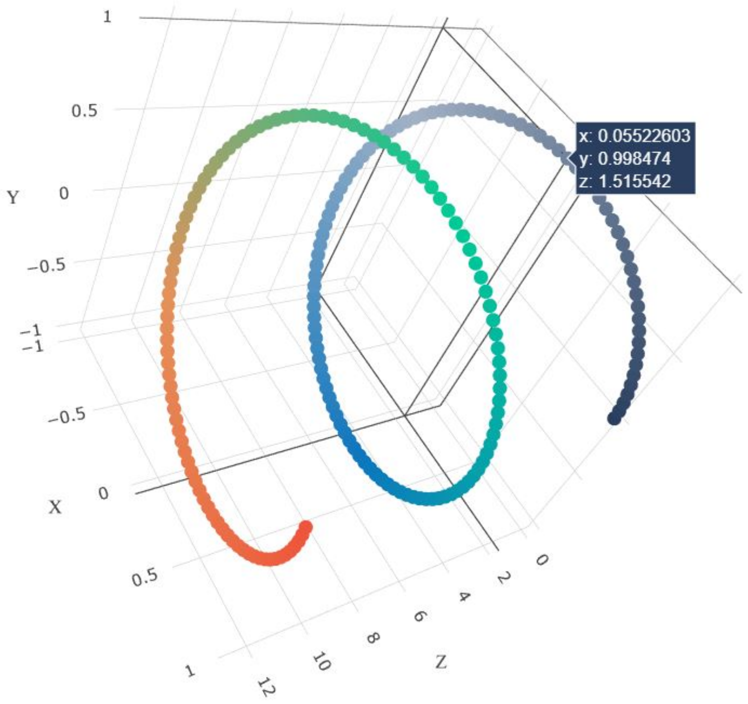
    - Find the top 20 important features from both feature sets Set 1 and Set 2 using `feature_importances_` method of Decision Tree Classifier (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html) and print their corresponding feature names

5. **Feature engineering**

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

6. **Representation of results**

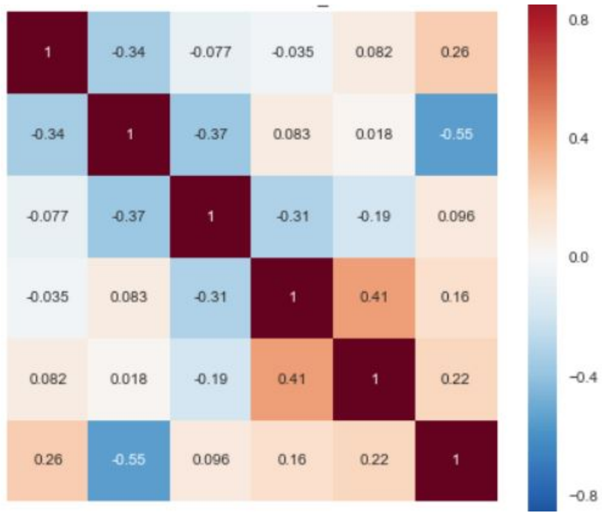
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



with X-axis as **min_sample_split**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

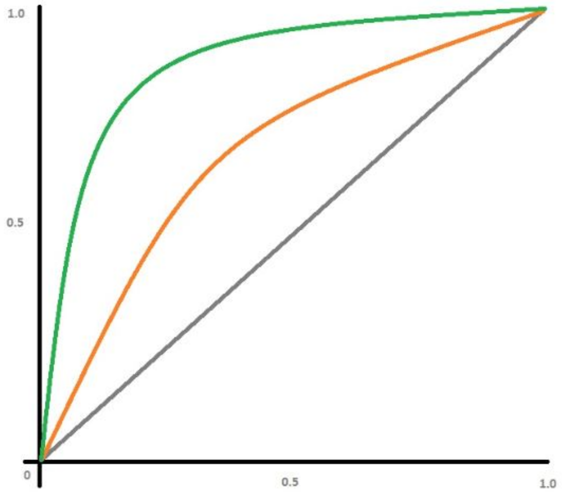
or

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



[seaborn heat maps (https://seaborn.pydata.org/generated/seaborn.heatmap.html)](https://seaborn.pydata.org/generated/seaborn.heatmap.html) with rows as **min_sample_split**, columns as **max_depth**, and values inside the cell representing **AUC Score**

- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.



- Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps. (https://seaborn.pydata.org/generated/seaborn.heatmap.html) (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

7. **Conclusion** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library (https://seaborn.pydata.org/generated/seaborn.heatmap.html) link (http://zetcode.com/python/prettytable/)

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. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

[5] Applying Decision Trees

In [29]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV

from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier


x = preprocessed_reviews
y = final['Score'].values

base_x_train, base_x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3,r
andom_state = 0)
```

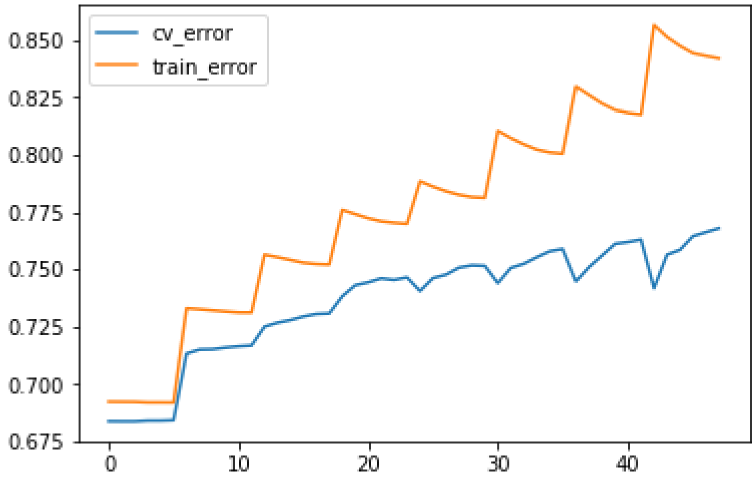
[5.1] Applying Decision Trees on BOW, SET 1

In [30]:

```
from sklearn.tree import DecisionTreeClassifier

bow_cnt_vect = CountVectorizer()
x_train = bow_cnt_vect.fit_transform(base_x_train)
x_test = bow_cnt_vect.transform(base_x_test)

depth = [4,6, 8, 9,10,12,14,17]
sample_split = [2,10,20,30,40,50]

param_grid = { 'max_depth' : depth ,
               'min_samples_split' : sample_split}

model  = GridSearchCV(DecisionTreeClassifier(), param_grid, scoring='roc_auc', cv =3
, n_jobs= -1, pre_dispatch= 2 ,return_train_score=True)
model.fit(x_train,y_train)
cv_error = model.cv_results_['mean_test_score']
train_error = model.cv_results_['mean_train_score']
y_pred = model.predict(x_train)
auc_score = roc_auc_score(y_train,y_pred)
optimum_split = model.best_params_['min_samples_split']
optimum_depth = model.best_params_['max_depth']

# print('Model with best paramenters - ', model.best_estimator_)
# print('AUC of the model -  ', model.score(x_test,y_test))
# print('Model with best paramenters - ', model.best_params_)
```

In [31]:

```
plt.plot(cv_error, label = 'cv_error')
plt.plot(train_error, label = 'train_error')

# plt.xlabel('Depth')
# plt.ylabel('')
plt.legend()
plt.show()
```

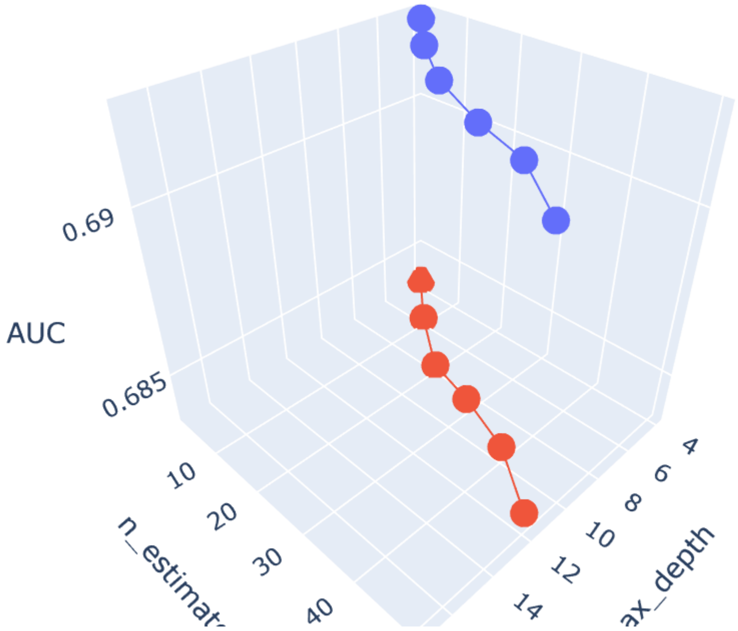


Representation of results

In [32]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [33]:
```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=depth,y=sample_split,z=train_error, name = 'train')
trace2 = go.Scatter3d(x=depth,y=sample_split,z=cv_error, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        yaxis = dict(title='n_estimators'),
        xaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

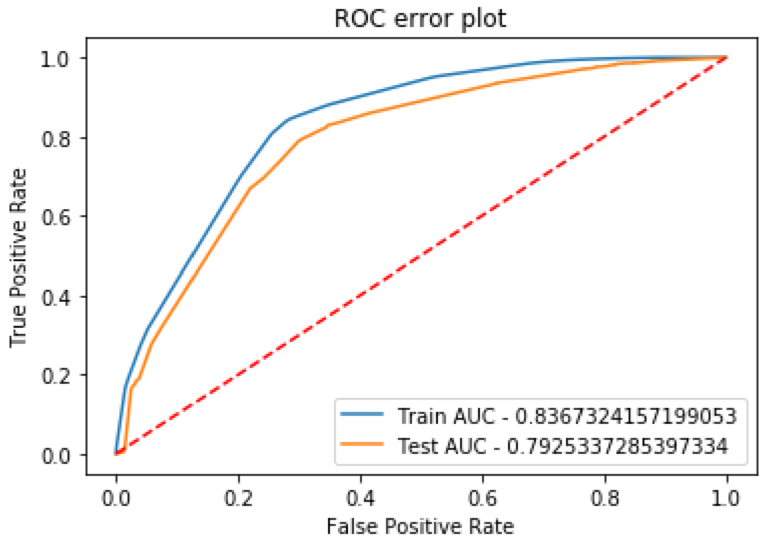


Testing with Test data

In [34]:

```
print('Best value of max_depth : ', optimum_depth )
print('Best value of min_samples_split :',optimum_split )
print('Roc Score on best hyper parameter :',auc_score)

model = DecisionTreeClassifier(max_depth = optimum_depth, min_samples_split = optimum
_split)
model.fit(x_train,y_train)

train_prob = model.predict_proba(x_train)[:,1]
test_prob = model.predict_proba(x_test)[:,1]

train_fpr,train_tpr, tresholds1 =  metrics.roc_curve(y_train,train_prob)
test_fpr,test_tpr,tresholds2 = metrics.roc_curve(y_test,test_prob)

plt.plot(train_fpr,train_tpr,label = 'Train AUC - ' + str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr, label = 'Test AUC - ' + str(auc(test_fpr,test_tpr)))

plt.title('ROC error plot')

plt.legend(loc = 'lower right')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot([0, 1], [0, 1],'r--')

plt.legend()
plt.show()
```

```
Best value of max_depth :  17
Best value of min_samples_split : 50
Roc Score on best hyper parameter : 0.6562003499755487
```

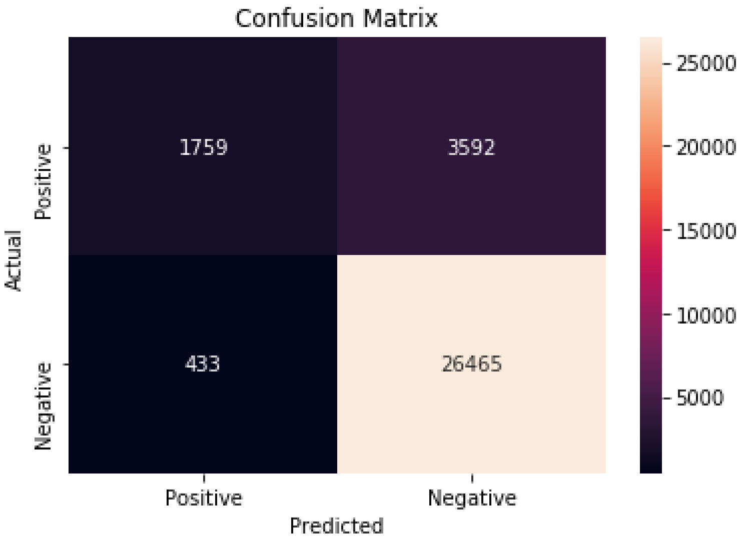


Confusion Matrix using Heatmap

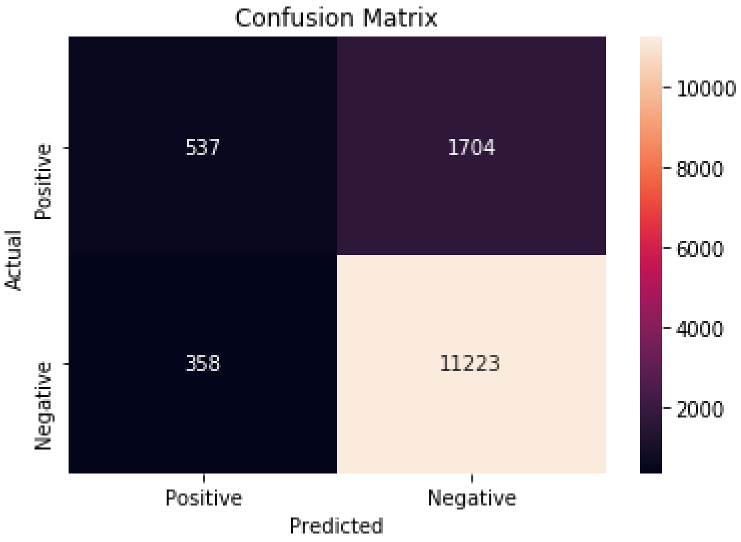
In [35]:

```
#confusion matrix using heatmap for train data
print('Confusion matrix of train data')
cm = confusion_matrix(y_train,model.predict(x_train))
class_labels = ['Positive','Negative']
df = pd.DataFrame(cm, index= class_labels, columns= class_labels)
sb.heatmap(df, annot= True, fmt = 'd')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

#confusion matrix using heatmap for test data
print('Confusion matrix of test data')
cm = confusion_matrix(y_test,model.predict(x_test))
class_labels = ['Positive','Negative']
df = pd.DataFrame(cm,index= class_labels, columns= class_labels)
sb.heatmap(df, annot= True, fmt = 'd')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Confusion matrix of train data
```



```
Confusion matrix of test data
```



[5.1.1] Top 20 important features from SET 1

In [36]:

```
fn = bow_cnt_vect.get_feature_names()
fi = model.feature_importances_

features = np.argsort(fi)[::-1]

for i in features[0:20]:
    print(fn[i])
```

```
not
disappointed
great
worst
awful
best
love
delicious
waste
return
good
terrible
horrible
bad
disappointing
perfect
threw
refund
nice
stale
```

[5.1.2] Graphviz visualization of Decision Tree on BOW, SET 1

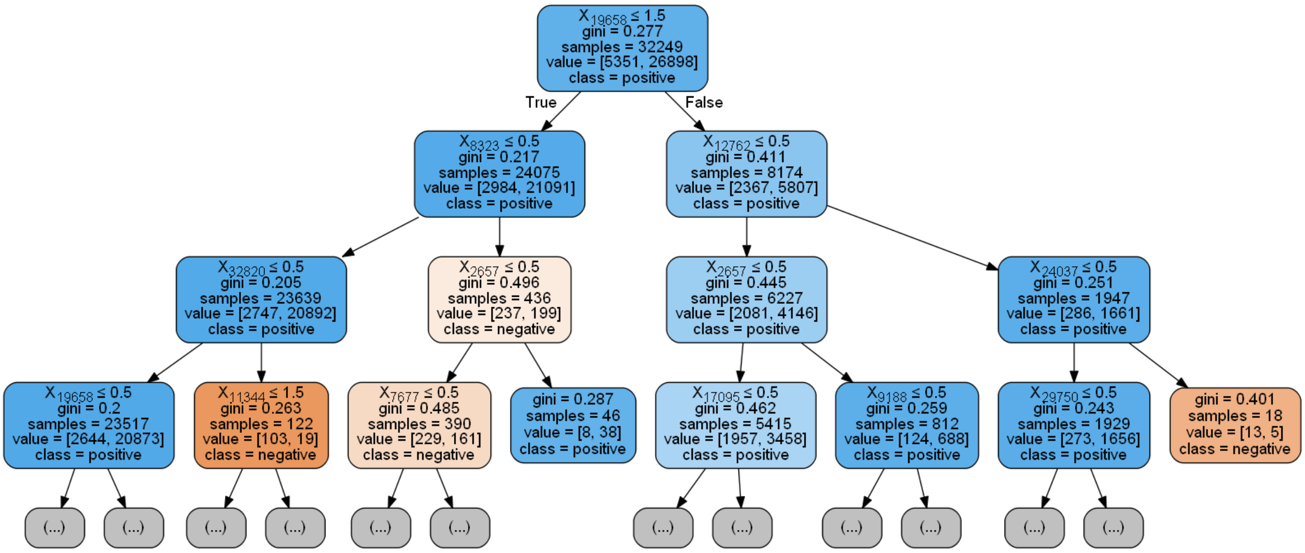
In [37]:

```
# # Please write all the code with proper documentation
# #Tree output in pdf format
# from sklearn import tree
# import graphviz

# target = ['negative','positive']
# model = DecisionTreeClassifier(max_depth = 3, min_samples_split = optimum_split)
# model.fit(x_train,y_train)

# # dot_data = export_graphviz(model, out_file=None, feature_names=fn)
# dot_data = tree.export_graphviz(model,out_file=None,max_depth= 3,class_names=targe
t,filled=True,rounded=True,special_characters=True)

# graph = graphviz.Source(dot_data)
# graph.render("tree_representation_bow")
```

In [38]:

```
# Show the graph in notebok

from sklearn import tree
import pydotplus
from IPython.display import Image
from IPython.display import SVG
from graphviz import Source
from IPython.display import display

target = ['negative','positive']
# Create DOT data
data = tree.export_graphviz(model,out_file=None,max_depth= 3,class_names=target,fille
d=True,rounded=True,special_characters=True)

# Draw graph
graph = pydotplus.graph_from_dot_data(data)
#graph = Source(data)

# Show graph
Image(graph.create_png())
#display(SVG(graph.pipe(format='svg')))
```

Out[38]:



[5.2] Applying Decision Trees on TFIDF, SET 2

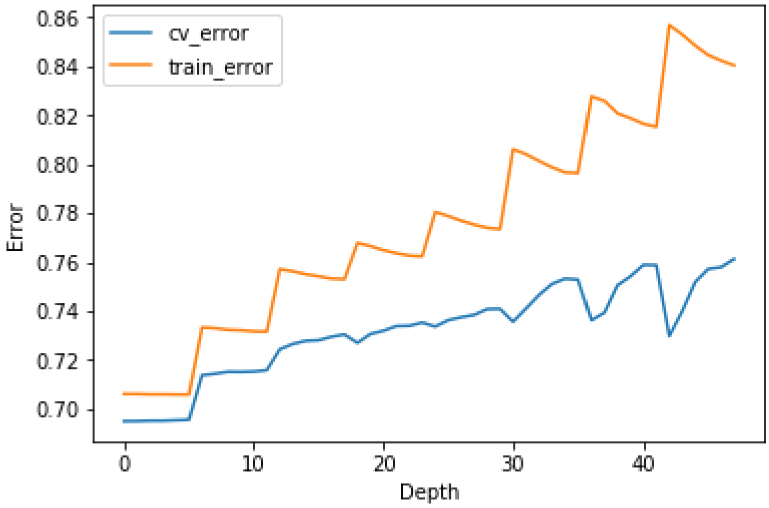
In [39]:

```
tf_idf_vect = TfidfVectorizer()
x_train = tf_idf_vect.fit_transform(base_x_train)
x_test = tf_idf_vect.transform(base_x_test)
```

In [40]:

```
depth = [4,6, 8, 9,10,12,14,17]
sample_split = [2,10,20,30,40,50]

param_grid = { 'max_depth' : depth ,
               'min_samples_split' : sample_split}

model  = GridSearchCV(DecisionTreeClassifier(), param_grid, scoring='roc_auc', cv =3
, n_jobs= -1, pre_dispatch= 2 ,return_train_score=True)
model.fit(x_train,y_train)
cv_error = model.cv_results_['mean_test_score']
train_error = model.cv_results_['mean_train_score']
y_pred = model.predict(x_train)
auc_score = roc_auc_score(y_train,y_pred)
optimum_split = model.best_params_['min_samples_split']
optimum_depth = model.best_params_['max_depth']
```

In [41]:
```
plt.plot(cv_error, label = 'cv_error')
plt.plot(train_error, label = 'train_error')

plt.xlabel('Depth')
plt.ylabel('Error')
plt.legend()
plt.show()
```

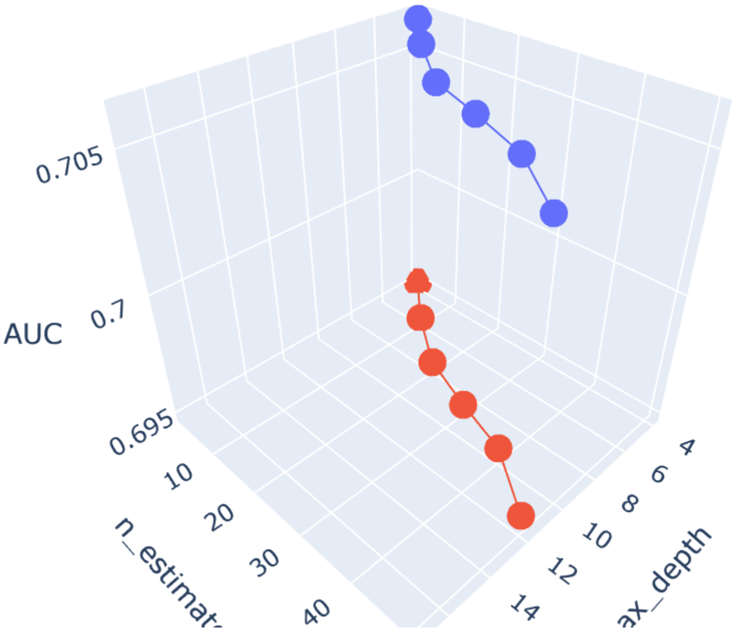


Representation of results

In [42]:
```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [43]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=depth,y=sample_split,z=train_error, name = 'train')
trace2 = go.Scatter3d(x=depth,y=sample_split,z=cv_error, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        yaxis = dict(title='n_estimators'),
        xaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

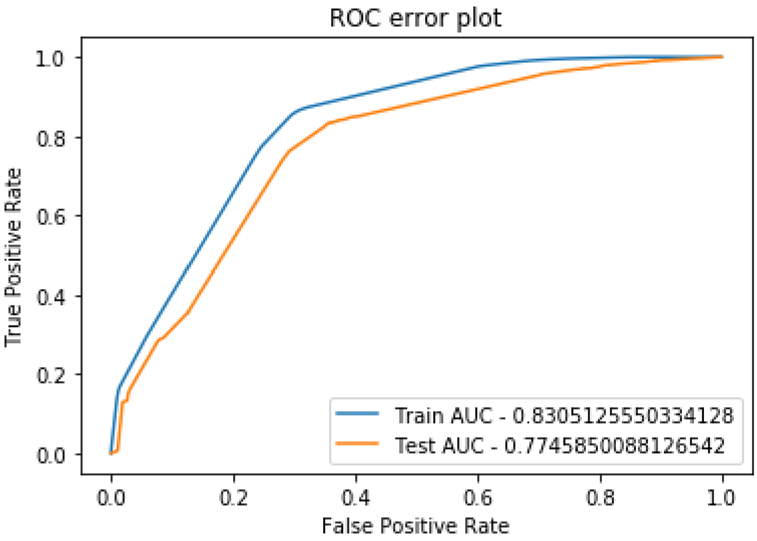


Testing with Test data

In [44]:

```
print('Best value of max_depth : ', optimum_depth )
print('Best value of min_samples_split :',optimum_split )
print('Roc Score on best hyper parameter :',auc_score)

model = DecisionTreeClassifier(max_depth = optimum_depth, min_samples_split = optimum
_split)
model.fit(x_train,y_train)

train_prob = model.predict_proba(x_train)[:,1]
test_prob = model.predict_proba(x_test)[:,1]

train_fpr,train_tpr, tresholds1 =  metrics.roc_curve(y_train,train_prob)
test_fpr,test_tpr,tresholds2 = metrics.roc_curve(y_test,test_prob)

plt.plot(train_fpr,train_tpr,label = 'Train AUC - ' + str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr, label = 'Test AUC - ' + str(auc(test_fpr,test_tpr)))

plt.title('ROC error plot')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

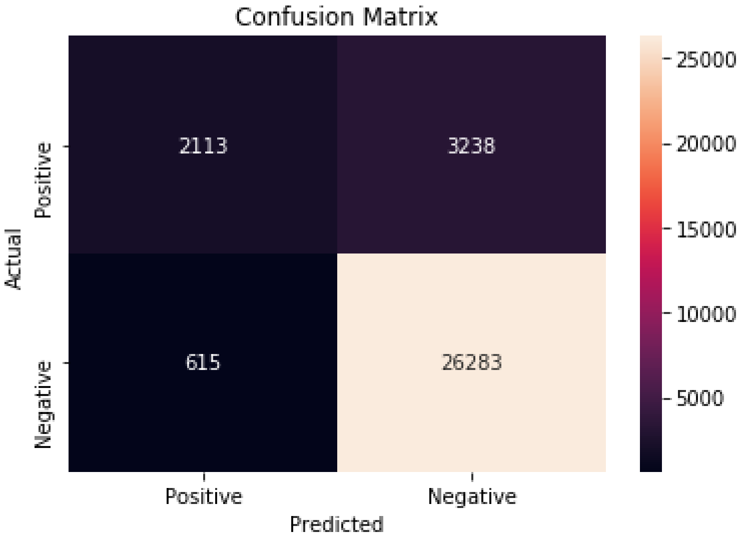
```
Best value of max_depth :  17
Best value of min_samples_split : 50
Roc Score on best hyper parameter : 0.6857459214645041
```



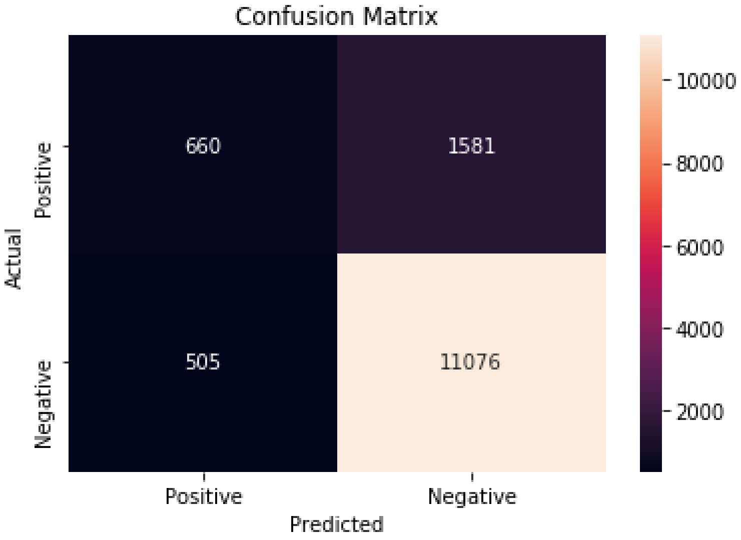
Confusion Matrix using Heatmap

```
In [45]: #confusion matrix using heatmap for train data
         print('Confusion matrix of train data')
         cm = confusion_matrix(y_train,model.predict(x_train))
         class_labels = ['Positive','Negative']
         df = pd.DataFrame(cm, index= class_labels, columns= class_labels)
         sb.heatmap(df, annot= True, fmt = 'd')

         plt.title('Confusion Matrix')
         plt.xlabel('Predicted')
         plt.ylabel('Actual')
         plt.show()

         #confusion matrix using heatmap for test data
         print('Confusion matrix of test data')
         cm = confusion_matrix(y_test,model.predict(x_test))
         class_labels = ['Positive','Negative']
         df = pd.DataFrame(cm,index= class_labels, columns= class_labels)
         sb.heatmap(df, annot= True, fmt = 'd')

         plt.title('Confusion Matrix')
         plt.xlabel('Predicted')
         plt.ylabel('Actual')
         plt.show()
```

```
Confusion matrix of train data
```



```
Confusion matrix of test data
```



[5.2.1] Top 20 important features from SET 2

In [46]:

```
fn = tf_idf_vect.get_feature_names()
fi = model.feature_importances_

features = np.argsort(fi)[::-1]

for i in features[0:20]:
    print(fn[i])
```

```
not
disappointed
great
worst
awful
best
love
waste
return
delicious
good
horrible
bad
disappointing
nice
perfect
terrible
threw
poor
loves
```

[5.2.2] Graphviz visualization of Decision Tree on TFIDF, SET 2

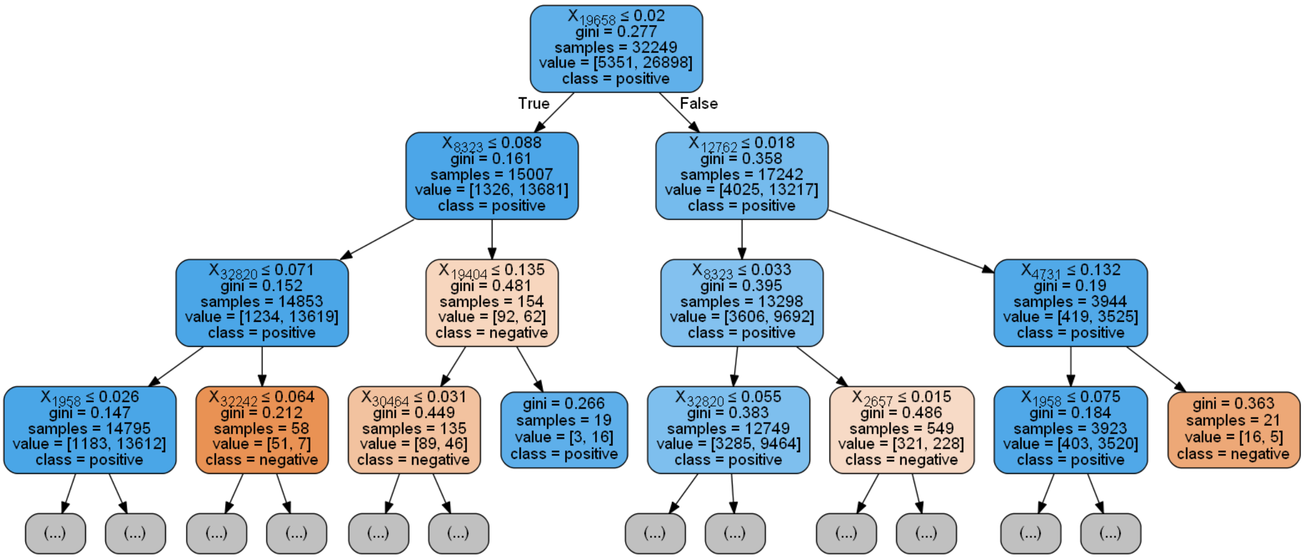
In [47]:

```
# # Please write all the code with proper documentation
# #Tree output in pdf format
# from sklearn import tree
# import graphviz

# model = DecisionTreeClassifier(max_depth = 3, min_samples_split = optimum_split)
# model.fit(x_train,y_train)

# target = ['negative','positive']
# # dot_data = export_graphviz(model, out_file=None, feature_names=fn)
# dot_data = tree.export_graphviz(model,out_file=None,max_depth= 3,class_names=targe
t,filled=True,rounded=True,special_characters=True)

# graph = graphviz.Source(dot_data)
# graph.render("tree_representation_tfidf")
```

In [48]:

```python
# Importing libraries
from sklearn import tree
import pydotplus
from IPython.display import Image
from IPython.display import SVG
from graphviz import Source
from IPython.display import display

target = ['negative','positive']
# Create DOT data
data = tree.export_graphviz(model,out_file=None,max_depth= 3,class_names=target,filled=True,rounded=True,special_characters=True)

# Draw graph
graph = pydotplus.graph_from_dot_data(data)
#graph = Source(data)

# Show graph
Image(graph.create_png())
#display(SVG(graph.pipe(format='svg')))
```

Out[48]:



[5.3] Applying Decision Trees on AVG W2V, SET 3

Training w2v model

In [49]:

```
# w2v for train

list_of_sentance_train = []
for sentance in base_x_train:
    list_of_sentance_train.append(sentance.split())
#training w2v model
w2v_model = Word2Vec(list_of_sentance_train, min_count= 5, size = 50, workers = 4 )
w2v_words = list(w2v_model.wv.vocab)

# Converting Train data text

sent_vectors = []
for sent in list_of_sentance_train :
    sent_vec = np.zeros(50)
    cnt_vec = 0
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec+= vec
            cnt_vec +=1
    if cnt_words != 0:
        sent_vec/= cnt_words
    sent_vectors.append(sent_vec)
sent_vectors_train = np.array(sent_vectors)
print(sent_vectors_train.shape)
```

```
(32249, 50)
```

In [50]:

```
#for test data
list_of_sentance_test = []
for sentance in base_x_test:
    list_of_sentance_test.append(sentance.split())

# Converting Train data text

sent_vectors_test = []
for sent in list_of_sentance_test :
    sent_vec = np.zeros(50)
    cnt_vec = 0
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec+= vec
            cnt_vec +=1
    if cnt_words != 0:
        sent_vec/= cnt_words
    sent_vectors_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
```

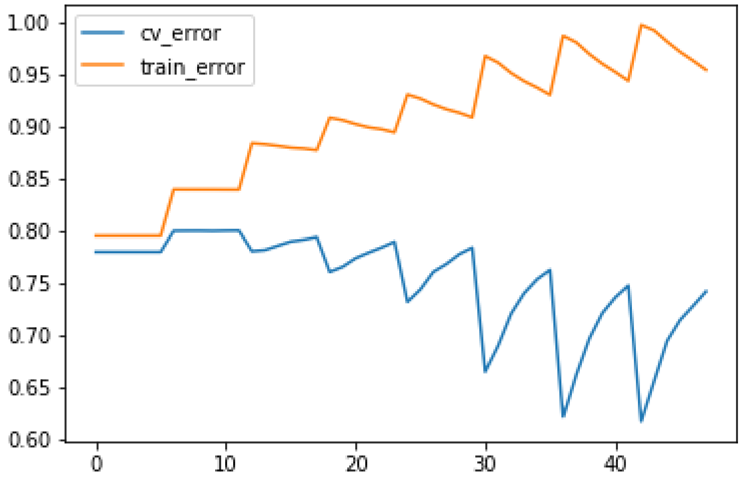
```
(13822, 50)
```

In [51]:

```
x_train = sent_vectors_train
x_test = sent_vectors_test

depth = [4,6, 8, 9,10,12,14,17]
sample_split = [2,10,20,30,40,50]

param_grid = { 'max_depth' : depth ,'min_samples_split' : sample_split}

model  = GridSearchCV(DecisionTreeClassifier(), param_grid, scoring='roc_auc', cv =3
, n_jobs= -1, pre_dispatch= 2 ,return_train_score=True)
model.fit(x_train,y_train)
cv_error = model.cv_results_['mean_test_score']
train_error = model.cv_results_['mean_train_score']
y_pred = model.predict(x_train)
auc_score = roc_auc_score(y_train,y_pred)
optimum_split = model.best_params_['min_samples_split']
optimum_depth = model.best_params_['max_depth']
```

In [52]:

```
plt.plot(cv_error, label = 'cv_error')
plt.plot(train_error, label = 'train_error')

# plt.xlabel('Depth')
# plt.ylabel('')
plt.legend()
plt.show()
```



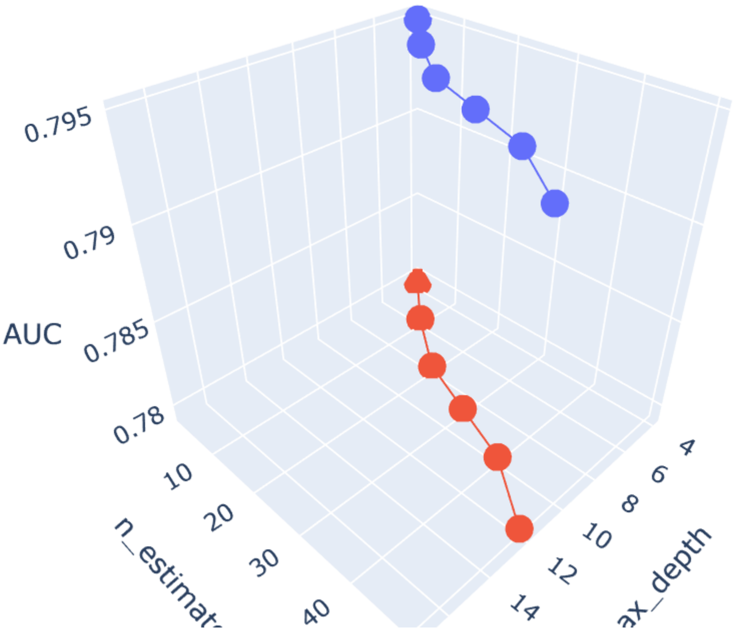
Representation of results

In [53]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [54]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=depth,y=sample_split,z=train_error, name = 'train')
trace2 = go.Scatter3d(x=depth,y=sample_split,z=cv_error, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        yaxis = dict(title='n_estimators'),
        xaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

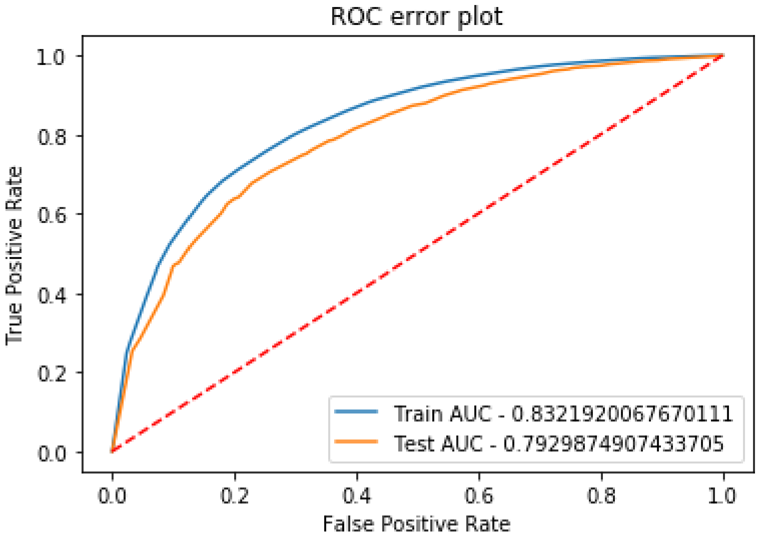


Testing with Test data

In [55]:

```
print('Best value of max_depth : ', optimum_depth )
print('Best value of min_samples_split :',optimum_split )
print('Roc Score on best hyper parameter :',auc_score)

model = DecisionTreeClassifier(max_depth = optimum_depth, min_samples_split = optimum
_split)
model.fit(x_train,y_train)

train_prob = model.predict_proba(x_train)[:,1]
test_prob = model.predict_proba(x_test)[:,1]

train_fpr,train_tpr, tresholds1 =  metrics.roc_curve(y_train,train_prob)
test_fpr,test_tpr,tresholds2 = metrics.roc_curve(y_test,test_prob)

plt.plot(train_fpr,train_tpr,label = 'Train AUC - ' + str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr, label = 'Test AUC - ' + str(auc(test_fpr,test_tpr)))

plt.title('ROC error plot')

plt.legend(loc = 'lower right')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot([0, 1], [0, 1],'r--')

plt.legend()
plt.show()
```

```
Best value of max_depth :  6
Best value of min_samples_split : 50
Roc Score on best hyper parameter : 0.6423923116376757
```

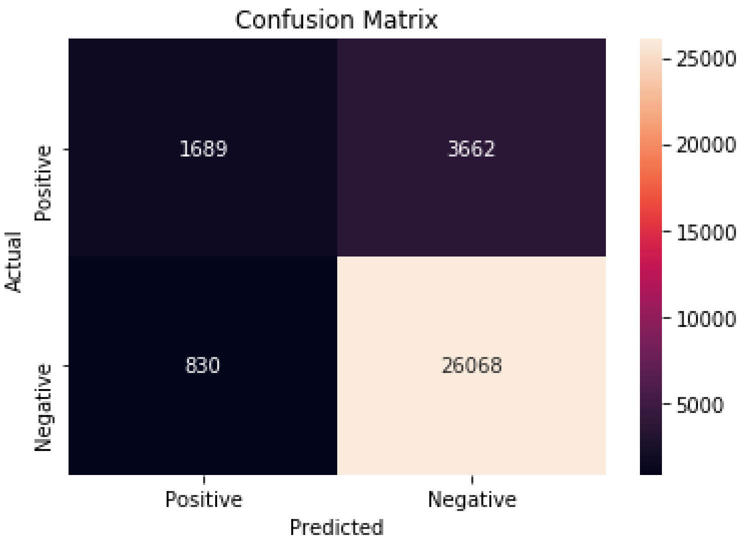


Confusion Matrix using Heatmap

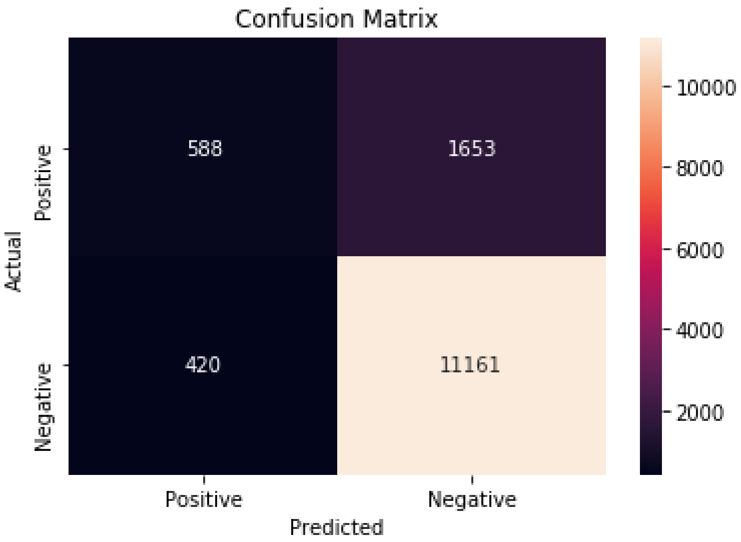
In [56]:

```
#confusion matrix using heatmap for train data
print('Confusion matrix of train data')
cm = confusion_matrix(y_train,model.predict(x_train))
class_labels = ['Positive','Negative']
df = pd.DataFrame(cm, index= class_labels, columns= class_labels)
sb.heatmap(df, annot= True, fmt = 'd')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

#confusion matrix using heatmap for test data
print('Confusion matrix of test data')
cm = confusion_matrix(y_test,model.predict(x_test))
class_labels = ['Positive','Negative']
df = pd.DataFrame(cm,index= class_labels, columns= class_labels)
sb.heatmap(df, annot= True, fmt = 'd')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Confusion matrix of train data
```



```
Confusion matrix of test data
```



[5.4] Applying Decision Trees on TFIDF W2V, SET 4

In [57]:

```
# w2v for train

list_of_sentance_train = []
for sentance in base_x_train:
    list_of_sentance_train.append(sentance.split())

w2v_model = Word2Vec(list_of_sentance_train , min_count = 5 ,size = 50, workers = 4)
w2v_words = list(w2v_model.wv.vocab)

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df= 10,max_features= 500)
tf_idf_matrix = tf_idf_vect.fit_transform(base_x_train)
tfidf_feat = tf_idf_vect.get_feature_names()
dictionary = dict(zip(tf_idf_vect.get_feature_names(),list(tf_idf_vect.idf_)))
```

In [58]:

```
# Converting Train data text

tfidf_sent_vectors_train = []
for sent in list_of_sentance_train :
    sent_vec = np.zeros(50)
    weight_sum = 0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec+= ( vec * tf_idf )
            weight_sum = tf_idf
    if weight_sum != 0:
        sent_vec/= weight_sum
    tfidf_sent_vectors_train.append(sent_vec)
    row +=1
```

In [59]:

```
#for test data

list_of_sentance_test = []
for sentance in base_x_test:
    list_of_sentance_test.append(sentance.split())

tfidf_sent_vectors_test = []
row = 0

# for sent in tqdm(list_of_sentance_test):
for sent in (list_of_sentance_test):

    sent_vec = np.zeros(50)
    weight_sum = 0
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum = tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

In [60]:

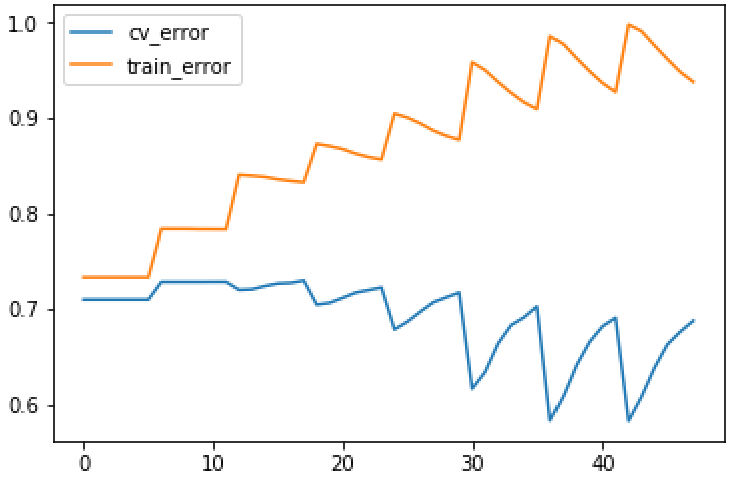
```
x_train = tfidf_sent_vectors_train
x_test = tfidf_sent_vectors_test
```

In [61]:

```
depth = [4,6, 8, 9,10,12,14,17]
sample_split = [2,10,20,30,40,50]

param_grid = { 'max_depth' : depth ,'min_samples_split' : sample_split}

model  = GridSearchCV(DecisionTreeClassifier(), param_grid, scoring='roc_auc', cv =3
, n_jobs= -1, pre_dispatch= 2 ,return_train_score=True)
model.fit(x_train,y_train)
cv_error = model.cv_results_['mean_test_score']
train_error = model.cv_results_['mean_train_score']
y_pred = model.predict(x_train)
auc_score = roc_auc_score(y_train,y_pred)
optimum_split = model.best_params_['min_samples_split']
optimum_depth = model.best_params_['max_depth']
```

In [62]:

```
plt.plot(cv_error, label = 'cv_error')
plt.plot(train_error, label = 'train_error')

# plt.xlabel('Depth')
# plt.ylabel('')
plt.legend()
plt.show()
```



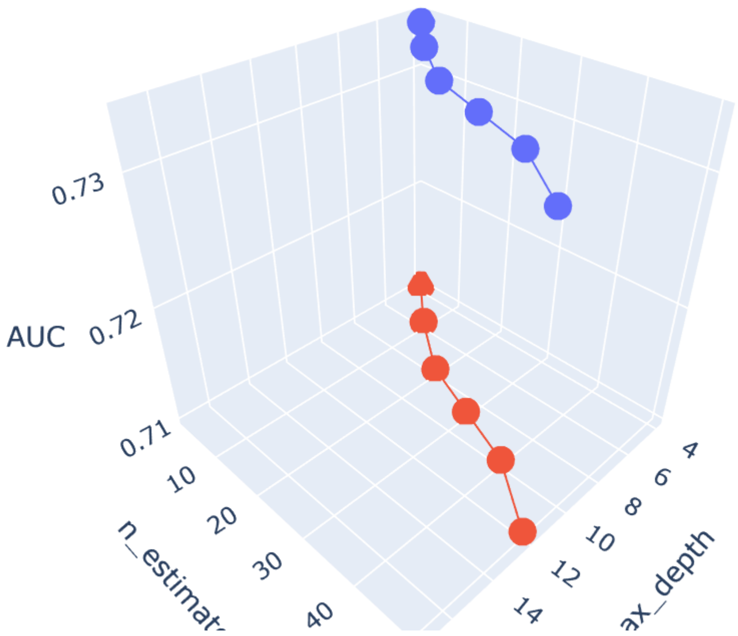
Representation of results

In [63]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [64]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=depth,y=sample_split,z=train_error, name = 'train')
trace2 = go.Scatter3d(x=depth,y=sample_split,z=cv_error, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        yaxis = dict(title='n_estimators'),
        xaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

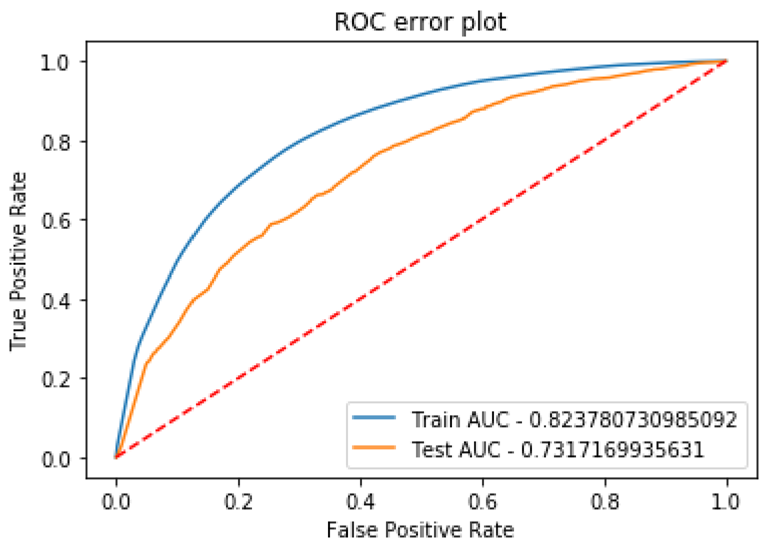


Testing with Test data

In [65]:

```
print('Best value of max_depth : ', optimum_depth )
print('Best value of min_samples_split :',optimum_split )
print('Roc Score on best hyper parameter :',auc_score)

model = DecisionTreeClassifier(max_depth = optimum_depth, min_samples_split = optimum
_split)
model.fit(x_train,y_train)

train_prob = model.predict_proba(x_train)[:,1]
test_prob = model.predict_proba(x_test)[:,1]

train_fpr,train_tpr, tresholds1 =  metrics.roc_curve(y_train,train_prob)
test_fpr,test_tpr,tresholds2 = metrics.roc_curve(y_test,test_prob)

plt.plot(train_fpr,train_tpr,label = 'Train AUC - ' + str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr, label = 'Test AUC - ' + str(auc(test_fpr,test_tpr)))

plt.title('ROC error plot')

plt.legend(loc = 'lower right')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot([0, 1], [0, 1],'r--')

plt.legend()
plt.show()
```

```
Best value of max_depth :  8
Best value of min_samples_split : 50
Roc Score on best hyper parameter : 0.6452803130284512
```

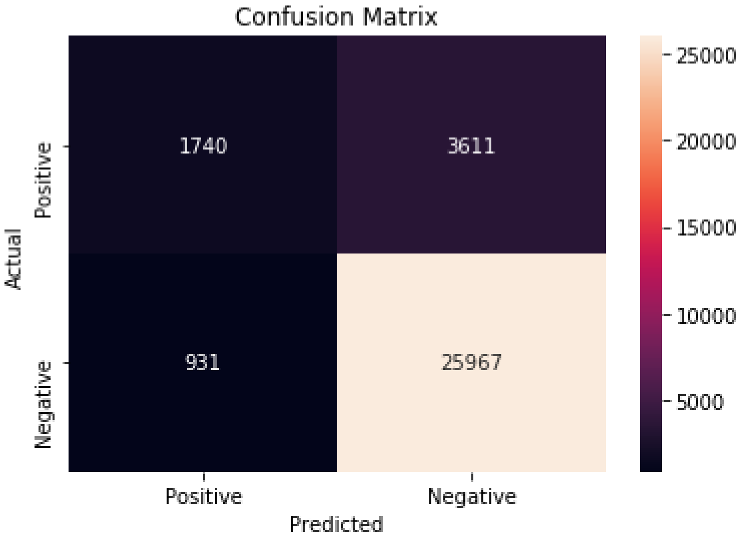


Confusion Matrix using Heatmap

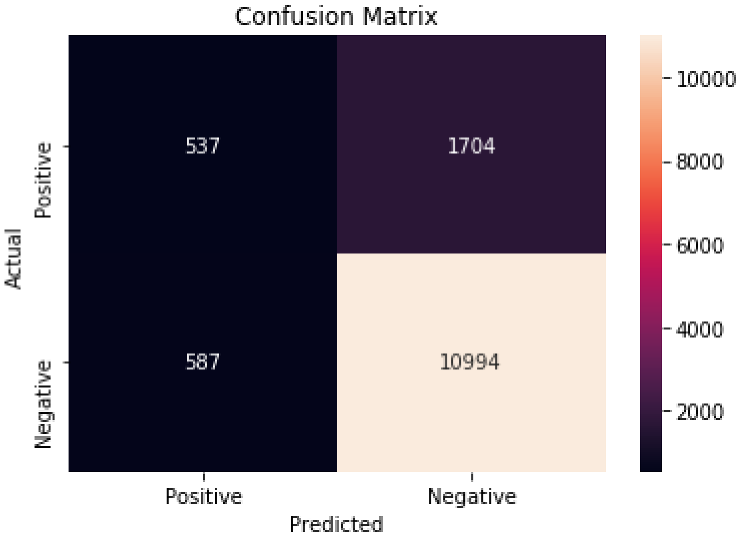
In [66]:

```
#confusion matrix using heatmap for train data
print('Confusion matrix of train data')
cm = confusion_matrix(y_train,model.predict(x_train))
class_labels = ['Positive','Negative']
df = pd.DataFrame(cm, index= class_labels, columns= class_labels)
sb.heatmap(df, annot= True, fmt = 'd')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

#confusion matrix using heatmap for test data
print('Confusion matrix of test data')
cm = confusion_matrix(y_test,model.predict(x_test))
class_labels = ['Positive','Negative']
df = pd.DataFrame(cm,index= class_labels, columns= class_labels)
sb.heatmap(df, annot= True, fmt = 'd')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Confusion matrix of train data
```



```
Confusion matrix of test data
```



[5.5] Feature engineering

In [67]:

```
preprocessed_reviews[100]
```

Out[67]:

```
'fyi customers item beef ocean fish formula red bag haste purchased thinking version
chicken rice formula woops went bought bag chicken rice mix beef fish not wreak havo
c pup digestive system say started feeding pup starting stinky farts never also rosh
an right fish breath ick overall dog no issues formula stinky stick chicken rice bag
done'
```

In [68]:

```
#Adding preprocessed summary and review length to preprocessed summary

for i in range(len(preprocessed_reviews)):
    preprocessed_reviews[i] += ' '+preprocessed_summary[i]+' '+str(len(final.Text.iloc[i]))

preprocessed_reviews[100]
```

Out[68]:

```
'fyi customers item beef ocean fish formula red bag haste purchased thinking version
chicken rice formula woops went bought bag chicken rice mix beef fish not wreak havo
c pup digestive system say started feeding pup starting stinky farts never also rosh
an right fish breath ick overall dog no issues formula stinky stick chicken rice bag
done wrong bag pictured 661'
```

In [69]:

```
x = preprocessed_reviews
y = final['Score'].values


#Train CV test split


base_x_train, base_x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3,random_state = 0)
```

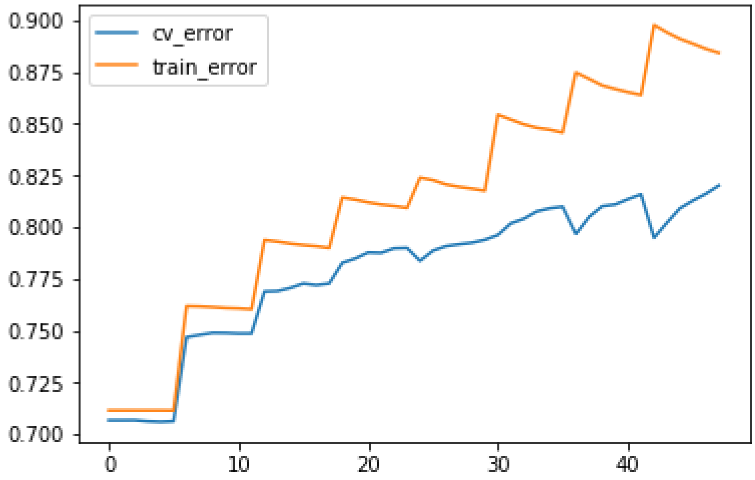
[5.1.2] Applying Decision Trees on BOW, SET 1

In [70]:

```
from sklearn.tree import DecisionTreeClassifier

bow_cnt_vect = CountVectorizer()
x_train = bow_cnt_vect.fit_transform(base_x_train)
x_test = bow_cnt_vect.transform(base_x_test)

depth = [4,6, 8, 9,10,12,14,17]
sample_split = [2,10,20,30,40,50]

param_grid = { 'max_depth' : depth ,
               'min_samples_split' : sample_split}

model  = GridSearchCV(DecisionTreeClassifier(), param_grid, scoring='roc_auc', cv =3
, n_jobs= -1, pre_dispatch= 2 ,return_train_score=True)
model.fit(x_train,y_train)
cv_error = model.cv_results_['mean_test_score']
train_error = model.cv_results_['mean_train_score']
y_pred = model.predict(x_train)
auc_score = roc_auc_score(y_train,y_pred)
optimum_split = model.best_params_['min_samples_split']
optimum_depth = model.best_params_['max_depth']

# print('Model with best paramenters - ', model.best_estimator_)
# print('AUC of the model -  ', model.score(x_test,y_test))
# print('Model with best paramenters - ', model.best_params_)
```

In [71]:

```
plt.plot(cv_error, label = 'cv_error')
plt.plot(train_error, label = 'train_error')

# plt.xlabel('Depth')
# plt.ylabel('')
plt.legend()
plt.show()
```

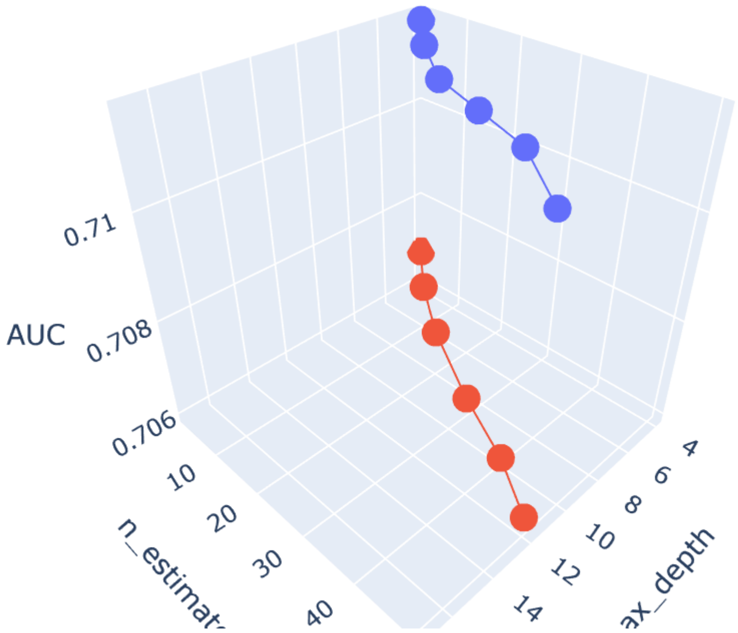


Representation of results

In [72]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [73]:
```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=depth,y=sample_split,z=train_error, name = 'train')
trace2 = go.Scatter3d(x=depth,y=sample_split,z=cv_error, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        yaxis = dict(title='n_estimators'),
        xaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

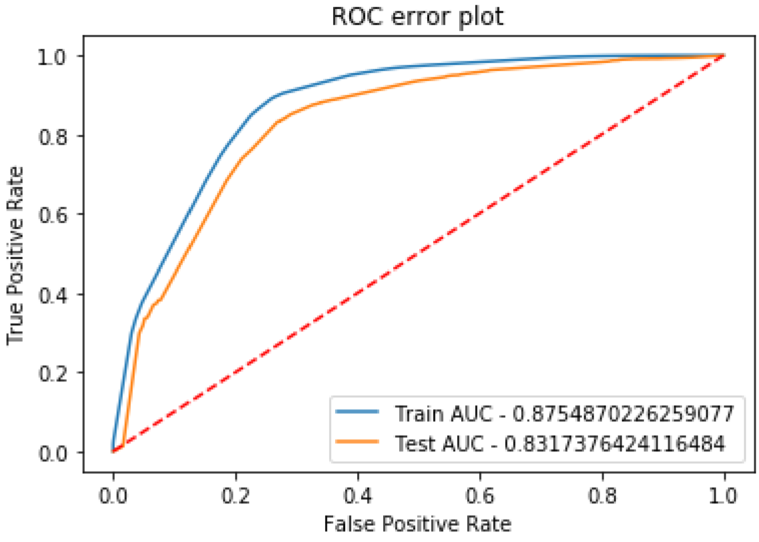


Testing with Test data

In [74]:

```
print('Best value of max_depth : ', optimum_depth )
print('Best value of min_samples_split :',optimum_split )
print('Roc Score on best hyper parameter :',auc_score)

model = DecisionTreeClassifier(max_depth = optimum_depth, min_samples_split = optimum
_split)
model.fit(x_train,y_train)

train_prob = model.predict_proba(x_train)[:,1]
test_prob = model.predict_proba(x_test)[:,1]

train_fpr,train_tpr, tresholds1 =  metrics.roc_curve(y_train,train_prob)
test_fpr,test_tpr,tresholds2 = metrics.roc_curve(y_test,test_prob)

plt.plot(train_fpr,train_tpr,label = 'Train AUC - ' + str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr, label = 'Test AUC - ' + str(auc(test_fpr,test_tpr)))

plt.title('ROC error plot')

plt.legend(loc = 'lower right')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot([0, 1], [0, 1],'r--')

plt.legend()
plt.show()
```

```
Best value of max_depth :  17
Best value of min_samples_split : 50
Roc Score on best hyper parameter : 0.758003726891789
```

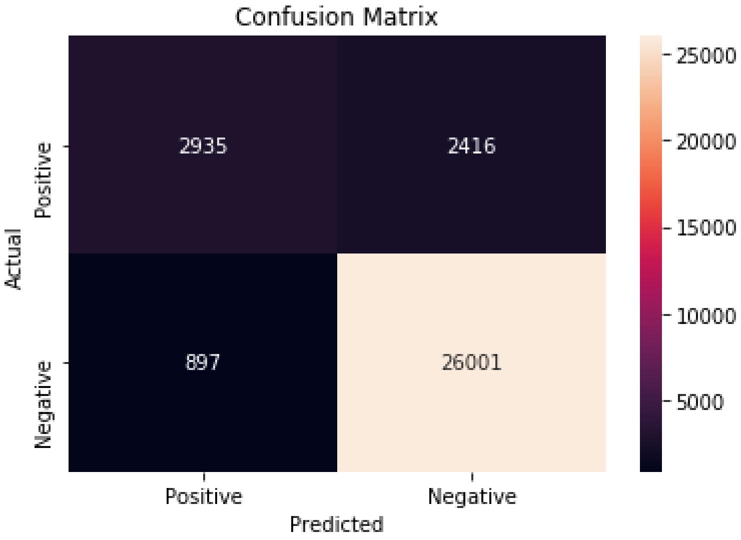


Confusion Matrix using Heatmap

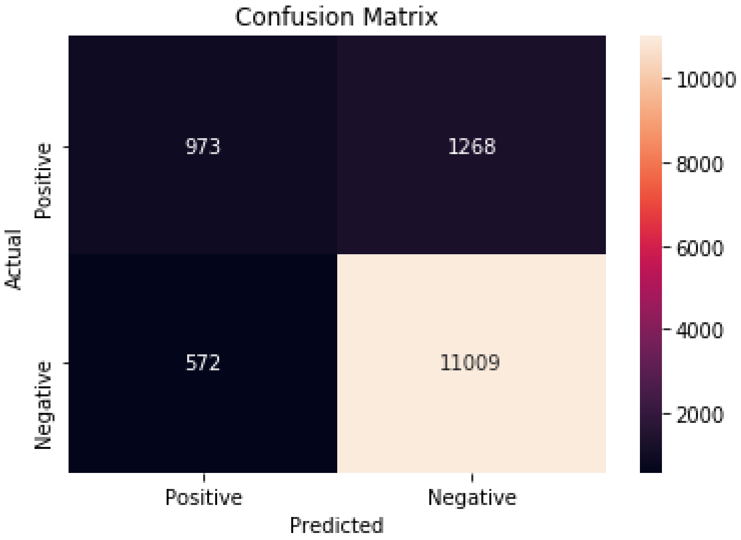
In [75]:

```
#confusion matrix using heatmap for train data
print('Confusion matrix of train data')
cm = confusion_matrix(y_train,model.predict(x_train))
class_labels = ['Positive','Negative']
df = pd.DataFrame(cm, index= class_labels, columns= class_labels)
sb.heatmap(df, annot= True, fmt = 'd')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

#confusion matrix using heatmap for test data
print('Confusion matrix of test data')
cm = confusion_matrix(y_test,model.predict(x_test))
class_labels = ['Positive','Negative']
df = pd.DataFrame(cm,index= class_labels, columns= class_labels)
sb.heatmap(df, annot= True, fmt = 'd')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Confusion matrix of train data
```



```
Confusion matrix of test data
```



[5.1.1] Top 20 important features from SET 1

In [76]:

```
fn = bow_cnt_vect.get_feature_names()
fi = model.feature_importances_

features = np.argsort(fi)[::-1]

for i in features[0:20]:
    print(fn[i])
```

```
not
great
worst
disappointed
best
delicious
good
horrible
love
disappointing
bad
terrible
perfect
tasty
awful
disgusting
loves
waste
nice
excellent
```

[5.2.2] Applying Decision Trees on TFIDF, SET 2

In [77]:

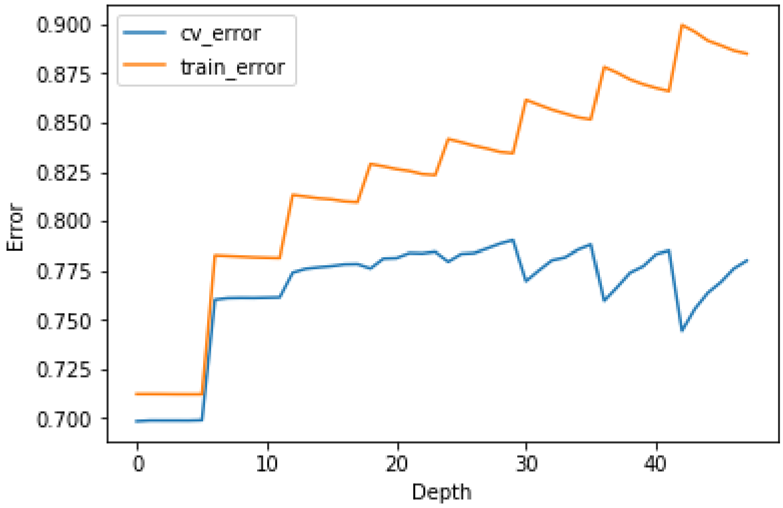
```
tf_idf_vect = TfidfVectorizer()
x_train = tf_idf_vect.fit_transform(base_x_train)
x_test = tf_idf_vect.transform(base_x_test)
```

In [78]:

```
depth = [4,6, 8, 9,10,12,14,17]
sample_split = [2,10,20,30,40,50]

param_grid = { 'max_depth' : depth ,
               'min_samples_split' : sample_split}

model  = GridSearchCV(DecisionTreeClassifier(), param_grid, scoring='roc_auc', cv =3
, n_jobs= -1, pre_dispatch= 2 ,return_train_score=True)
model.fit(x_train,y_train)
cv_error = model.cv_results_['mean_test_score']
train_error = model.cv_results_['mean_train_score']
y_pred = model.predict(x_train)
auc_score = roc_auc_score(y_train,y_pred)
optimum_split = model.best_params_['min_samples_split']
optimum_depth = model.best_params_['max_depth']
```

In [79]:

```
plt.plot(cv_error, label = 'cv_error')
plt.plot(train_error, label = 'train_error')

plt.xlabel('Depth')
plt.ylabel('Error')
plt.legend()
plt.show()
```



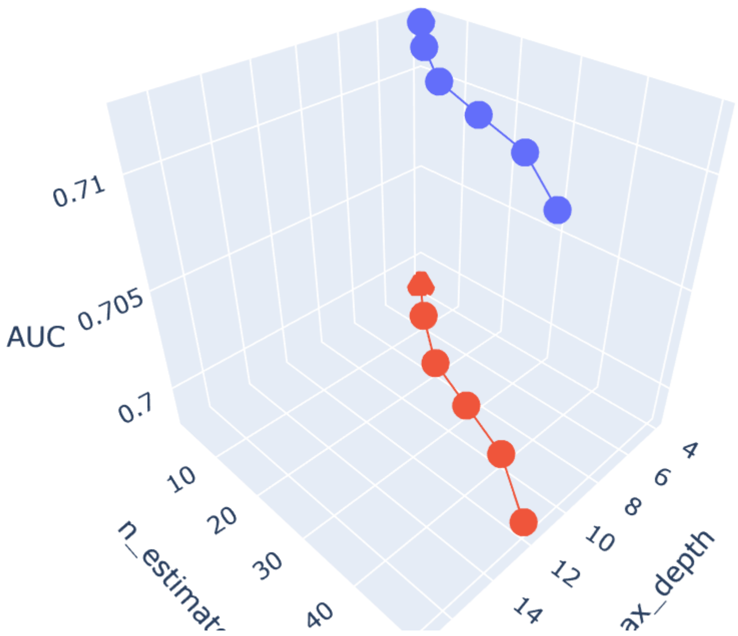
Representation of results

In [80]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [81]:

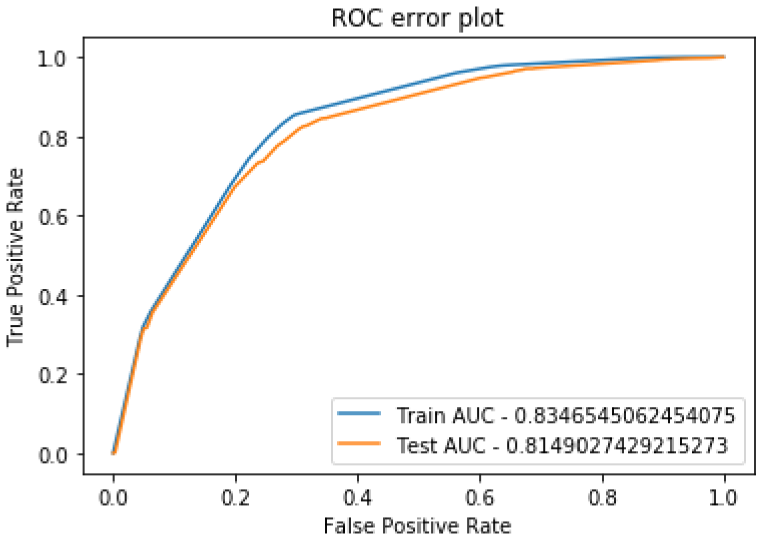
```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=depth,y=sample_split,z=train_error, name = 'train')
trace2 = go.Scatter3d(x=depth,y=sample_split,z=cv_error, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        yaxis = dict(title='n_estimators'),
        xaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



Testing with Test data

```
In [82]: print('Best value of max_depth : ', optimum_depth )
         print('Best value of min_samples_split :',optimum_split )
         print('Roc Score on best hyper parameter :',auc_score)

         model = DecisionTreeClassifier(max_depth = optimum_depth, min_samples_split = optimum
         _split)
         model.fit(x_train,y_train)

         train_prob = model.predict_proba(x_train)[:,1]
         test_prob = model.predict_proba(x_test)[:,1]

         train_fpr,train_tpr, tresholds1 =  metrics.roc_curve(y_train,train_prob)
         test_fpr,test_tpr,tresholds2 = metrics.roc_curve(y_test,test_prob)

         plt.plot(train_fpr,train_tpr,label = 'Train AUC - ' + str(auc(train_fpr,train_tpr)))
         plt.plot(test_fpr,test_tpr, label = 'Test AUC - ' + str(auc(test_fpr,test_tpr)))

         plt.title('ROC error plot')
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.legend()
         plt.show()
```

```
Best value of max_depth :  10
Best value of min_samples_split : 50
Roc Score on best hyper parameter : 0.6741593021410132
```

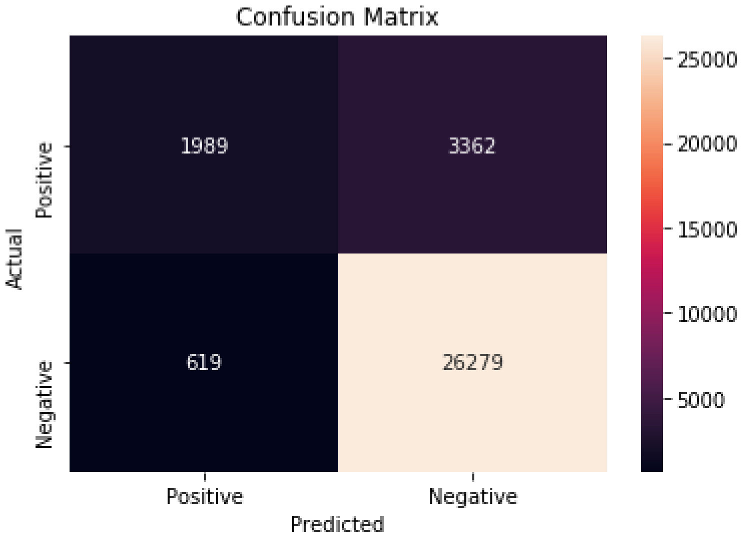


Confusion Matrix using Heatmap

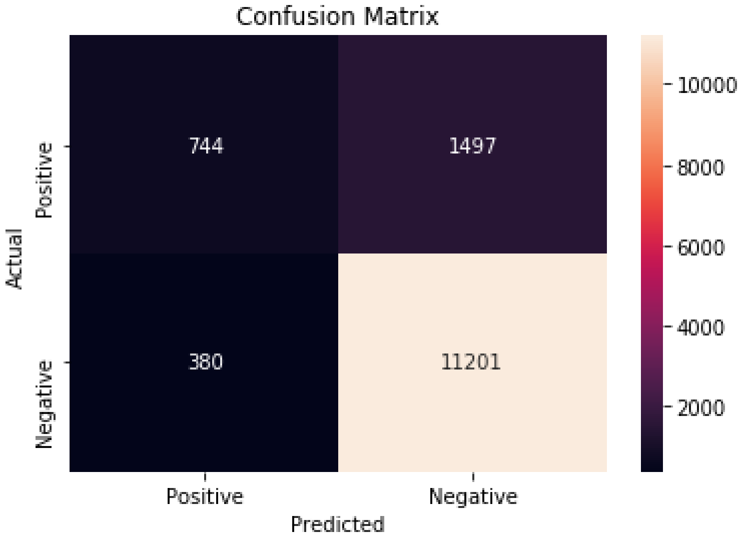
In [83]:

```
#confusion matrix using heatmap for train data
print('Confusion matrix of train data')
cm = confusion_matrix(y_train,model.predict(x_train))
class_labels = ['Positive','Negative']
df = pd.DataFrame(cm, index= class_labels, columns= class_labels)
sb.heatmap(df, annot= True, fmt = 'd')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

#confusion matrix using heatmap for test data
print('Confusion matrix of test data')
cm = confusion_matrix(y_test,model.predict(x_test))
class_labels = ['Positive','Negative']
df = pd.DataFrame(cm,index= class_labels, columns= class_labels)
sb.heatmap(df, annot= True, fmt = 'd')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Confusion matrix of train data
```



```
Confusion matrix of test data
```



[5.2.1] Top 20 important features from SET 2

In [84]:

```
fn = tf_idf_vect.get_feature_names()
fi = model.feature_importances_

features = np.argsort(fi)[::-1]

for i in features[0:20]:
    print(fn[i])
```

```
not
great
disappointed
worst
awful
best
delicious
good
love
bad
disappointing
horrible
tasty
favorite
loves
stale
yuck
changed
nice
return
```

[5.3.2] Applying Decision Trees on AVG W2V, SET 3

Training w2v model

In [85]:

```
# w2v for train

list_of_sentance_train = []
for sentance in base_x_train:
    list_of_sentance_train.append(sentance.split())
#training w2v model
w2v_model = Word2Vec(list_of_sentance_train, min_count= 5, size = 50, workers = 4 )
w2v_words = list(w2v_model.wv.vocab)

# Converting Train data text

sent_vectors = []
for sent in list_of_sentance_train :
    sent_vec = np.zeros(50)
    cnt_vec = 0
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec+= vec
            cnt_vec +=1
    if cnt_words != 0:
        sent_vec/= cnt_words
    sent_vectors.append(sent_vec)
sent_vectors_train = np.array(sent_vectors)
print(sent_vectors_train.shape)
```

```
(32249, 50)
```

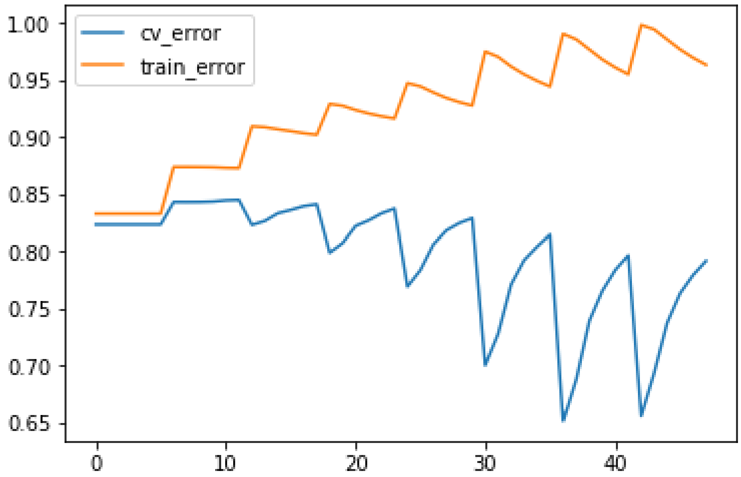
In [86]:

```
#for test data
list_of_sentance_test = []
for sentance in base_x_test:
    list_of_sentance_test.append(sentance.split())

# Converting Train data text

sent_vectors_test = []
for sent in list_of_sentance_test :
    sent_vec = np.zeros(50)
    cnt_vec = 0
    for word in sent:
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec+= vec
            cnt_vec +=1
    if cnt_words != 0:
        sent_vec/= cnt_words
    sent_vectors_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
```

```
(13822, 50)
```

```
In [87]: x_train = sent_vectors_train
         x_test = sent_vectors_test

         depth = [4,6, 8, 9,10,12,14,17]
         sample_split = [2,10,20,30,40,50]

         param_grid = { 'max_depth' : depth ,'min_samples_split' : sample_split}

         model  = GridSearchCV(DecisionTreeClassifier(), param_grid, scoring='roc_auc', cv =3
         , n_jobs= -1, pre_dispatch= 2 ,return_train_score=True)
         model.fit(x_train,y_train)
         cv_error = model.cv_results_['mean_test_score']
         train_error = model.cv_results_['mean_train_score']
         y_pred = model.predict(x_train)
         auc_score = roc_auc_score(y_train,y_pred)
         optimum_split = model.best_params_['min_samples_split']
         optimum_depth = model.best_params_['max_depth']
```

```
In [88]: plt.plot(cv_error, label = 'cv_error')
         plt.plot(train_error, label = 'train_error')

         # plt.xlabel('Depth')
         # plt.ylabel('')
         plt.legend()
         plt.show()
```

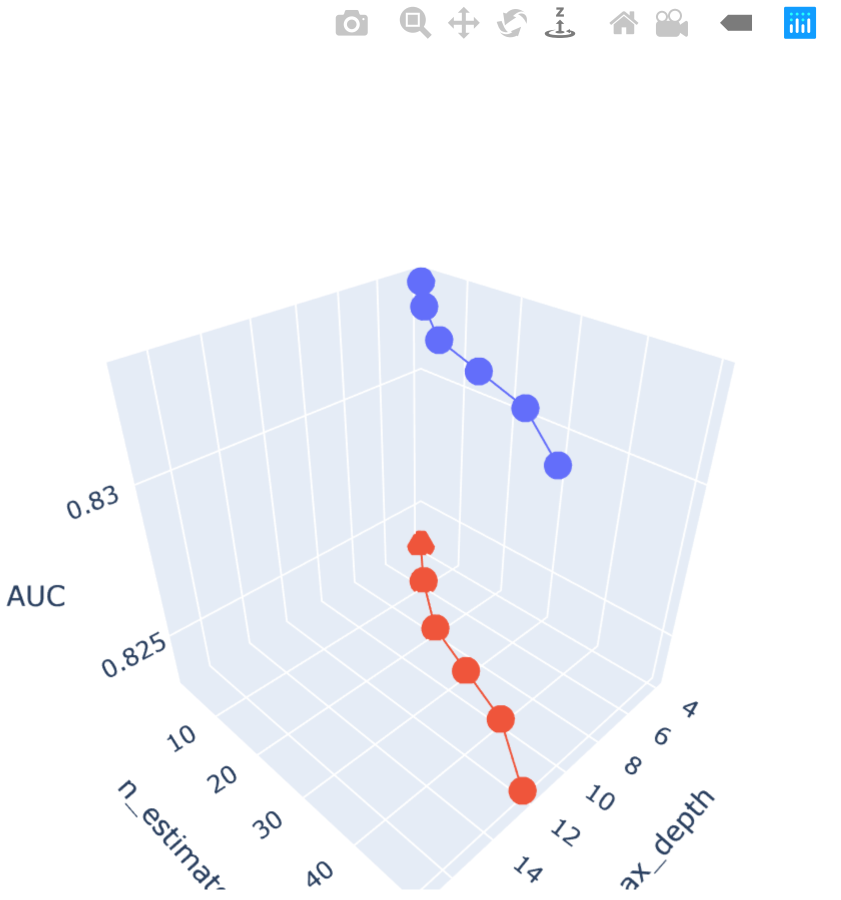


Representation of results

```
In [89]: import plotly.offline as offline
         import plotly.graph_objs as go
         offline.init_notebook_mode()
         import numpy as np
```

In [90]:

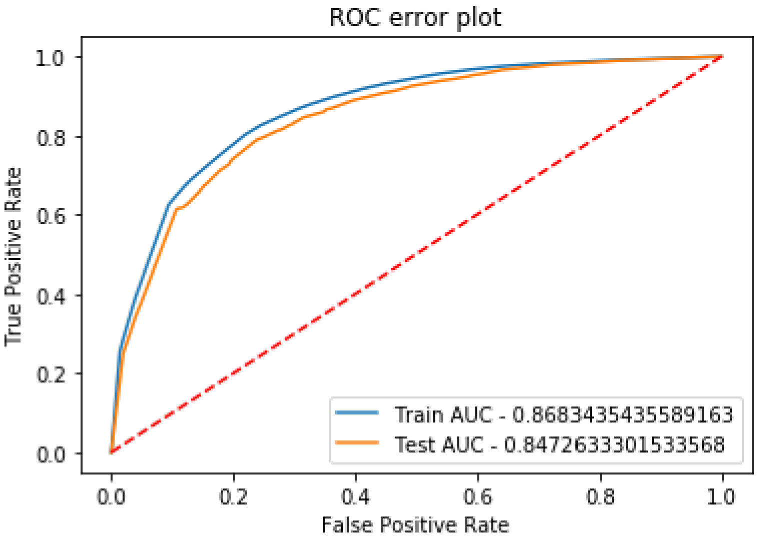
```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=depth,y=sample_split,z=train_error, name = 'train')
trace2 = go.Scatter3d(x=depth,y=sample_split,z=cv_error, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        yaxis = dict(title='n_estimators'),
        xaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



Testing with Test data

In [91]:
```
print('Best value of max_depth : ', optimum_depth )
print('Best value of min_samples_split :',optimum_split )
print('Roc Score on best hyper parameter :',auc_score)

model = DecisionTreeClassifier(max_depth = optimum_depth, min_samples_split = optimum
_split)
model.fit(x_train,y_train)

train_prob = model.predict_proba(x_train)[:,1]
test_prob = model.predict_proba(x_test)[:,1]

train_fpr,train_tpr, tresholds1 =  metrics.roc_curve(y_train,train_prob)
test_fpr,test_tpr,tresholds2 = metrics.roc_curve(y_test,test_prob)

plt.plot(train_fpr,train_tpr,label = 'Train AUC - ' + str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr, label = 'Test AUC - ' + str(auc(test_fpr,test_tpr)))

plt.title('ROC error plot')

plt.legend(loc = 'lower right')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot([0, 1], [0, 1],'r--')

plt.legend()
plt.show()
```

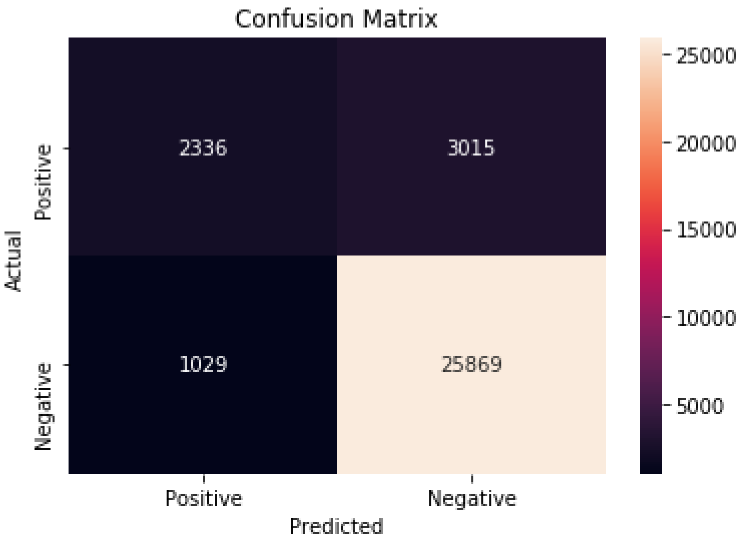
```
Best value of max_depth :  6
Best value of min_samples_split : 50
Roc Score on best hyper parameter : 0.6991491413835103
```



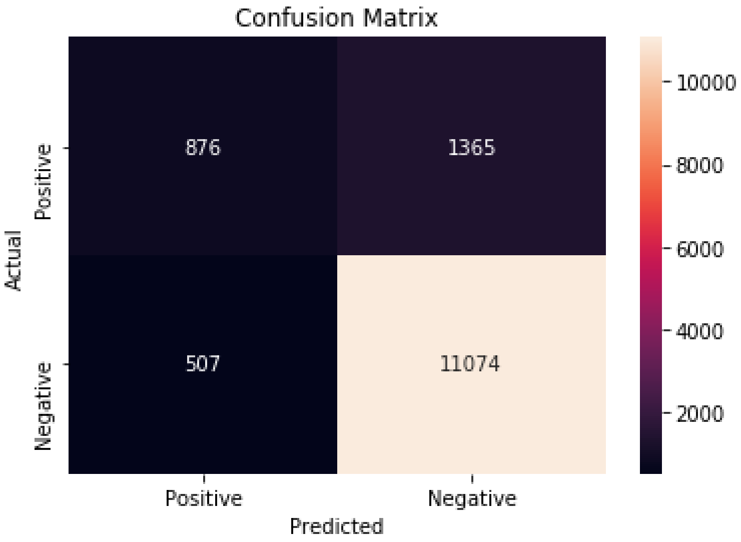
Confusion Matrix using Heatmap

```
In [92]: #confusion matrix using heatmap for train data
         print('Confusion matrix of train data')
         cm = confusion_matrix(y_train,model.predict(x_train))
         class_labels = ['Positive','Negative']
         df = pd.DataFrame(cm, index= class_labels, columns= class_labels)
         sb.heatmap(df, annot= True, fmt = 'd')

         plt.title('Confusion Matrix')
         plt.xlabel('Predicted')
         plt.ylabel('Actual')
         plt.show()

         #confusion matrix using heatmap for test data
         print('Confusion matrix of test data')
         cm = confusion_matrix(y_test,model.predict(x_test))
         class_labels = ['Positive','Negative']
         df = pd.DataFrame(cm,index= class_labels, columns= class_labels)
         sb.heatmap(df, annot= True, fmt = 'd')

         plt.title('Confusion Matrix')
         plt.xlabel('Predicted')
         plt.ylabel('Actual')
         plt.show()
```

```
Confusion matrix of train data
```



```
Confusion matrix of test data
```



[5.4.2] Applying Decision Trees on TFIDF W2V, SET 4

In [93]:

```
# w2v for train

list_of_sentance_train = []
for sentance in base_x_train:
    list_of_sentance_train.append(sentance.split())

w2v_model = Word2Vec(list_of_sentance_train , min_count = 5 ,size = 50, workers = 4)
w2v_words = list(w2v_model.wv.vocab)

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df= 10,max_features= 500)
tf_idf_matrix = tf_idf_vect.fit_transform(base_x_train)
tfidf_feat = tf_idf_vect.get_feature_names()
dictionary = dict(zip(tf_idf_vect.get_feature_names(),list(tf_idf_vect.idf_)))
```

In [94]:

```
# Converting Train data text

tfidf_sent_vectors_train = []
for sent in list_of_sentance_train :
    sent_vec = np.zeros(50)
    weight_sum = 0;
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec+= ( vec * tf_idf )
            weight_sum = tf_idf
    if weight_sum != 0:
        sent_vec/= weight_sum
    tfidf_sent_vectors_train.append(sent_vec)
    row +=1
```

In [95]:

```
#for test data

list_of_sentance_test = []
for sentance in base_x_test:
    list_of_sentance_test.append(sentance.split())

tfidf_sent_vectors_test = []
row = 0

# for sent in tqdm(list_of_sentance_test):
for sent in (list_of_sentance_test):

    sent_vec = np.zeros(50)
    weight_sum = 0
    for word in sent:
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum = tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

In [96]:

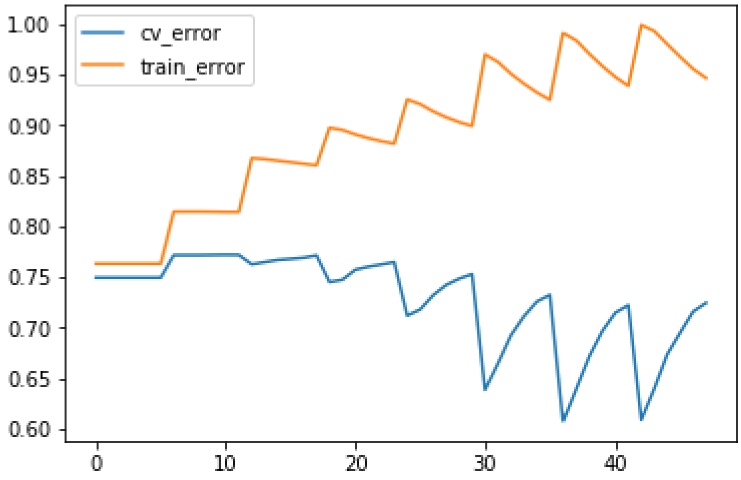
```
x_train = tfidf_sent_vectors_train
x_test = tfidf_sent_vectors_test
```

In [97]:

```
depth = [4,6, 8, 9,10,12,14,17]
sample_split = [2,10,20,30,40,50]

param_grid = { 'max_depth' : depth ,'min_samples_split' : sample_split}

model  = GridSearchCV(DecisionTreeClassifier(), param_grid, scoring='roc_auc', cv =3
, n_jobs= -1, pre_dispatch= 2 ,return_train_score=True)
model.fit(x_train,y_train)
cv_error = model.cv_results_['mean_test_score']
train_error = model.cv_results_['mean_train_score']
y_pred = model.predict(x_train)
auc_score = roc_auc_score(y_train,y_pred)
optimum_split = model.best_params_['min_samples_split']
optimum_depth = model.best_params_['max_depth']
```

In [98]:

```
plt.plot(cv_error, label = 'cv_error')
plt.plot(train_error, label = 'train_error')

# plt.xlabel('Depth')
# plt.ylabel('')
plt.legend()
plt.show()
```



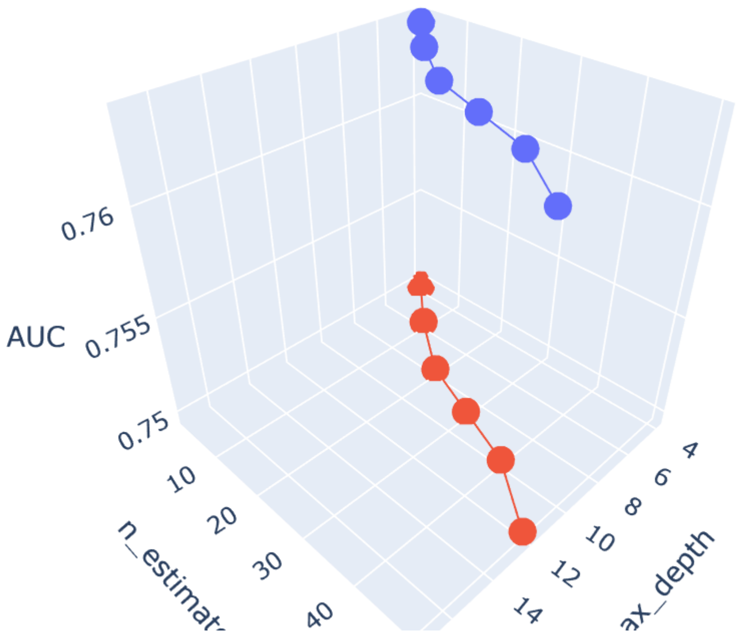
Representation of results

In [99]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [100]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=depth,y=sample_split,z=train_error, name = 'train')
trace2 = go.Scatter3d(x=depth,y=sample_split,z=cv_error, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        yaxis = dict(title='n_estimators'),
        xaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

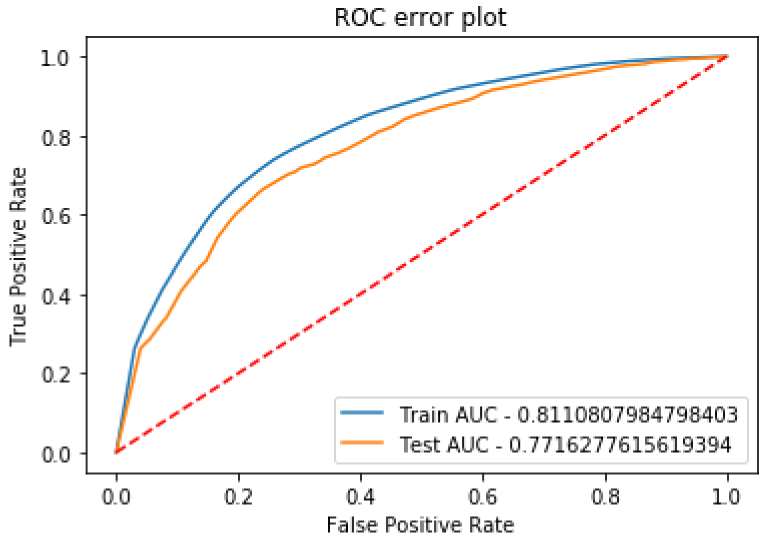


Testing with Test data

In [101]:

```
print('Best value of max_depth : ', optimum_depth )
print('Best value of min_samples_split :',optimum_split )
print('Roc Score on best hyper parameter :',auc_score)

model = DecisionTreeClassifier(max_depth = optimum_depth, min_samples_split = optimum
_split)
model.fit(x_train,y_train)

train_prob = model.predict_proba(x_train)[:,1]
test_prob = model.predict_proba(x_test)[:,1]

train_fpr,train_tpr, tresholds1 =  metrics.roc_curve(y_train,train_prob)
test_fpr,test_tpr,tresholds2 = metrics.roc_curve(y_test,test_prob)

plt.plot(train_fpr,train_tpr,label = 'Train AUC - ' + str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr, label = 'Test AUC - ' + str(auc(test_fpr,test_tpr)))

plt.title('ROC error plot')

plt.legend(loc = 'lower right')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot([0, 1], [0, 1],'r--')

plt.legend()
plt.show()
```

```
Best value of max_depth :  6
Best value of min_samples_split : 40
Roc Score on best hyper parameter : 0.600129893311942
```

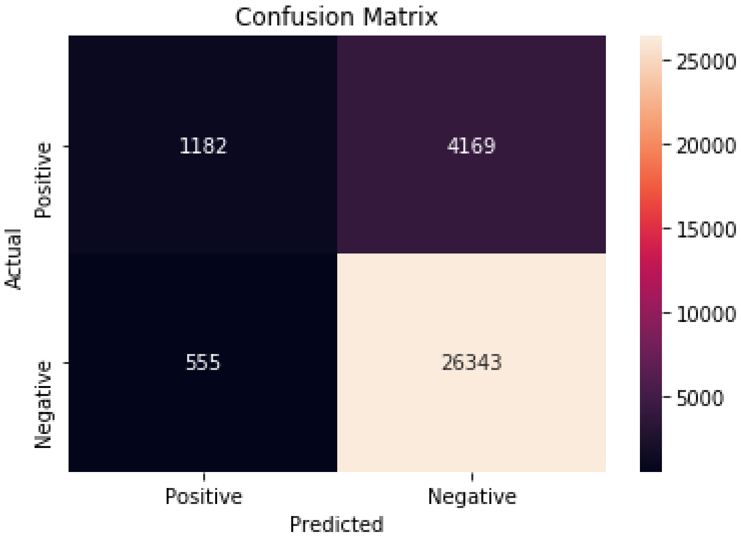


Confusion Matrix using Heatmap

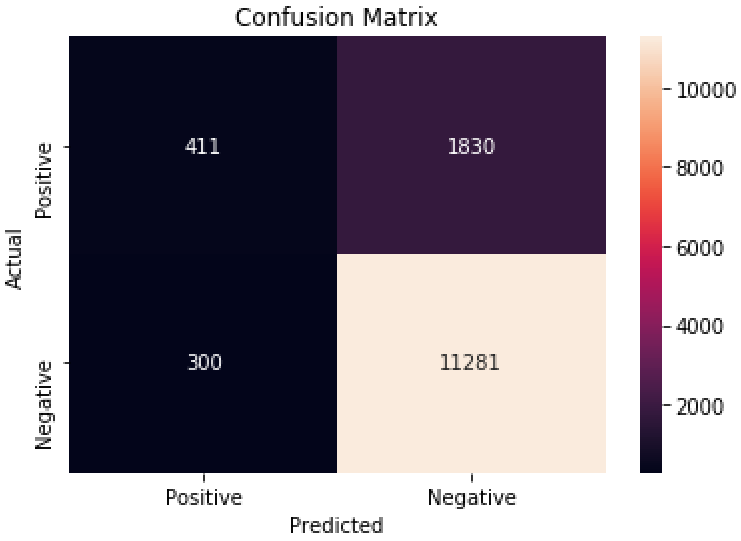
In [102]:

```
#confusion matrix using heatmap for train data
print('Confusion matrix of train data')
cm = confusion_matrix(y_train,model.predict(x_train))
class_labels = ['Positive','Negative']
df = pd.DataFrame(cm, index= class_labels, columns= class_labels)
sb.heatmap(df, annot= True, fmt = 'd')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

#confusion matrix using heatmap for test data
print('Confusion matrix of test data')
cm = confusion_matrix(y_test,model.predict(x_test))
class_labels = ['Positive','Negative']
df = pd.DataFrame(cm,index= class_labels, columns= class_labels)
sb.heatmap(df, annot= True, fmt = 'd')

plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Confusion matrix of train data
```



```
Confusion matrix of test data
```



[6] Conclusions

In [103]:

```python
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer","Best depth", "Best min_samples_split", "Feature engineering","AUC"]
x.add_row(['BOW','17','50','Not featured','0.79'])
x.add_row(['TFIDF','17','50','Not featured','0.77'])
x.add_row(['AVG W2vec','6','50','Not featured','0.80'])
x.add_row(['TFIDF W2vec','6','20','Not featured','0.74'])
x.add_row(['BOW','17','50','featured','0.83'])
x.add_row(['TFIDF','10','50','featured','0.81'])
x.add_row(['AVG W2vec','6','50','featured','0.81'])
x.add_row(['TFIDF W2vec','6','30','featured','0.76'])
print(x)
```

```
+-------------+------------+-----------------------+---------------------+------+
|  Vectorizer | Best depth | Best min_samples_split | Feature engineering | AUC  |
+-------------+------------+-----------------------+---------------------+------+
|     BOW     |     17     |           50          |     Not featured    | 0.79 |
|    TFIDF    |     17     |           50          |     Not featured    | 0.77 |
|  AVG W2vec  |     6      |           50          |     Not featured    | 0.80 |
| TFIDF W2vec |     6      |           20          |     Not featured    | 0.74 |
|     BOW     |     17     |           50          |       featured      | 0.83 |
|    TFIDF    |     10     |           50          |       featured      | 0.81 |
|  AVG W2vec  |     6      |           50          |       featured      | 0.81 |
| TFIDF W2vec |     6      |           30          |       featured      | 0.76 |
+-------------+------------+-----------------------+---------------------+------+
```

In [ ]: