09 Amazon Fine Food Reviews Analysis_RF

August 10, 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")
        from bs4 import BeautifulSoup


        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/
        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

        from sklearn.ensemble import RandomForestClassifier
        import xgboost as xgb
        from sklearn import tree
        import pydotplus
        from IPython.display import Image
        from IPython.display import SVG
        from graphviz import Source
        import graphviz
        from IPython.display import display
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from prettytable import PrettyTable
from wordcloud import WordCloud
from sklearn.externals import joblib
from datetime import datetime
from xgboost import XGBClassifier
```

```
C:\Users\hp\Anaconda3\lib\site-packages\sklearn\externals\joblib\__init__.py:15: DeprecationWa
  warnings.warn(msg, category=DeprecationWarning)
```

In [2]:
```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data 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 50
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 1000

# 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 (100000, 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]:
```python
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [4]:
```python
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]:
```python
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]:
```python
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=Fal
```

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

```
Out[9]: (87775, 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]: 87.775
```

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

```
In [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()
```

```
(87773, 10)
```

Out[13]:
```
1    73592
0    14181
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 [0]:
```
# 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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br />http://www.amazon.c
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the oth
==================================================
love to order my coffee on amazon.  easy and shows up quickly.<br />This k cup is great coffee
==================================================
```

In [0]:
```
# 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)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victor
```

In [0]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-


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)
```

```
Why is this $[...] when the same product is available for $[...] here? />The Victor M380 and M5
==================================================
I recently tried this flavor/brand and was surprised at how delicious these chips are.  The bes
==================================================
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the oth
==================================================
```

```
love to order my coffee on amazon.  easy and shows up quickly.This k cup is great coffee.  dca
```

```
In [14]: # 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 [0]: sent_1500 = decontracted(sent_1500)
        print(sent_1500)
        print("="*50)
```

```
Wow.  So far, two two-star reviews.  One obviously had no idea what they were ordering; the oth
==================================================
```

```
In [0]: #remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
        sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
        print(sent_0)
```

```
Why is this $[...] when the same product is available for $[...] here?<br /> /><br />The Victo
```

```
In [0]: #remove spacial character: https://stackoverflow.com/a/5843547/4084039
        sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
        print(sent_1500)
```

```
Wow So far two two star reviews One obviously had no idea what they were ordering the other wa
```

```
In [15]: # 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', '
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
            '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 [16]:
```
# 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 stopwo
    preprocessed_reviews.append(sentance.strip())
```

```
100%|| 87773/87773 [00:34<00:00, 2517.72it/s]
```

In [221]:
```
preprocessed_reviews[1500]
```

Out[221]:
```
'favorite stevia product subscribe save queried customer service nunaturals gmo use y
```

5 [4] Featurization

Before we apply various featurizations to the data we need to split the data appropriately as train data, cross validation data and test data

In [17]:
```
x = preprocessed_reviews
y = final["Score"].values
```

In [18]:
```
# splitting the data into 3 parts for furhter process,
# train data, cross validation data and test data
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30)         # th
x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, test_size=0.30) # th
```

```
In [19]: # number of rows in earch data set, train, cross validation and test data respectivel
         print(len(x_train))
         print(len(x_cv))
         print(len(x_test))
```

```
43008
18433
26332
```

5.1 [4.1] BAG OF WORDS

```
In [20]: #BoW
         count_vect = CountVectorizer(max_features=5000) #in scikit-learn
         count_vect.fit(x_train)
         print("some feature names ", count_vect.get_feature_names()[:10])
         print('='*50)

         x_train_bow = count_vect.transform(x_train)
         x_test_bow = count_vect.transform(x_test)
         x_cv_bow = count_vect.transform(x_cv)

         print(x_train_bow.shape, y_train.shape)
         print(x_cv_bow.shape, y_cv.shape)
         print(x_test_bow.shape, y_test.shape)
```

```
some feature names  ['ability', 'able', 'absolute', 'absolutely', 'absorb', 'absorbed', 'acai'
==================================================
(43008, 5000) (43008,)
(18433, 5000) (18433,)
(26332, 5000) (26332,)
```

5.2 [4.2] TF-IDF

```
In [22]: # TFIDF using scikit-learn

         tf_idf = TfidfVectorizer(max_features=5000, dtype=int) #arguments: ngram_range=(1,2),
         tf_idf.fit(x_train)

         print("some sample features",tf_idf.get_feature_names()[0:10])
         print('='*50)

         x_train_tf = tf_idf.transform(x_train)
         x_cv_tf    = tf_idf.transform(x_cv)
```

```
x_test_tf  = tf_idf.transform(x_test)

print("After featurization\n")

print(x_train_tf.shape, y_train.shape)
print(x_cv_tf.shape, y_cv.shape)
print(x_test_tf.shape, y_test.shape)
```

```
some sample features ['ability', 'able', 'absolute', 'absolutely', 'absorb', 'absorbed', 'acai
==================================================
After featurization

(43008, 5000) (43008,)
(18433, 5000) (18433,)
(26332, 5000) (26332,)
```

5.3 [4.3] Word2Vec

In [227]:
```
# Train your own Word2Vec model using your own text corpus
list_of_sentance_train =[]

for sentance in x_train:
    list_of_sentance_train.append(sentance.split())
```

In [228]:
```
# this line of code trains your w2v model on the give list of sentances
w2v_model = Word2Vec(list_of_sentance_train, min_count=5, size=50, workers=-1)
```

In [229]:
```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  7075
sample words  ['eating', 'fruit', 'years', 'recently', 'added', 'maltodextrin', 'check', 'new'
```

5.4 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v Converting Train data

In [230]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this li
for sent in tqdm(list_of_sentance_train): # 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_train.append(sent_vec)
    sent_vectors_train = np.array(sent_vectors_train)
    print(sent_vectors_train.shape)
    print(sent_vectors_train[0])
```

```
100%|| 13755/13755 [00:17<00:00, 803.27it/s]


(13755, 50)
[ 2.08400891e-04 -9.54966694e-05  1.05930530e-03 -9.86759152e-04
  1.03871688e-03 -7.95648566e-04 -8.83831157e-04  1.80738693e-03
 -5.75234919e-04 -6.88477222e-04 -5.77197190e-04  5.88051477e-04
  1.85811609e-04 -1.05708501e-03 -1.87508407e-03 -1.14425059e-03
  4.71166263e-04 -1.84508484e-04 -1.86353642e-03  6.82333575e-05
 -5.87147159e-05 -4.54119609e-04  3.10231787e-04 -1.26688662e-03
  1.15916298e-03 -2.93292250e-04 -1.41279414e-03 -5.22746093e-04
  5.34880144e-04 -1.12079959e-03 -1.86780546e-04  1.06564831e-03
 -2.18338521e-04  1.25704851e-03 -2.04193813e-03  7.33595160e-04
 -2.25365151e-04 -2.60276241e-04 -1.69951864e-04 -8.34980092e-04
  2.92323443e-04 -1.46243181e-04 -8.50822371e-04 -1.98671771e-04
 -8.19005312e-04  6.45404169e-04 -1.70866120e-03 -1.76786259e-04
  4.06971635e-04  1.89786434e-03]
```

Converting cross validation data

In [231]:
```
list_of_sentance_cv=[]
for sentance in x_cv:
    list_of_sentance_cv.append(sentance.split())
```

In [232]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_cv): # 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_cv.append(sent_vec)
sent_vectors_cv = np.array(sent_vectors_cv)
```

```
print(sent_vectors_cv.shape)
print(sent_vectors_cv[0])
```

```
100%|| 5895/5895 [00:07<00:00, 799.38it/s]


(5895, 50)
[-6.17560340e-04  7.55331482e-06  1.52604801e-03  2.40280726e-04
 -1.42922778e-03 -1.54502122e-03  4.51177624e-04  1.19154557e-03
 -1.10354970e-03 -2.68429812e-04 -1.13595537e-03  7.80510857e-05
  1.11606513e-03 -8.40682008e-04 -2.22607275e-04 -4.25091946e-04
 -2.58808414e-04 -1.01643840e-03 -1.29009563e-04 -1.67991852e-03
  7.73125610e-05  2.24201082e-04  3.51375646e-05 -1.27404562e-03
  1.43981027e-04 -4.53258076e-04  3.02517129e-04 -1.50014476e-03
 -7.89884356e-04 -9.79280966e-04 -1.49033784e-04  3.14250708e-04
  1.96919172e-04 -3.11776286e-04  6.92771680e-04  1.98478877e-03
 -4.84275350e-04 -3.95487196e-05  5.34298823e-04  1.56117009e-04
  6.55606185e-04 -4.20311362e-04  6.91347921e-05 -9.35181200e-04
 -5.39197399e-04 -2.93190391e-03 -1.53409674e-03 -6.43824854e-04
 -9.83724858e-05  1.59880866e-03]
```

Converting test data

In [233]:
```
list_of_sentance_test=[]
for sentance in x_test:
    list_of_sentance_test.append(sentance.split())
```

In [234]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this lis
for sent in tqdm(list_of_sentance_test): # 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_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
print(sent_vectors_test[0])
```

```
100%|| 8422/8422 [00:10<00:00, 822.97it/s]
```

```
(8422, 50)
[-1.56219284e-03 -1.73564562e-03 -1.00170932e-03 -5.73513190e-04
 -2.28344657e-04  9.35348695e-05 -1.69841804e-03 -6.61160501e-04
  7.64101506e-04  5.97771903e-04  1.10285685e-03  4.80593677e-04
 -1.45315857e-03  1.22763432e-03  7.32614992e-06  1.34973279e-03
  1.88991957e-04 -6.46397938e-04 -7.48702534e-04 -6.22506058e-04
 -1.04922720e-03  9.21336006e-04  1.67835982e-03 -1.53249911e-03
 -9.46205665e-04  3.29417911e-05 -8.83911290e-04 -2.19926414e-05
  8.43897685e-04 -1.53225916e-03  2.04198787e-03  3.66656531e-04
  1.07966012e-03 -2.06591927e-03  8.75746773e-05 -2.57333644e-04
 -4.93653766e-04 -1.08368469e-03 -5.23676692e-04  5.19617067e-04
  7.22169644e-05  1.51460578e-03 -3.96378940e-04 -2.23183922e-03
  1.38375961e-03  1.05212250e-03 -8.95680219e-04  8.87400502e-04
 -2.79059866e-04 -8.46741223e-04]
```

[4.4.1.2] TFIDF weighted W2v

```
In [235]: # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
          model = TfidfVectorizer()
          tf_idf_matrix_train = model.fit_transform(x_train)
          # we are converting a dictionary with word as a key, and the idf as a value
          dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

Converting train data using tf-idf w2v

```
In [236]: # 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 = tfid

          tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review is stored in
          row=0;
          for sent in tqdm(list_of_sentance_train): # 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_train.append(sent_vec)
              row += 1
```

```
100%|| 13755/13755 [02:40<00:00, 85.76it/s]
```

```
In [331]: tfidf_sent_vectors_train = np.array(tfidf_sent_vectors_train)
```

converting cross validation data

```
In [237]: # 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 = tfid

          tfidf_sent_vectors_cv = []; # the tfidf-w2v for each sentence/review is stored in th
          row=0;
          for sent in tqdm(list_of_sentance_cv): # 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_cv.append(sent_vec)
              row += 1

100%|| 5895/5895 [01:07<00:00, 86.75it/s]
```

```
In [332]: tfidf_sent_vectors_cv = np.array(tfidf_sent_vectors_cv)
```

Converting test data

```
In [238]: # 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 = tfid

          tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in
          row=0;
          for sent in tqdm(list_of_sentance_test): # 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_test.append(sent_vec)
        row += 1
```

```
100%|| 8422/8422 [01:36<00:00, 87.13it/s]
```

In [333]:
```
tfidf_sent_vectors_test = np.array(tfidf_sent_vectors_test)
```

6 [5] Assignment 9: Random Forests

```
<li><strong>Apply Random Forests & GBDT 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>The hyper paramter tuning (Consider two hyperparameters: n_estimators & max_depth)
    <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>Feature importance</strong>
    <ul>
<li>Get top 20 important features and represent them in a word cloud. Do this for BOW & TFIDF.
    </ul>
</li>
<br>
<li><strong>Feature engineering</strong>
    <ul>
<li>To increase the performance of your model, you can also experiment with with feature engin
        <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='3d_plot.JPG' width=500px> with X-axis as <strong>n_estimators</strong>, Y-axis as <s
        <p style="text-align:center;font-size:30px;color:red;"><strong>(or)</strong></p> <br>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='heat_map.JPG' width=300px> <a href='https://seaborn.pydata.org/generated/seaborn.hea
<li>You choose either of the plotting techniques out of 3d plot or heat map</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.

6.1 Loading featurized data using joblib

```
In [4]: ##########################################    BoW    ##################################

        x_train_bow = joblib.load('x_tr_bow100k.pkl')
        x_test_bow  = joblib.load('x_te_bow100k.pkl')
        x_cv_bow    = joblib.load('x_cv_bow100k.pkl')

        y_train = joblib.load('y_train.pkl')
        y_test  = joblib.load('y_test.pkl')
        y_cv    = joblib.load('y_cv.pkl')
```

```
######################################### TF-IDF ########################################

x_train_tf = joblib.load('x_tr_tfidf100k.pkl')
x_test_tf  = joblib.load('x_te_tfidf100k.pkl')
x_cv_tf    = joblib.load('x_cv_tfidf100k.pkl')

######################################## average w2v ######################################

sent_vectors_train = joblib.load('sent_vectors_train_100k.pkl')
sent_vectors_test  = joblib.load('sent_vectors_test_100k.pkl')
sent_vectors_cv    = joblib.load('sent_vectors_cv_100k.pkl')

####################################### tfidf w2v  #######################################

tfidf_sent_vectors_train = joblib.load('tfidf_sent_vectors_train_100k.pkl')
tfidf_sent_vectors_test  = joblib.load('tfidf_sent_vectors_test_100k.pkl')
tfidf_sent_vectors_cv    = joblib.load('tfidf_sent_vectors_cv_100k.pkl')
```

6.2 [5.1] Applying RF

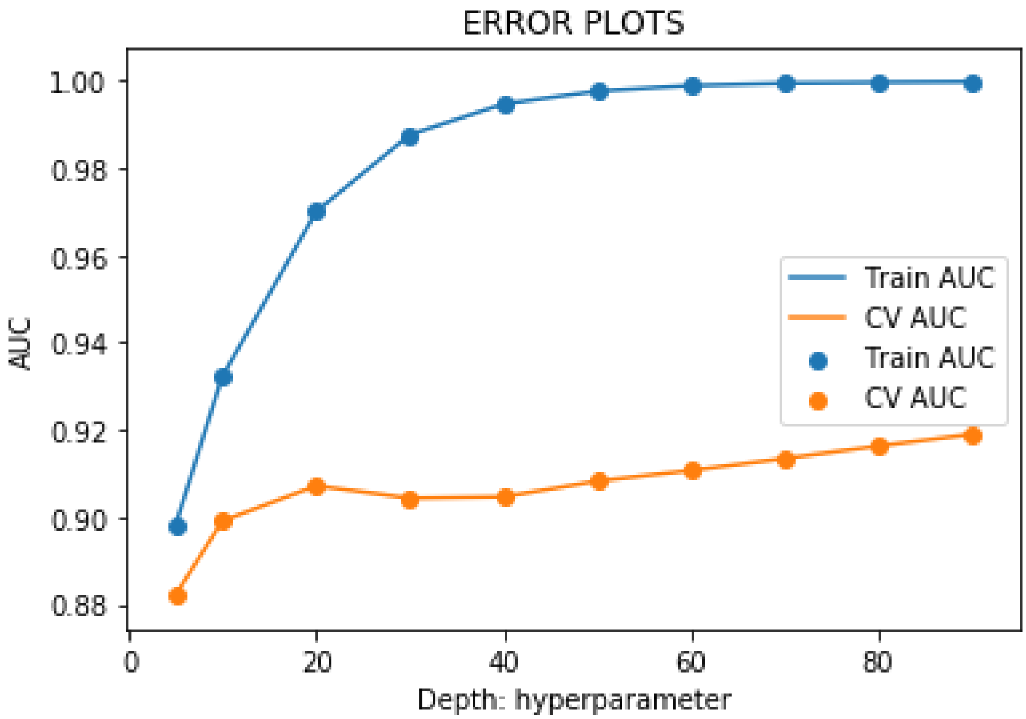
As here we have to tune two hyperparameter, so we'll use GridSearchCV for hyperparameter tuning.

In [24]:
```
############ tuning depth parameter first ###########
start = datetime.now()
depth = [5,10,20,30,40,50,60,70,80,90]
parameters = {'max_depth': depth}
grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',n_estimators = 10
grid.fit(x_train_bow, y_train)

print("Accuracy on train data = ", grid.best_score_*100)
optimal_depth1 = grid.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth1)
print("Time taken: ", datetime.now() - start)
```

```
Accuracy on train data =  91.62604888002083
The optimal number of depth is :  90
Time taken:  0:04:21.318210
```

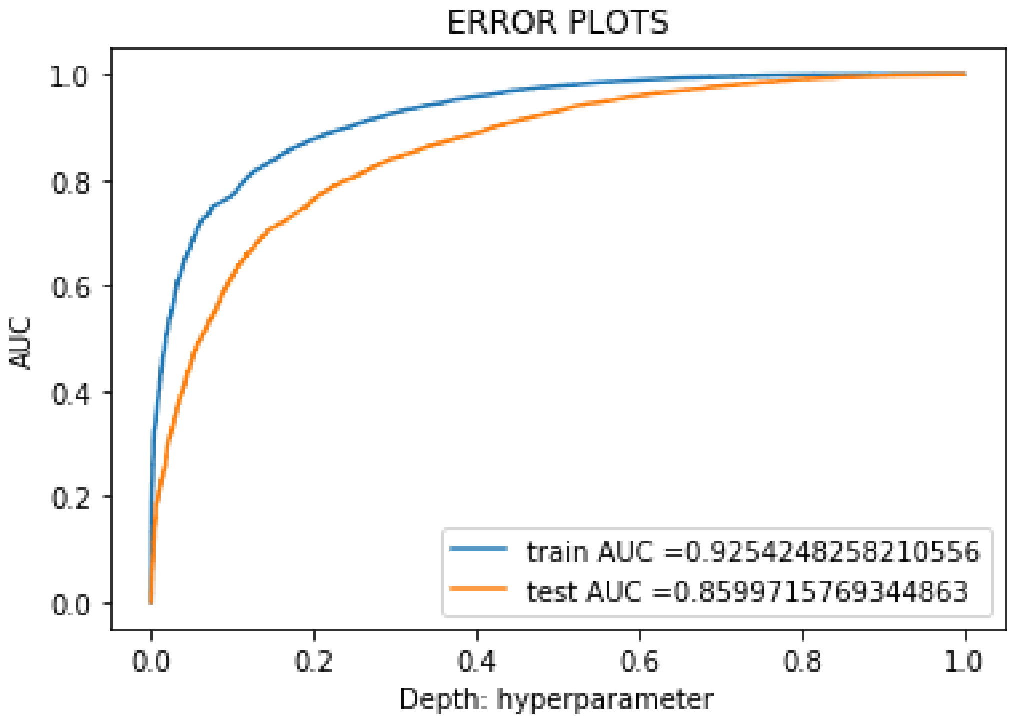
In [8]:
```
train_auc_bow = grid.cv_results_['mean_train_score']
cv_auc_bow = grid.cv_results_['mean_test_score']

plt.plot(depth, train_auc_bow, label='Train AUC')
plt.scatter(depth, train_auc_bow, label='Train AUC')

plt.plot(depth, cv_auc_bow, label='CV AUC')
plt.scatter(depth, cv_auc_bow, label='CV AUC')
```

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



Note: Here we can observe that after depth=20 model's performance is not increasing much. So we could take depth=20 and if we take depth= 90 then model may overfit.

```
In [26]: clf = RandomForestClassifier(max_depth = 20, class_weight ='balanced')
         clf.fit(x_train_bow, y_train)

         train_fpr_bow, train_tpr_bow, thresholds_bow = roc_curve(y_train, clf.predict_proba(x_
         test_fpr_bow, test_tpr_bow, thresholds_bow = roc_curve(y_test, clf.predict_proba(x_tes

         plt.plot(train_fpr_bow, train_tpr_bow, label="train AUC ="+str(auc(train_fpr_bow, trai
         plt.plot(test_fpr_bow, test_tpr_bow, label="test AUC ="+str(auc(test_fpr_bow, test_tpr

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

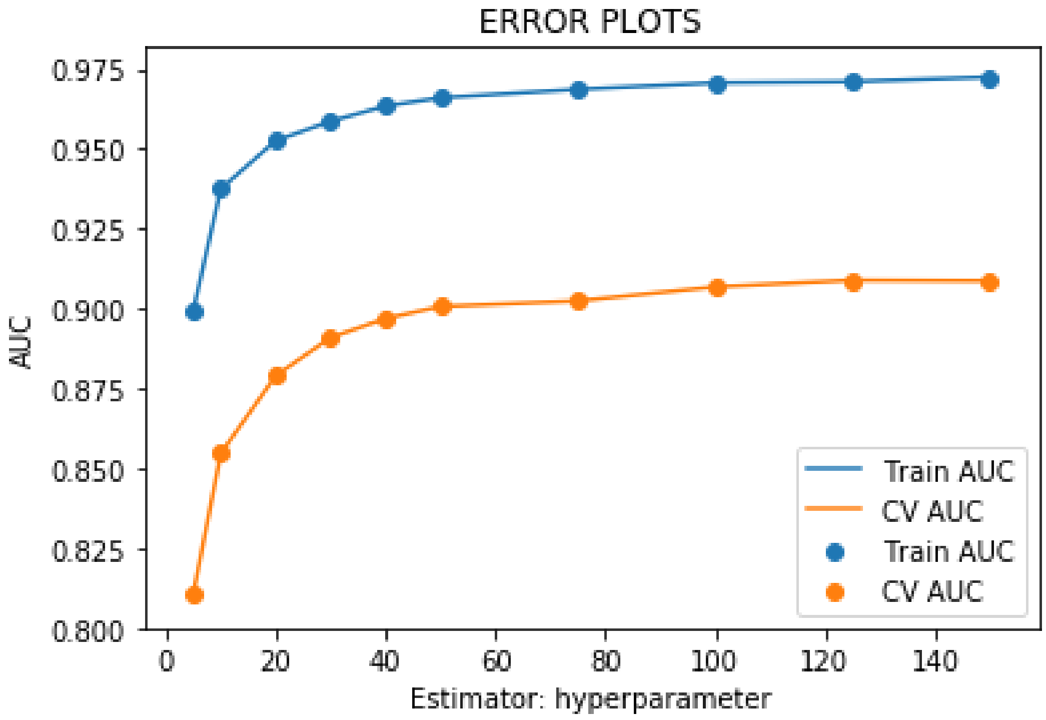



```
In [15]: ######################### tuning for hyperparameter n_estimator ###########

         start = datetime.now()
         estimator = [5,10,20,30,40,50,75,100,125,150]
         parameters = {'n_estimators': estimator}
         grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',max_depth = 20), p
         grid.fit(x_train_bow, y_train)

         print("Accuracy on train data = ", grid.best_score_*100)
         optimal_est1 = grid.best_estimator_.n_estimators
         print("The optimal number of depth is : ",optimal_est1)
         print("Time taken: ", datetime.now() - start)
```

```
Accuracy on train data =  90.9036607195439
The optimal number of depth is :  125
Time taken:  0:00:57.372467
```

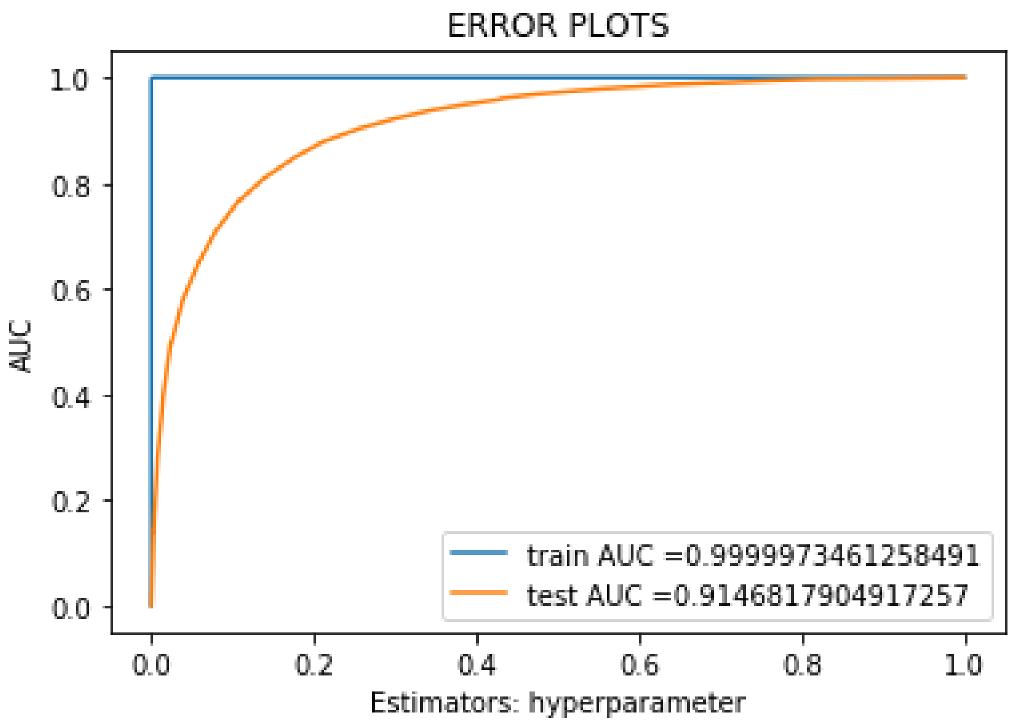
```
In [16]: train_auc_bow = grid.cv_results_['mean_train_score']
         cv_auc_bow = grid.cv_results_['mean_test_score']

         plt.plot(estimator, train_auc_bow, label='Train AUC')
         plt.scatter(estimator, train_auc_bow, label='Train AUC')
```

```
plt.plot(estimator, cv_auc_bow, label='CV AUC')
plt.scatter(estimator, cv_auc_bow, label='CV AUC')

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




```
In [21]: start = datetime.now()
         clf = RandomForestClassifier(n_estimators = 40, class_weight ='balanced')
         clf.fit(x_train_bow, y_train)

         train_fpr_bow, train_tpr_bow, thresholds_bow = roc_curve(y_train, clf.predict_proba(x_
         test_fpr_bow, test_tpr_bow, thresholds_bow = roc_curve(y_test, clf.predict_proba(x_tes

         plt.plot(train_fpr_bow, train_tpr_bow, label="train AUC ="+str(auc(train_fpr_bow, trai
         plt.plot(test_fpr_bow, test_tpr_bow, label="test AUC ="+str(auc(test_fpr_bow, test_tp

         plt.legend()
         plt.xlabel("Estimators: hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
```

```
plt.show()
print("Time taken: ", datetime.now() - start)
```




```
Time taken:  0:00:28.668026
```

```
In [22]: start = datetime.now()
         depth     = [5,10,20,30,40,50,60,70, 80, 90 ]
         estimator = [5,10,20,30,40,50,75,100,125,150]

         parameters = {'n_estimators': estimator,'max_depth': depth}
         grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced'), parameters, cv=
         grid.fit(x_train_bow, y_train)
         print("Time taken: ", datetime.now() - start)
```

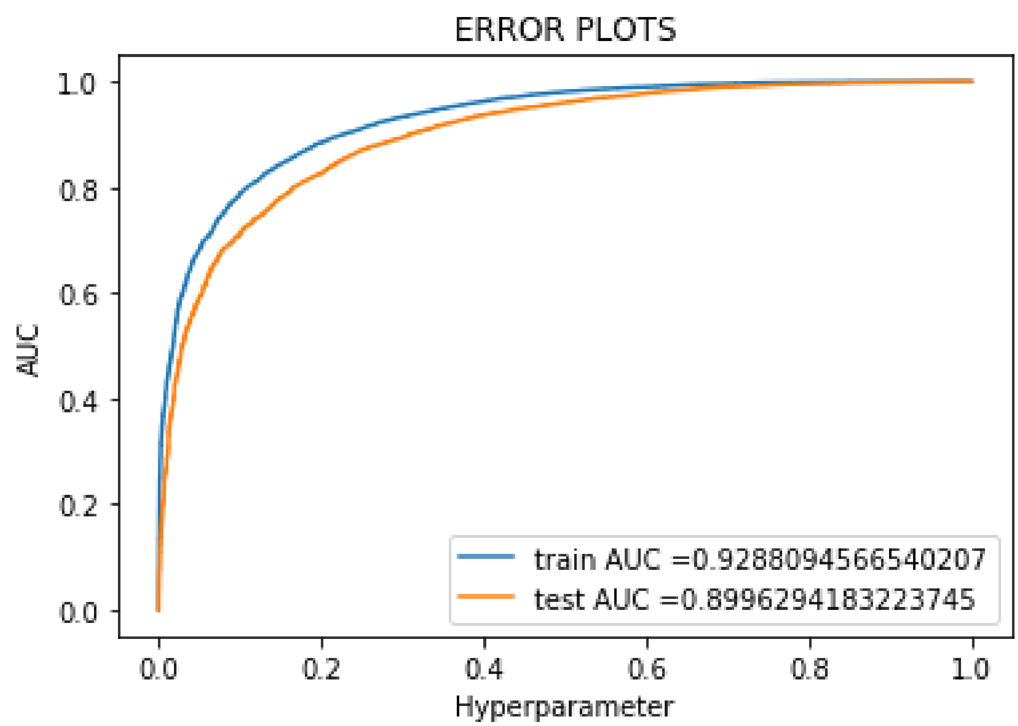
```
Time taken:  0:23:16.855739
```

```
In [23]: grid.best_params_
```

```
Out[23]: {'max_depth': 90, 'n_estimators': 125}
```

```
In [364]: optimal_depth1=20
          optimal_est1  =40
```

```
In [29]: clf = RandomForestClassifier(max_depth = optimal_depth1, n_estimators= optimal_est1, c
         clf.fit(x_train_bow, y_train)

         train_fpr_bow, train_tpr_bow, thresholds_bow = roc_curve(y_train, clf.predict_proba(x_
         test_fpr_bow, test_tpr_bow, thresholds_bow = roc_curve(y_test, clf.predict_proba(x_tes

         plt.plot(train_fpr_bow, train_tpr_bow, label="train AUC ="+str(auc(train_fpr_bow, trai
         plt.plot(test_fpr_bow, test_tpr_bow, label="test AUC ="+str(auc(test_fpr_bow, test_tpr
         plt.legend()
         plt.xlabel("Hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```




```
In [30]: clf = RandomForestClassifier(n_estimators = optimal_est1, max_depth = optimal_depth1)
         clf.fit(x_train_bow, y_train)
         predb1 = clf.predict(x_test_bow)

         acc_1 = accuracy_score(y_test, predb1) * 100
         pre_1 = precision_score(y_test, predb1) * 100
         rec_1 = recall_score(y_test, predb1) * 100
         f1_1  = f1_score(y_test, predb1) * 100

         print('\nAccuracy = %f%%' % (acc_1))
```

```
print('\nprecision= %f%%' % (pre_1))
print('\nrecall    = %f%%' % (rec_1))
print('\nF1-Score = %f%%' % (f1_1))
```

```
Accuracy = 84.182743%

precision= 84.181542%

recall    = 100.000000%

F1-Score = 91.411486%
```

6.2.1 [5.1.1] Applying Random Forests on BOW, SET 1

6.2.2 [5.1.2] Wordcloud of top 20 important features from SET 1

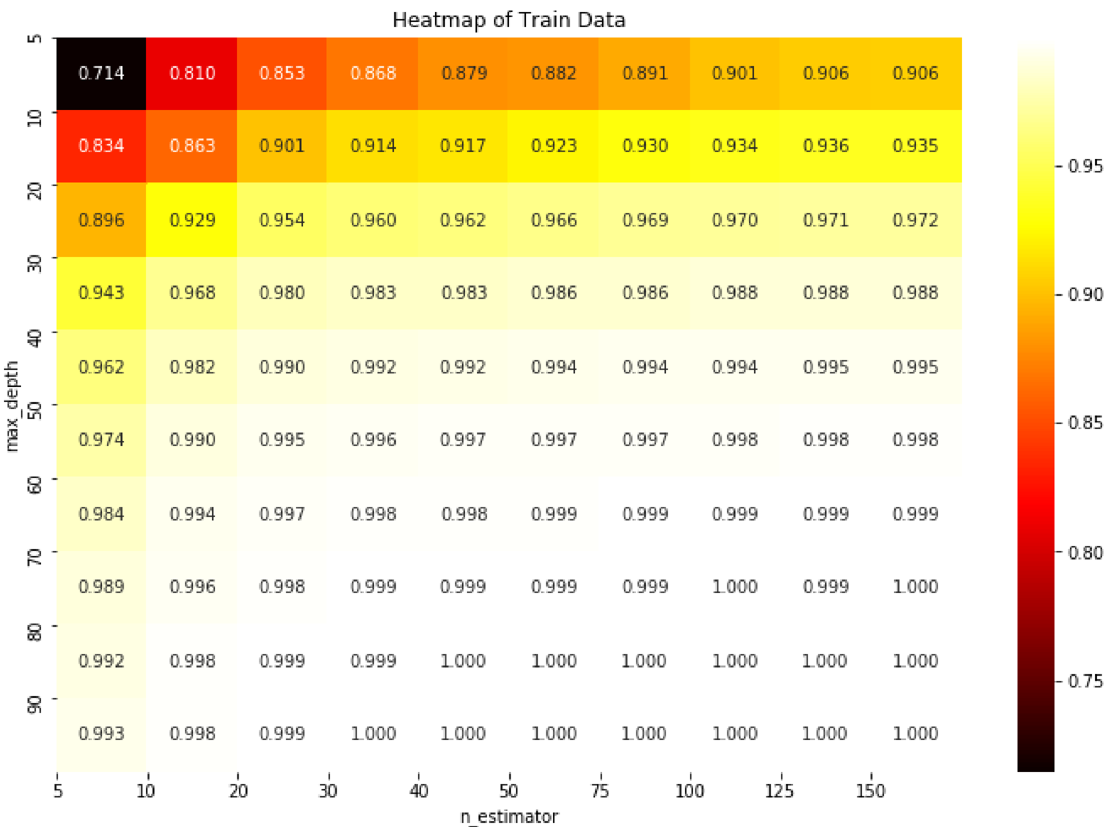
```
In [40]: # Calculate feature importances from decision trees
         importances = clf.feature_importances_
         indices = list(np.argsort(importances)[::-1][:20])
         names = np.array(count_vect.get_feature_names())
         print(names[indices],)
```

```
['not' 'great' 'love' 'favorite' 'disappointed' 'delicious' 'worst'
 'highly' 'bad' 'loves' 'waste' 'easy' 'money' 'wonderful' 'excellent'
 'nice' 'maybe' 'best' 'find' 'perfect']
```

```
In [39]: text = str(names[indices])
         wordcloud = WordCloud(max_font_size=50, max_words=30, background_color="white").genera

         # Display the generated image:
         plt.imshow(wordcloud, interpolation="bilinear")
         plt.axis("off")
         plt.show()
```

6.3 Heatmap for test data

In [33]:
```
heat_map1 = grid.cv_results_['mean_train_score'].reshape(len(estimator),len(depth))
plt.figure(figsize=(12,8))
sns.heatmap(heat_map1, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator,
plt.xlabel('n_estimator')
plt.ylabel('max_depth')
plt.xticks(np.arange(len(estimator)), estimator)
plt.yticks(np.arange(len(depth)), depth)
plt.title('Heatmap of Train Data')
plt.show()
```



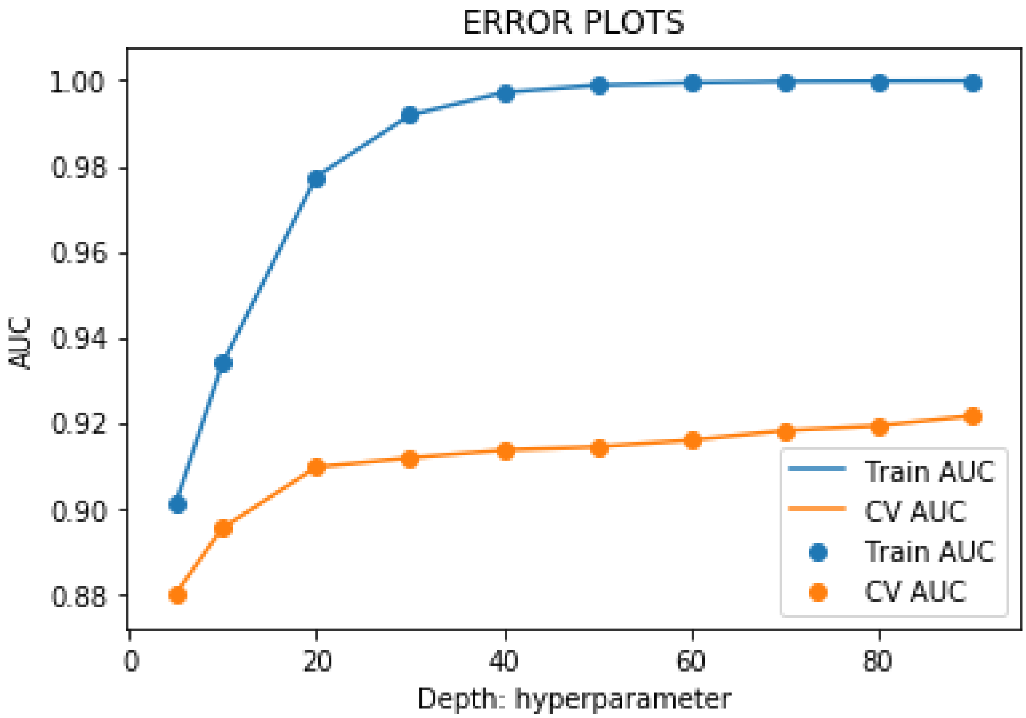
6.3.1 [5.1.3] Applying Random Forests on TFIDF, SET 2

```
In [34]: ############ tuning depth parameter first ###########
         start = datetime.now()
         depth = [5,10,20,30,40,50,60,70,80,90]
         parameters = {'max_depth': depth}
         grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',n_estimators = 10
         grid.fit(x_train_tf, y_train)

         print("Accuracy on train data = ", grid.best_score_*100)
         optimal_depth2 = grid.best_estimator_.max_depth
         print("The optimal number of depth is : ",optimal_depth2)
         print("Time taken: ", datetime.now() - start)
```

```
Accuracy on train data =  92.15098732102456
The optimal number of depth is :  90
Time taken:  0:04:18.953366
```

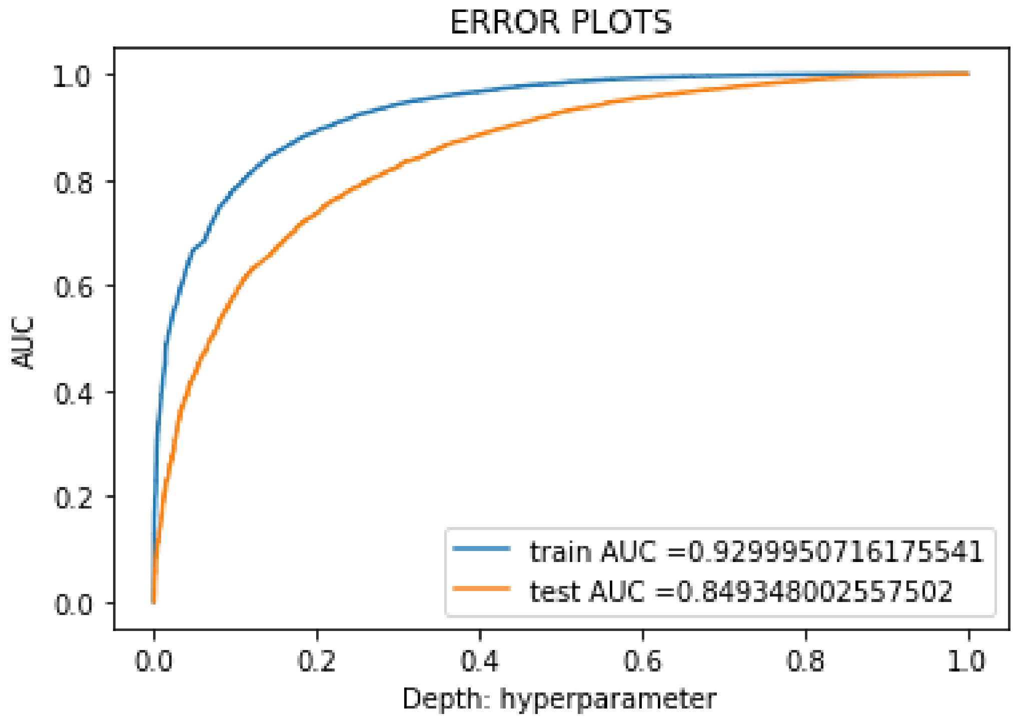
```
In [35]: train_auc = grid.cv_results_['mean_train_score']
         cv_auc = grid.cv_results_['mean_test_score']
```

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

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

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




```
In [44]: clf = RandomForestClassifier(max_depth = 20, class_weight ='balanced')
         clf.fit(x_train_tf, y_train)

         train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_tf)[:
         test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(x_test_tf)[:,1]

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

         plt.legend()
         plt.xlabel("Depth: hyperparameter")
```

```
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

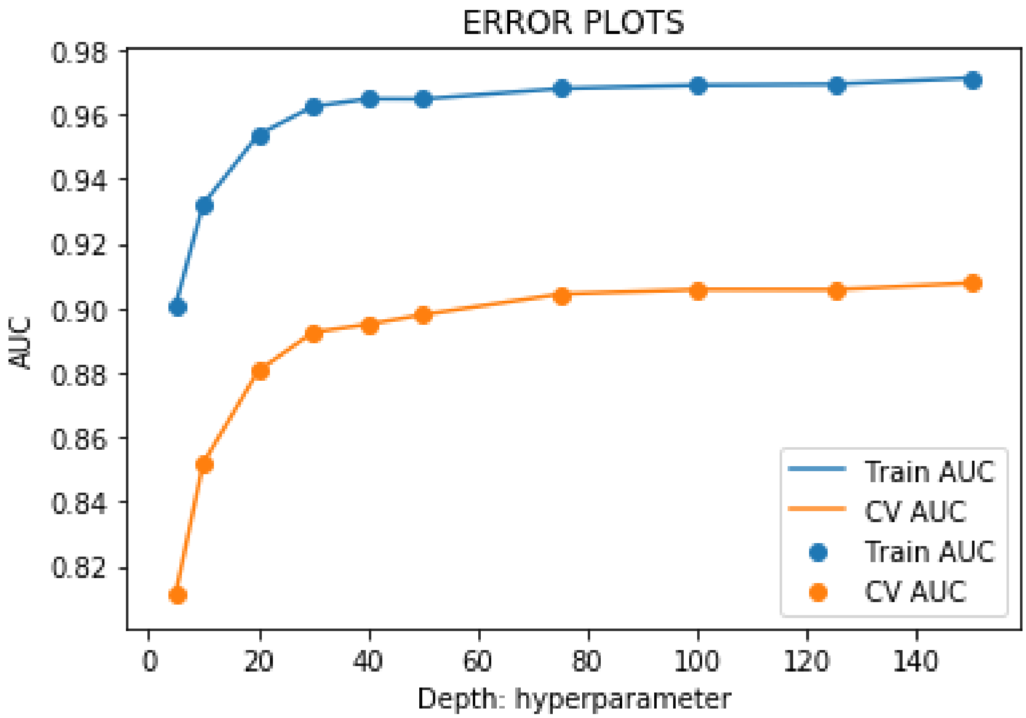



```
In [55]: ######################### tuning for hyperparameter n_estimator ###########

         start = datetime.now()
         estimator = [5,10,20,30,40,50,75,100,125,150]
         parameters = {'n_estimators': estimator}
         grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',max_depth = 20), p
         grid.fit(x_train_bow, y_train)

         print("Accuracy on train data = ", grid.best_score_*100)
         optimal_est2 = grid.best_estimator_.n_estimators
         print("The optimal number of estimators : ",optimal_est2)
         print("Time taken: ", datetime.now() - start)
```

```
Accuracy on train data =  90.78019606103344
The optimal number of estimators :  150
Time taken:  0:00:52.949006
```

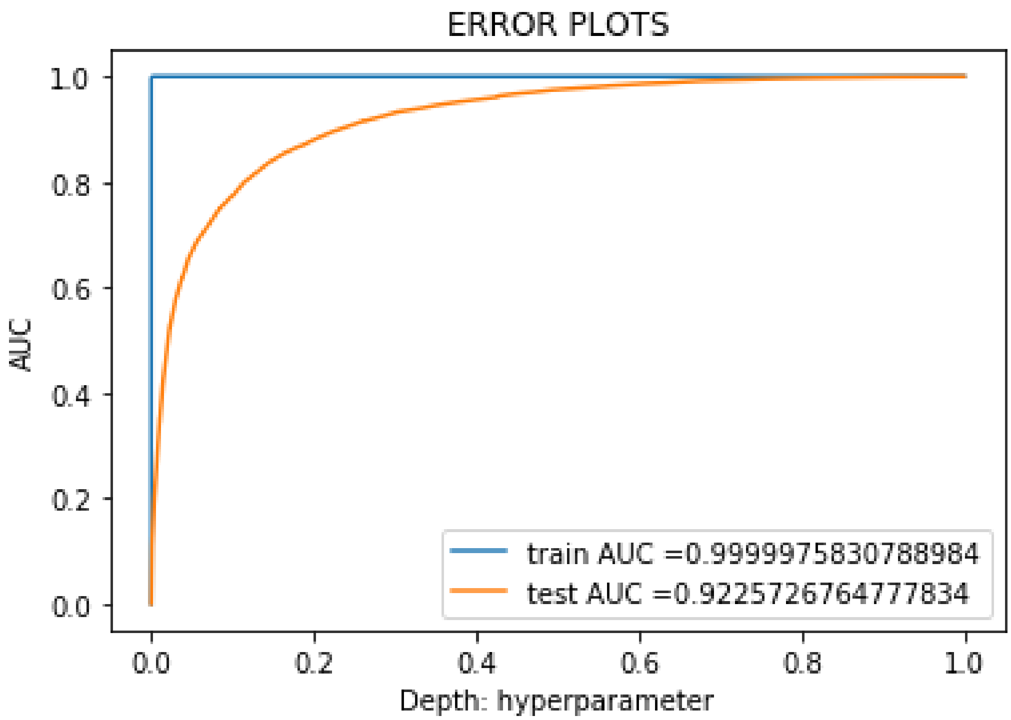
```
In [62]: train_auc = grid.cv_results_['mean_train_score']
         cv_auc = grid.cv_results_['mean_test_score']
```

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

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

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




```
In [63]: clf = RandomForestClassifier(n_estimators= 80, class_weight ='balanced')
         clf.fit(x_train_tf, y_train)

         train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_tf)[:
         test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(x_test_tf)[:,1]

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

         plt.legend()
         plt.xlabel("Depth: hyperparameter")
```

```
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```




```
In [64]: start = datetime.now()
         depth     = [5,10,20,30,40,50,60,70, 80, 90 ]
         estimator = [5,10,20,30,40,50,75,100,125,150]

         parameters = {'n_estimators': estimator,'max_depth': depth}
         grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced'), parameters, cv=
         grid.fit(x_train_tf, y_train)
         print("Time taken: ", datetime.now() - start)
```

```
Time taken:  0:23:10.944086
```

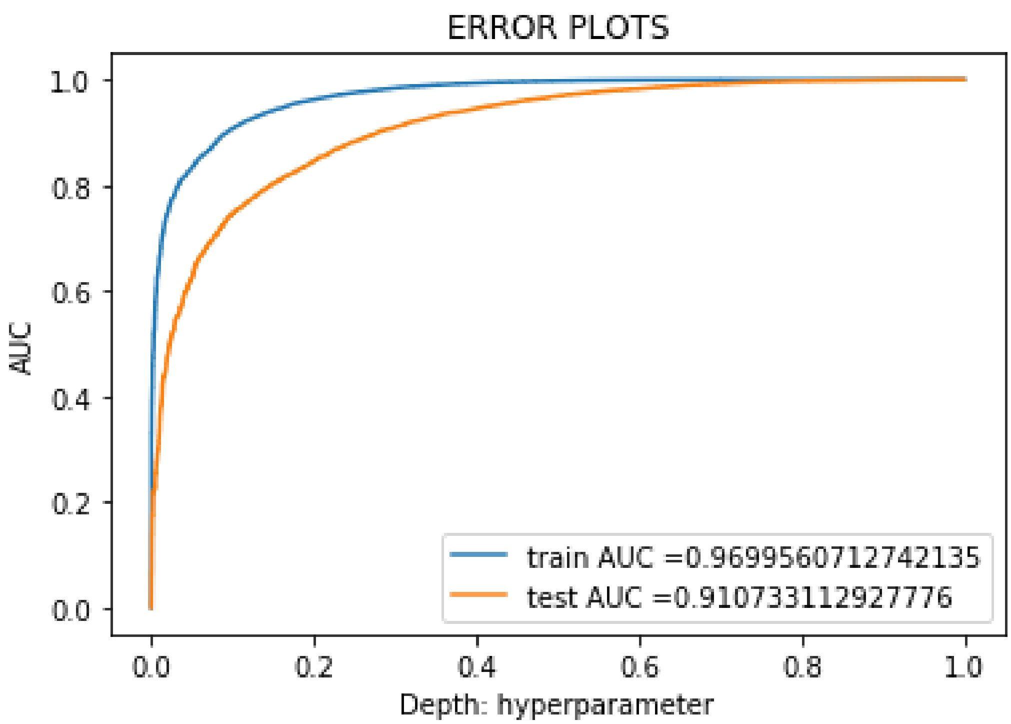
```
In [66]: grid.best_params_
```

```
Out[66]: {'max_depth': 90, 'n_estimators': 150}
```

```
In [365]: optimal_depth2=20
          optimal_est2  =80
```

```
In [68]: clf = RandomForestClassifier(n_estimators=optimal_est2  , max_depth=optimal_depth2 ,
         clf.fit(x_train_tf, y_train)
```

```
train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_tf)[:
test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(x_test_tf)[:,1]

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

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




```
In [69]: clf = RandomForestClassifier(max_depth = optimal_depth2, n_estimators= optimal_est2)
         clf.fit(x_train_tf,y_train)
         pred = clf.predict(x_test_tf)

         acc_2 = accuracy_score(y_test, pred) * 100
         pre_2 = precision_score(y_test, pred) * 100
         rec_2 = recall_score(y_test, pred) * 100
         f1_2  = f1_score(y_test, pred) * 100

         print('\nAccuracy = %f%%' % (acc_2))
         print('\nprecision= %f%%' % (pre_2))
```

```
print('\nrecall    = %f%%' % (rec_2))
print('\nF1-Score = %f%%' % (f1_2))
```

```
Accuracy = 84.649856%

precision= 84.579279%

recall    = 99.995488%

F1-Score = 91.643581%
```

6.3.2 [5.1.4] Wordcloud of top 20 important features from SET 2

```
In [45]: # Calculate feature importances from decision trees
         importances = clf.feature_importances_
         indices = list(np.argsort(importances)[::-1][:20])
         names = np.array(tf_idf.get_feature_names())
         print(names[indices],)
```

```
['not' 'love' 'great' 'delicious' 'best' 'disappointed' 'highly' 'worst'
 'bad' 'money' 'maybe' 'good' 'find' 'product' 'unfortunately' 'loves'
 'would' 'disgusting' 'easy' 'away']
```

```
In [46]: text = str(names[indices])
         wordcloud = WordCloud(max_font_size=50, max_words=30, background_color="white").genera

         # Display the generated image:
         plt.imshow(wordcloud, interpolation="bilinear")
         plt.axis("off")
         plt.show()
```

6.4 Heatmap on test data

```
In [70]: heat_map2 = grid.cv_results_['mean_train_score'].reshape(len(estimator),len(depth))
         plt.figure(figsize=(12,8))
         sns.heatmap(heat_map2, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator,
         plt.xlabel('n_estimator')
         plt.ylabel('max_depth')
         plt.xticks(np.arange(len(estimator)), estimator)
         plt.yticks(np.arange(len(depth)), depth)
         plt.title('Heatmap of Train Data')
         plt.show()
```

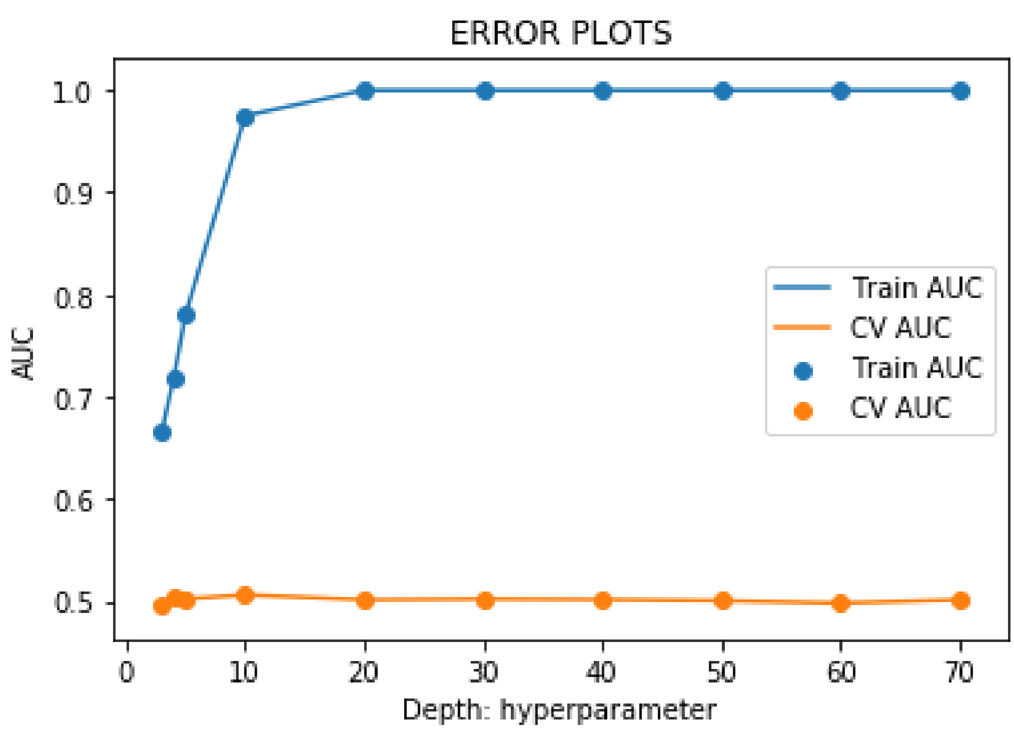
6.4.1 [5.1.5] Applying Random Forests on AVG W2V, SET 3

```
In [88]: ############ tuning depth parameter first ###########
         start = datetime.now()
         depth = [3,4,5,10,20,30,40,50,60,70]
         parameters = {'max_depth': depth}
```

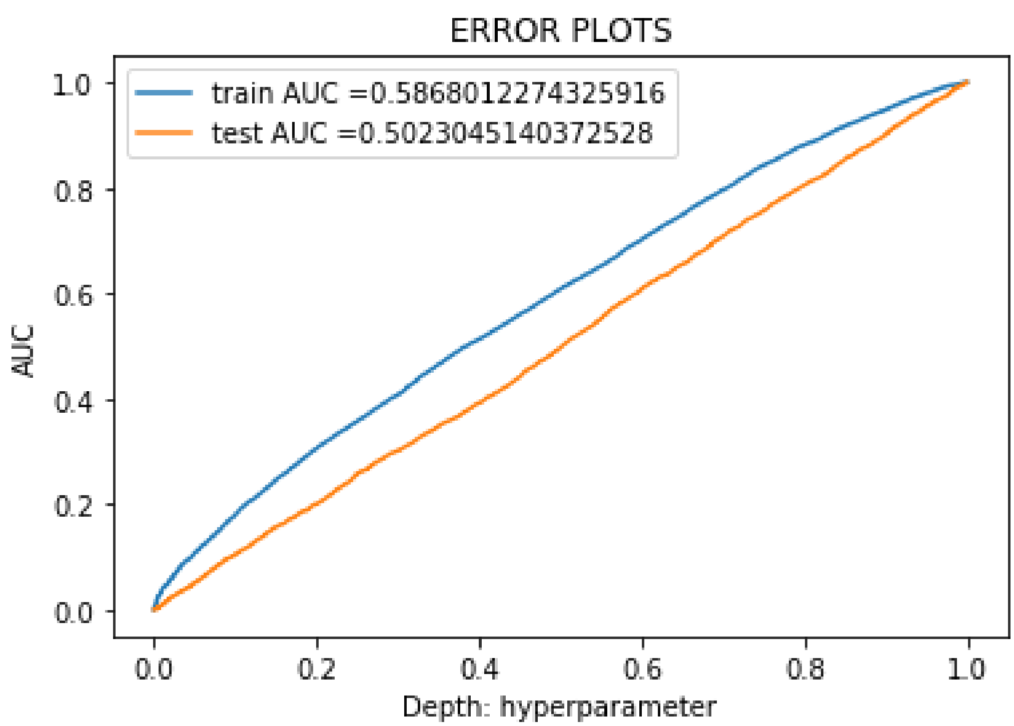
```
grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',n_estimators = 100
grid.fit(sent_vectors_train, y_train)

print("Accuracy on train data = ", grid.best_score_*100)
optimal_depth3 = grid.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth3)
print("Time taken: ", datetime.now() - start)
```

```
Accuracy on train data =  50.60775812944234
The optimal number of depth is :  10
Time taken:  0:09:40.777559
```

In [89]:
```
train_auc = grid.cv_results_['mean_train_score']
cv_auc = grid.cv_results_['mean_test_score']

plt.plot(depth, train_auc, label='Train AUC')
plt.scatter(depth, train_auc, label='Train AUC')

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

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



In [92]:
```
clf = RandomForestClassifier(max_depth= 4, class_weight ='balanced')
clf.fit(sent_vectors_train, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(sent_vectors_t
test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(sent_vectors_te

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

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

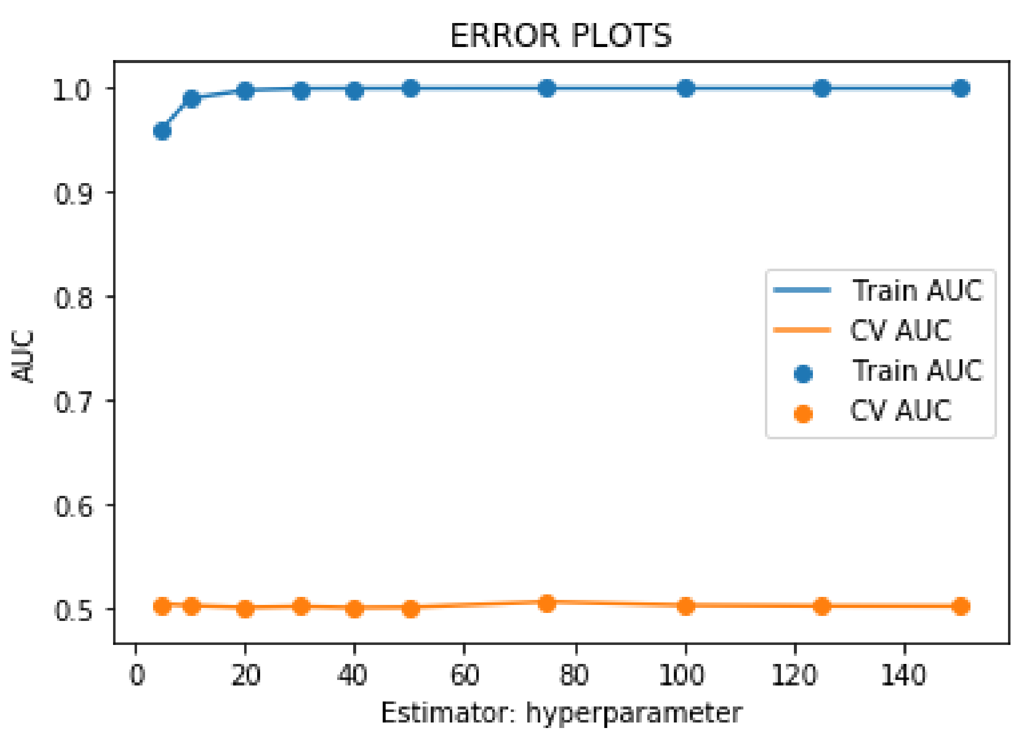


In [93]:
```
########################## tuning for hyperparameter n_estimator ###########

start = datetime.now()
estimator = [5,10,20,30,40,50,75,100,125,150]
parameters = {'n_estimators': estimator}
grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',max_depth = 20), p
```

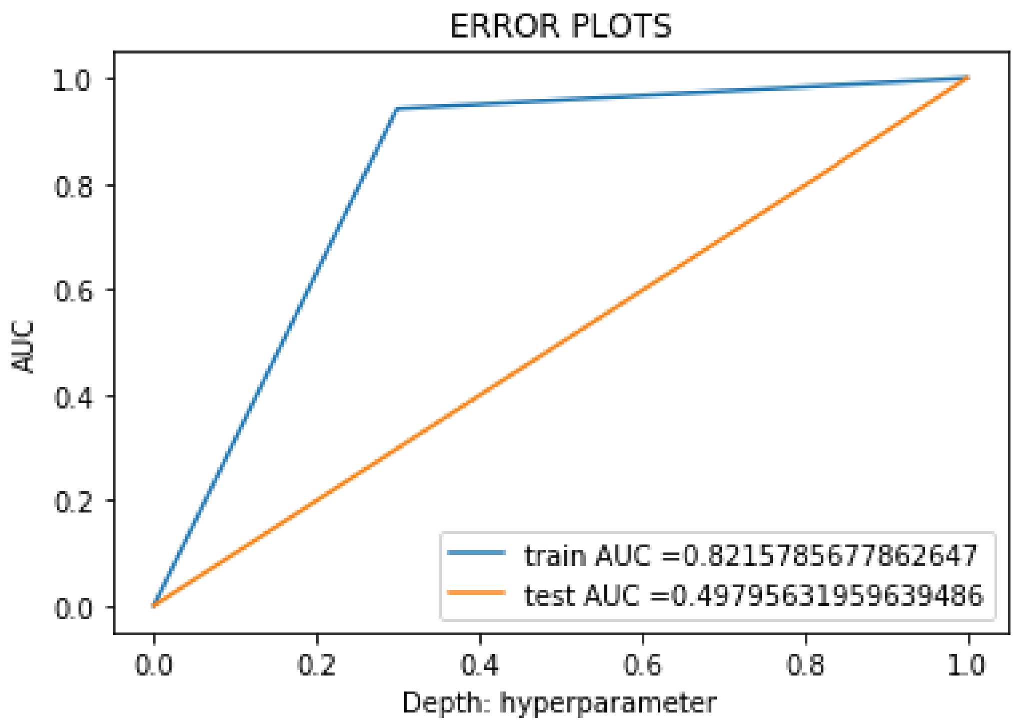
```
grid.fit(sent_vectors_train, y_train)

print("Accuracy on train data = ", grid.best_score_*100)
optimal_est3 = grid.best_estimator_.n_estimators
print("The optimal number of depth is : ",optimal_est3)
print("Time taken: ", datetime.now() - start)
```

```
Accuracy on train data =  50.497741150059085
The optimal number of depth is :  75
Time taken:  0:07:44.084420
```

In [94]:
```
train_auc = grid.cv_results_['mean_train_score']
cv_auc = grid.cv_results_['mean_test_score']

plt.plot(estimator, train_auc, label='Train AUC')
plt.scatter(estimator, train_auc, label='Train AUC')

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

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



In [99]:
```
clf = RandomForestClassifier(n_estimators= 1, class_weight ='balanced')
clf.fit(sent_vectors_train, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(sent_vectors_t
test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(sent_vectors_te

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

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



In [100]:
```
start = datetime.now()
depth     = [3,4,5,10,20,30,40,50,60,70]
estimator = [1,2,3,5, 10,20,30,40,50,75]

parameters = {'n_estimators': estimator,'max_depth': depth}
grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced'), parameters, cv=
```

```
grid.fit(sent_vectors_train, y_train)
print("Time taken: ", datetime.now() - start)
```

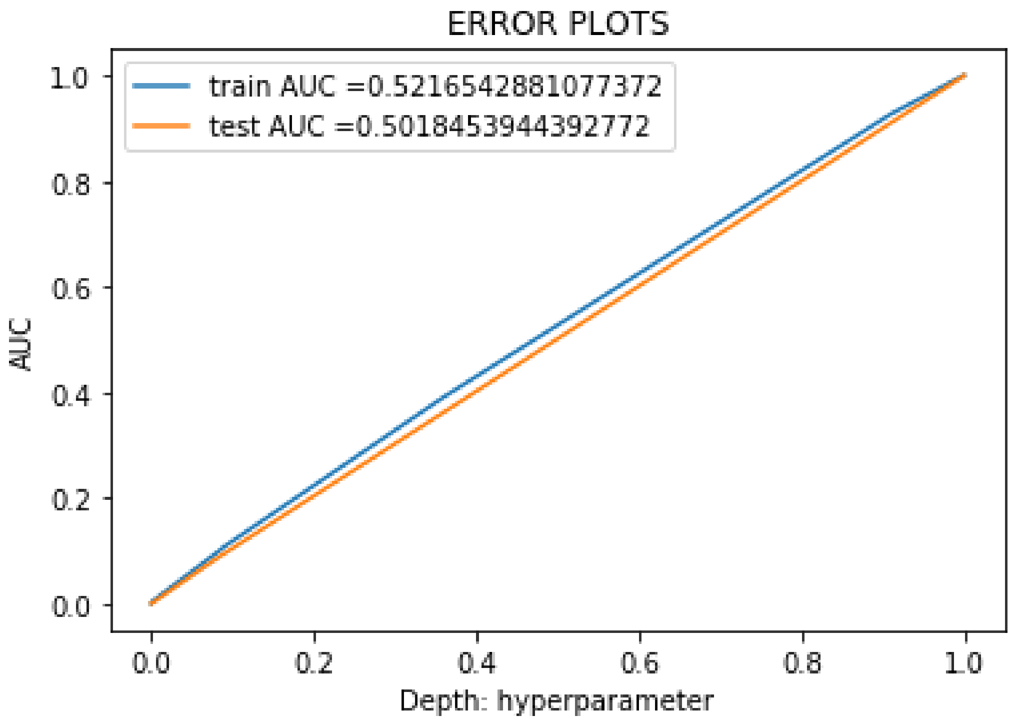
```
Time taken:  0:21:27.924150
```

In [101]:
```
grid.best_params_
```

Out[101]:
```
{'max_depth': 50, 'n_estimators': 20}
```

In [110]:
```
optimal_depth3=3
optimal_est3  =1
```

In [111]:
```
clf = RandomForestClassifier(n_estimators= 1, max_depth= 3, class_weight ='balanced')
clf.fit(sent_vectors_train, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(sent_vectors_
test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(sent_vectors_t

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

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

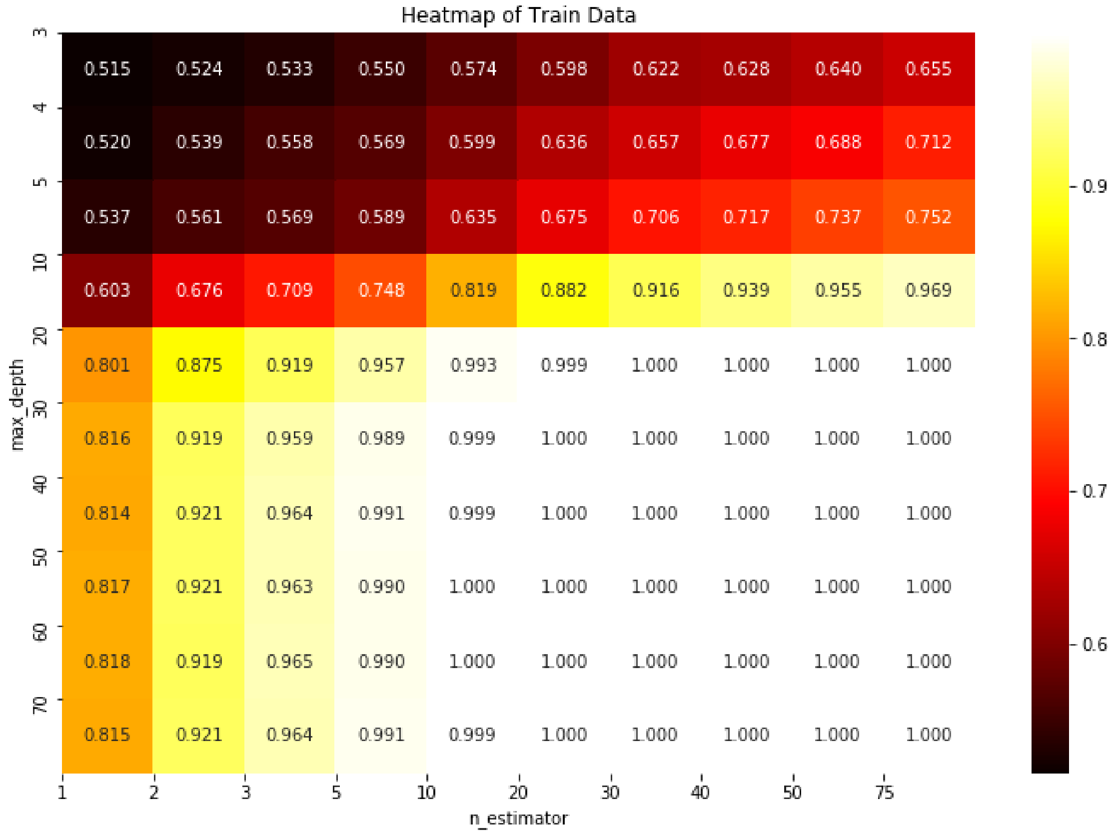


In [120]:
```
clf = RandomForestClassifier(max_depth = optimal_depth3, n_estimators= optimal_est3,
clf.fit(sent_vectors_train,y_train)
pred = clf.predict(sent_vectors_test)

acc_3 = accuracy_score(y_test, pred) * 100
pre_3 = precision_score(y_test, pred) * 100
rec_3 = recall_score(y_test, pred) * 100
f1_3  = f1_score(y_test, pred) * 100

print('\nAccuracy = %f%%' % (acc_3))
print('\nprecision= %f%%' % (pre_3))
print('\nrecall   = %f%%' % (rec_3))
print('\nF1-Score = %f%%' % (f1_3))
```

```
Accuracy = 82.990278%

precision= 84.179808%

recall   = 98.258516%

F1-Score = 90.675937%
```

6.5 Heatmap for test data

In [121]:
```
heat_map3 = grid.cv_results_['mean_train_score'].reshape(len(estimator),len(depth))
plt.figure(figsize=(12,8))
sns.heatmap(heat_map3, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator
plt.xlabel('n_estimator')
plt.ylabel('max_depth')
plt.xticks(np.arange(len(estimator)), estimator)
plt.yticks(np.arange(len(depth)), depth)
plt.title('Heatmap of Train Data')
plt.show()
```



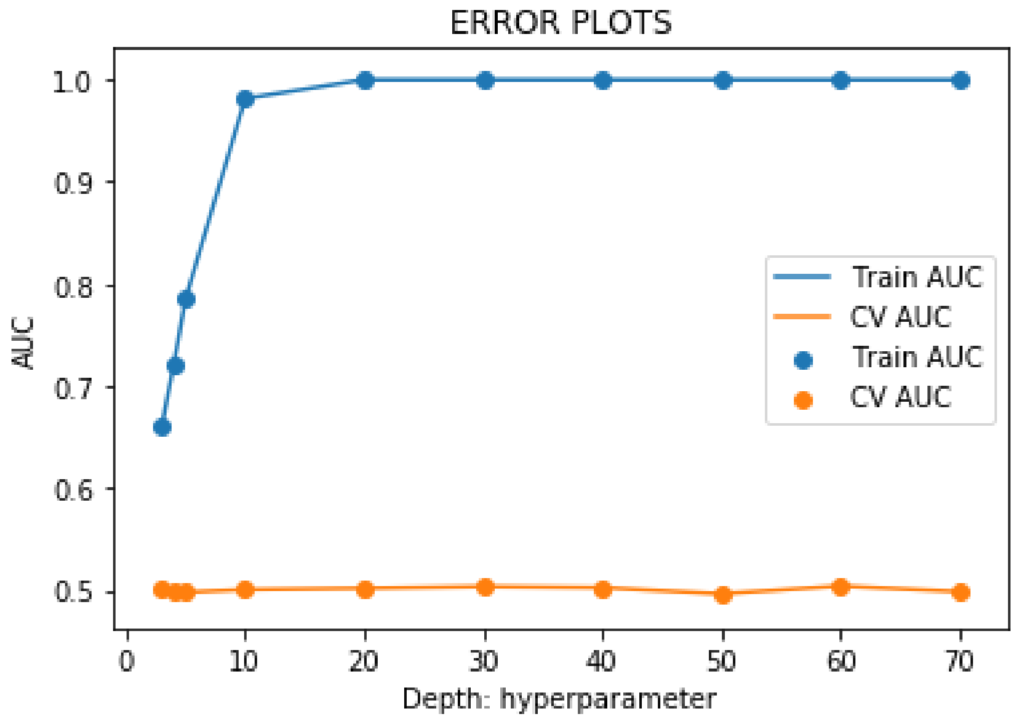
6.5.1 [5.1.6] Applying Random Forests on TFIDF W2V, SET 4

```
In [122]: ############ tuning depth parameter first ###########
          start = datetime.now()
          depth = [3,4,5,10,20,30,40,50,60,70] #estimator = [1,2,3,5, 10,20,30,40,50,75]
          parameters = {'max_depth': depth}
          grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',n_estimators = 1(
          grid.fit(tfidf_sent_vectors_train, y_train)

          print("Accuracy on train data = ", grid.best_score_*100)
          optimal_depth4 = grid.best_estimator_.max_depth
          print("The optimal number of depth is : ",optimal_depth4)
          print("Time taken: ", datetime.now() - start)
```

```
Accuracy on train data =  50.4149733641015
The optimal number of depth is :  60
Time taken:  0:11:00.031061
```

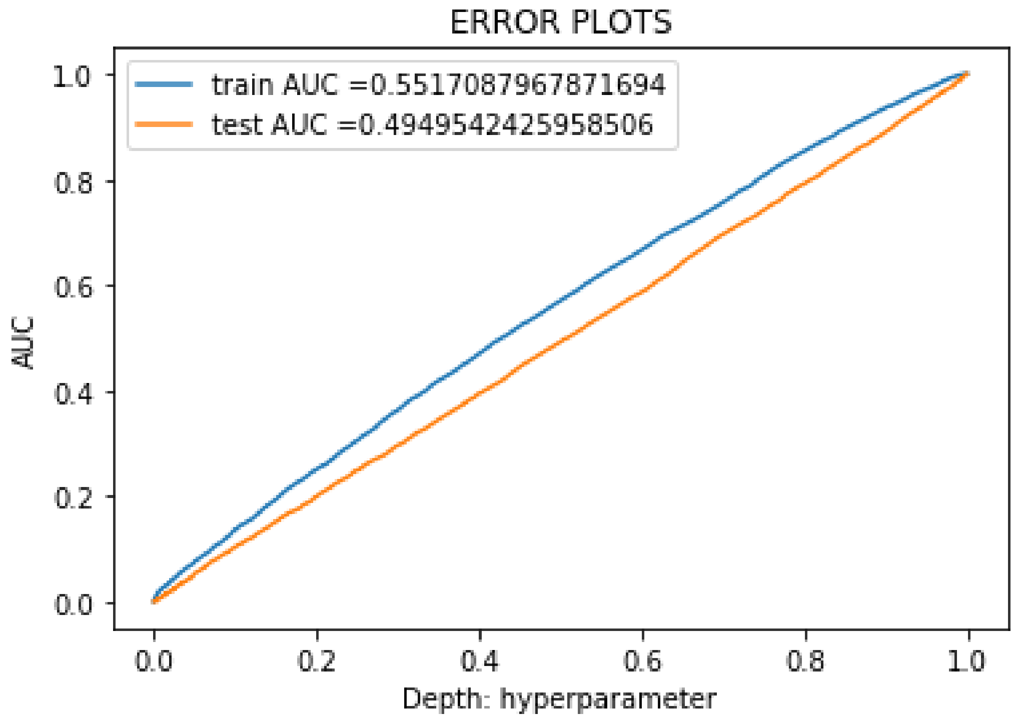
```
In [123]: train_auc = grid.cv_results_['mean_train_score']
          cv_auc = grid.cv_results_['mean_test_score']
```

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

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

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




```
In [127]: clf = RandomForestClassifier(max_depth= 3, class_weight ='balanced')
          clf.fit(tfidf_sent_vectors_train, y_train)

          train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(tfidf_sent_ve
          test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(tfidf_sent_vec

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

          plt.legend()
          plt.xlabel("Depth: hyperparameter")
```

```
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

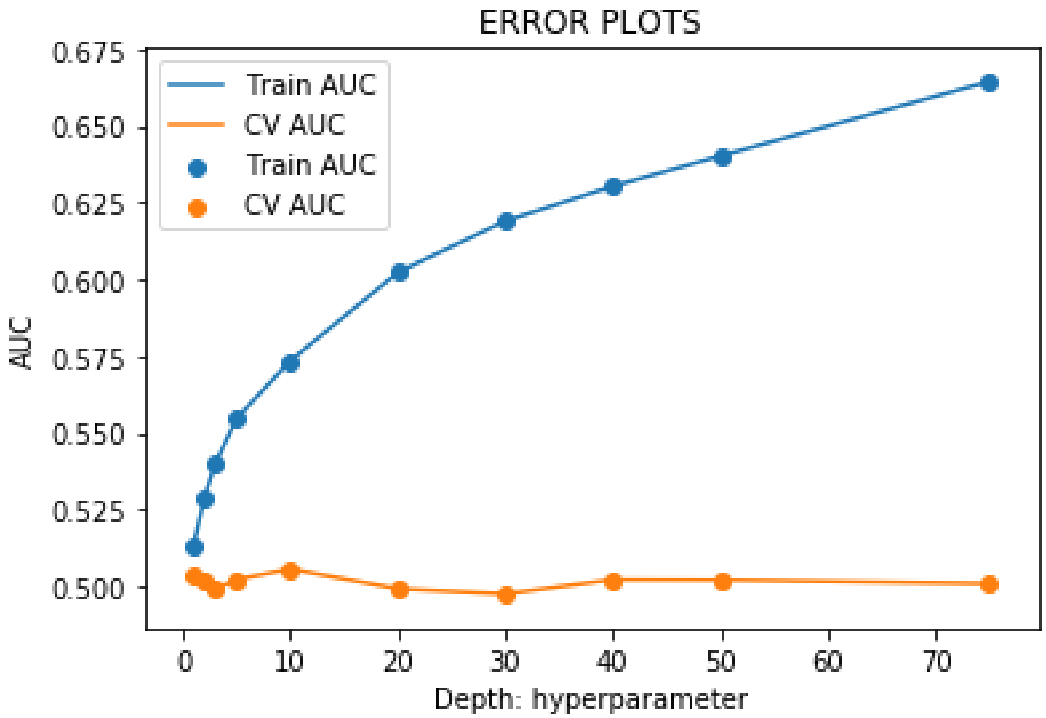


In [128]:
```
########################## tuning for hyperparameter n_estimator ###########

start = datetime.now()
estimator = [1,2,3,5, 10,20,30,40,50,75]
parameters = {'n_estimators': estimator}
grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',max_depth = 3), p
grid.fit(tfidf_sent_vectors_train, y_train)

print("Accuracy on train data = ", grid.best_score_*100)
optimal_est4 = grid.best_estimator_.n_estimators
print("The optimal number of depth is : ",optimal_est4)
print("Time taken: ", datetime.now() - start)
```

```
Accuracy on train data =  50.56224487250075
The optimal number of depth is :  10
Time taken:  0:01:16.246425
```

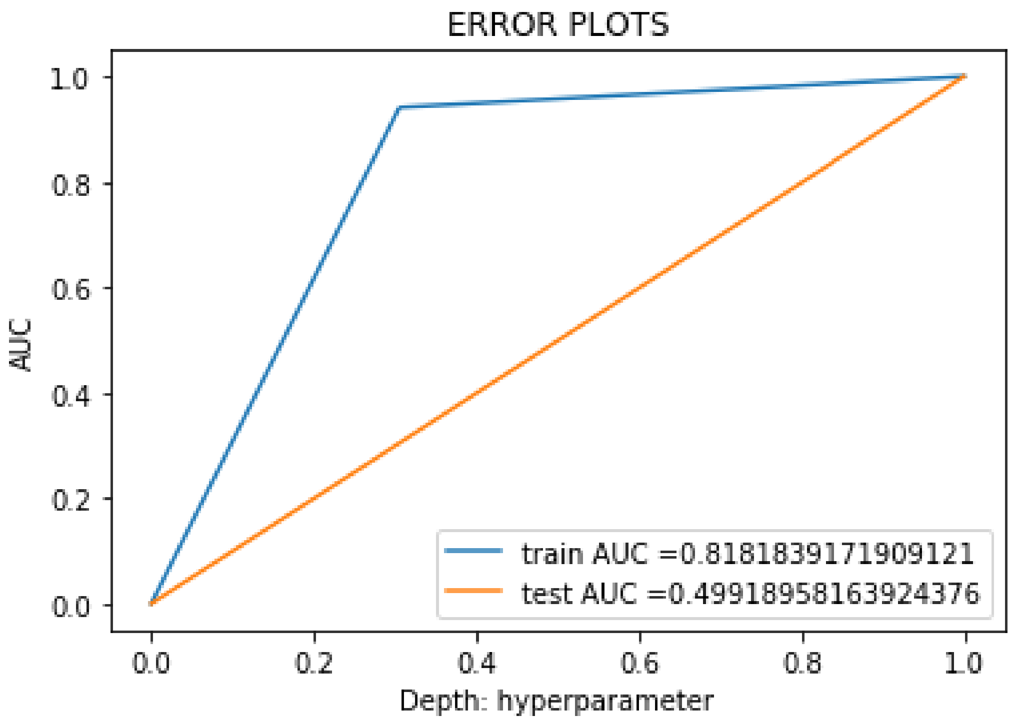
In [129]:
```
train_auc = grid.cv_results_['mean_train_score']
cv_auc = grid.cv_results_['mean_test_score']
```

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

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

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



In [134]:
```
clf = RandomForestClassifier(n_estimators= 1, class_weight ='balanced')
clf.fit(tfidf_sent_vectors_train, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(tfidf_sent_ve
test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(tfidf_sent_vec

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

plt.legend()
plt.xlabel("Depth: hyperparameter")
```

```
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```




```
In [135]: start = datetime.now()
          depth     = [5,10,20,30,40,50,60,70, 80, 90 ]
          estimator = [5,10,20,30,40,50,75,100,125,150]

          parameters = {'n_estimators': estimator,'max_depth': depth}
          grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced'), parameters, cv=
          grid.fit(tfidf_sent_vectors_train, y_train)
          print("Time taken: ", datetime.now() - start)
```

```
Time taken:  1:12:04.250837
```

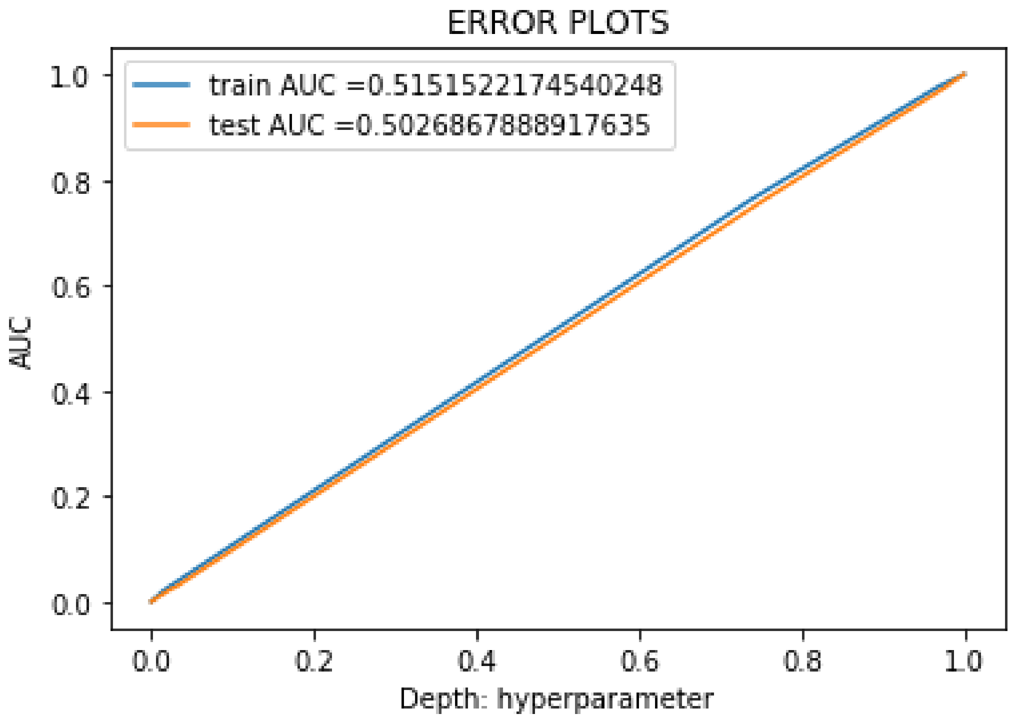
```
In [136]: grid.best_params_
```

```
Out[136]: {'max_depth': 30, 'n_estimators': 100}
```

```
In [139]: optimal_depth4=3
          optimal_est4  =1
```

```
In [138]: clf = RandomForestClassifier(n_estimators= 1, max_depth=3, class_weight ='balanced')
          clf.fit(tfidf_sent_vectors_train, y_train)
```

```
train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(tfidf_sent_ve
test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(tfidf_sent_vec

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

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




```
In [153]: clf = RandomForestClassifier(max_depth = optimal_depth4, n_estimators= optimal_est4,
          clf.fit(tfidf_sent_vectors_train,y_train)
          pred = clf.predict(tfidf_sent_vectors_test)

          acc_4 = accuracy_score(y_test, pred) * 100
          pre_4 = precision_score(y_test, pred) * 100
          rec_4 = recall_score(y_test, pred) * 100
          f1_4  = f1_score(y_test, pred) * 100

          print('\nAccuracy = %f%%' % (acc_4))
          print('\nprecision= %f%%' % (pre_4))
```

```
print('\nrecall    = %f%%' % (rec_4))
print('\nF1-Score = %f%%' % (f1_4))
```

```
Accuracy = 75.436731%

precision= 84.030699%

recall    = 87.435145%

F1-Score = 85.699124%
```

6.6 Heatmap for test data

```
In [154]: heat_map4 = grid.cv_results_['mean_train_score'].reshape(len(estimator),len(depth))
          plt.figure(figsize=(12,8))
          sns.heatmap(heat_map4, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator
          plt.xlabel('n_estimator')
          plt.ylabel('max_depth')
          plt.xticks(np.arange(len(estimator)), estimator)
          plt.yticks(np.arange(len(depth)), depth)
          plt.title('Heatmap of Train Data')
          plt.show()
```

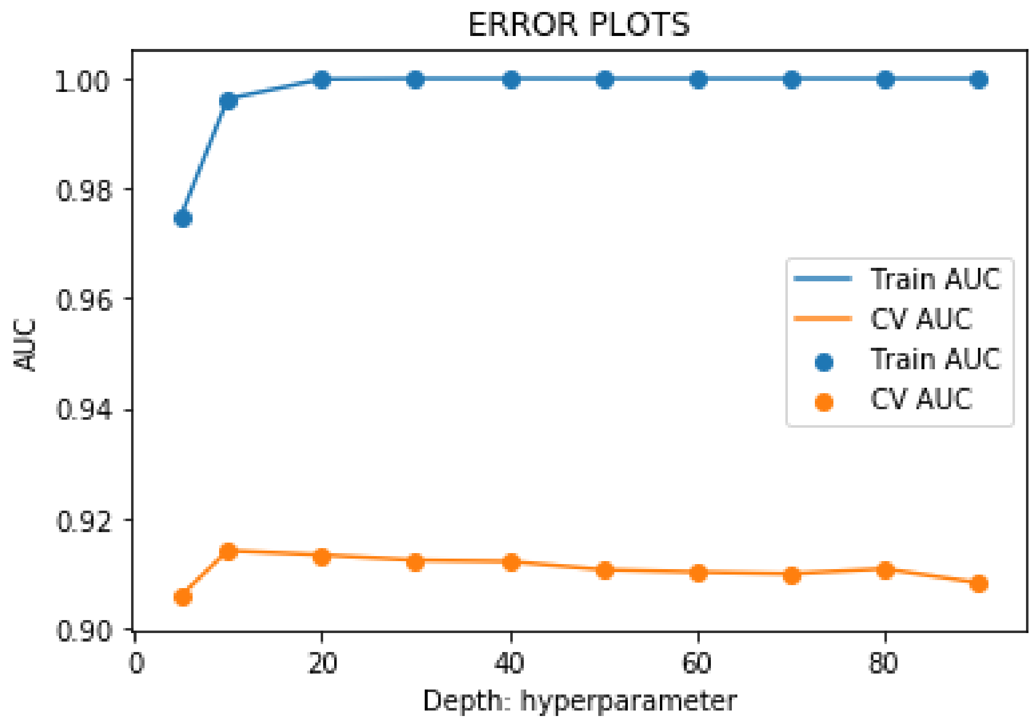
6.7 [5.2] Applying GBDT using XGBOOST

6.7.1 [5.2.1] Applying XGBOOST on BOW, SET 1

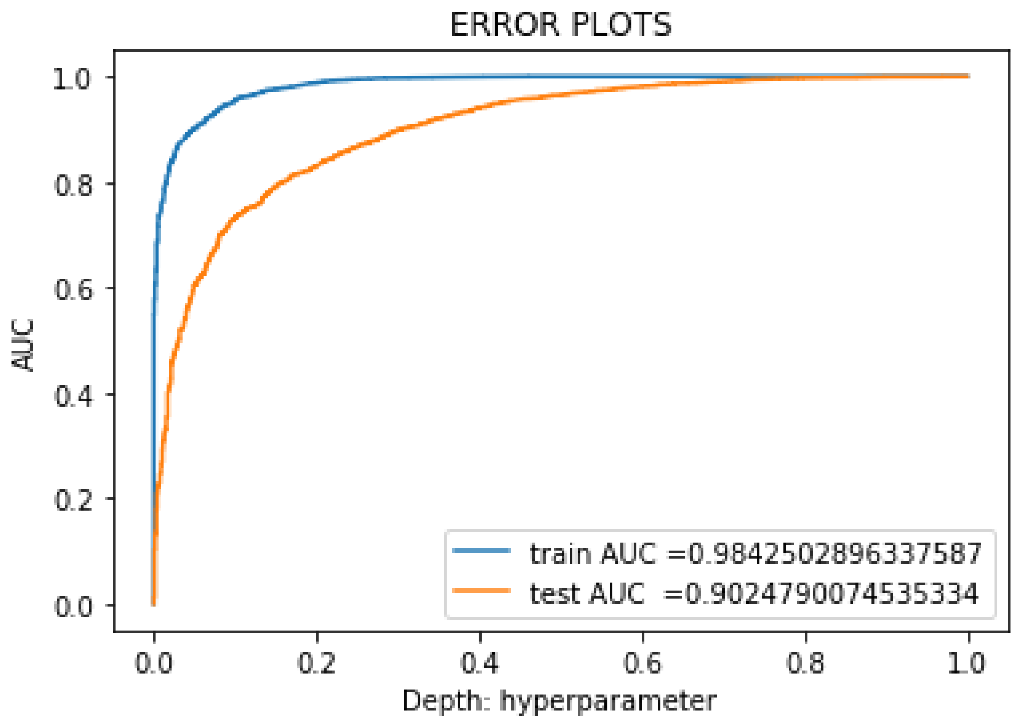
```
In [249]: ############ tuning depth parameter first ###########
          start = datetime.now()
          depth = [5,10,20,30,40,50,60,70,80,90]
          parameters = {'max_depth': depth}
          grid = GridSearchCV(XGBClassifier(booster='gbtree',n_estimators=200), parameters, cv=
          grid.fit(x_train_bow, y_train)

          print("Accuracy on train data = ", grid.best_score_*100)
          optimal_depth5 = grid.best_estimator_.max_depth
          print("The optimal number of depth is : ",optimal_depth5)
          print("Time taken: ", datetime.now() - start)
```

```
Accuracy on train data =  91.41168382532123
The optimal number of depth is :  10
Time taken:  0:12:03.908564
```

```
In [250]: train_auc = grid.cv_results_['mean_train_score']
          cv_auc = grid.cv_results_['mean_test_score']

          plt.plot(depth, train_auc, label='Train AUC')
          plt.scatter(depth, train_auc, label='Train AUC')

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

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




```
In [261]: clf = XGBClassifier(booster='gbtree',max_depth=10)
          clf.fit(x_train_bow, y_train)

          train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_bow)
          test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(x_test_bow)[:

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

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

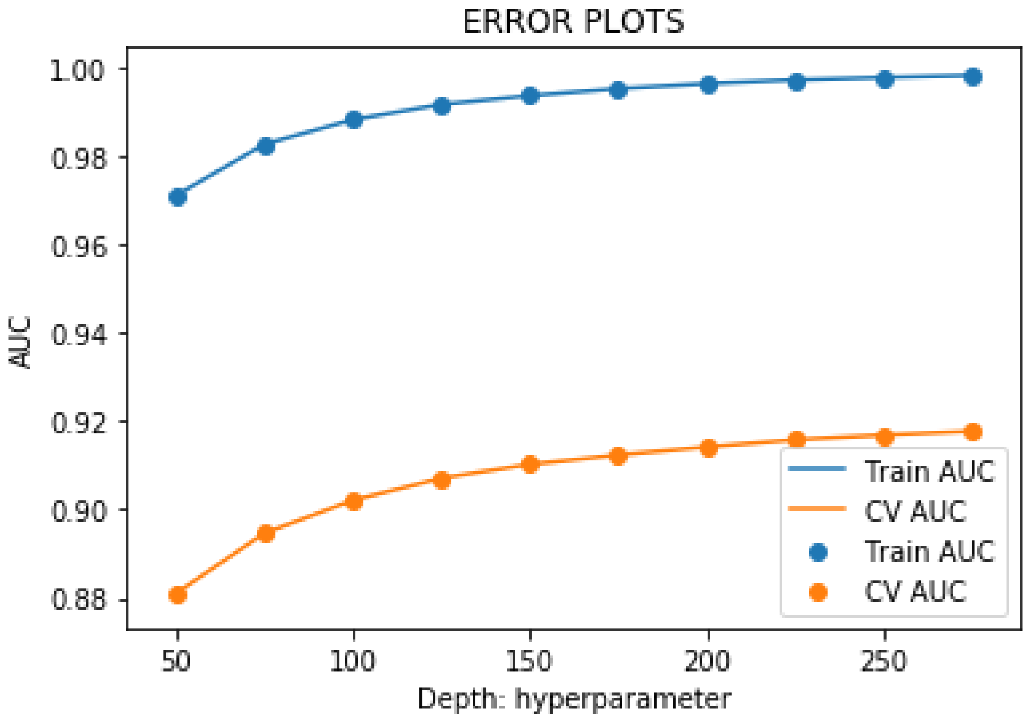



```
In [264]: ######################### tuning for hyperparameter n_estimator ###########

          start = datetime.now()
          estimator = [50,75,100,125,150,175,200,225,250,275]
          parameters = {'n_estimators': estimator}
          grid = GridSearchCV(XGBClassifier(booster='gbtree', class_weight ='balanced',max_dep
          grid.fit(x_train_bow, y_train)

          print("Accuracy on train data = ", grid.best_score_*100)
          optimal_est5 = grid.best_estimator_.n_estimators
          print("The optimal number of depth is : ",optimal_est5)
          print("Time taken: ", datetime.now() - start)
```

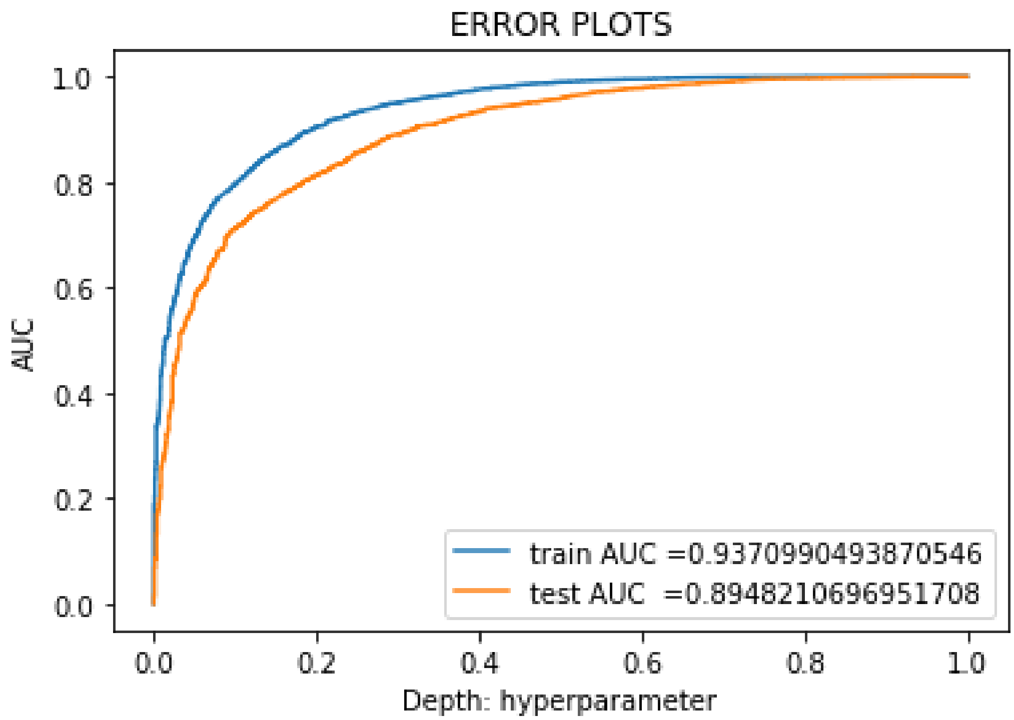
```
Accuracy on train data =  91.76834436807442
The optimal number of depth is :  275
Time taken:  0:03:24.036335
```

```
In [265]: train_auc = grid.cv_results_['mean_train_score']
          cv_auc = grid.cv_results_['mean_test_score']

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

```
plt.plot(estimator, cv_auc, label='CV AUC')
plt.scatter(estimator, cv_auc, label='CV AUC')

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



In [269]:
```
clf = XGBClassifier(booster='gbtree',n_estimators= 200, class_weight ='balanced')
clf.fit(x_train_bow, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_bow)
test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(x_test_bow)[:

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

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




```
In [270]: start = datetime.now()
          depth     = [5,10,20,30,40,50,60,70, 80, 90 ]
          estimator = [50,75,100,125,150,175,200,225,250,275]

          parameters = {'n_estimators': estimator,'max_depth': depth}
          grid = GridSearchCV(XGBClassifier(booster='gbtree',class_weight ='balanced'), parame
          grid.fit(x_train_bow, y_train)
          print("Time taken: ", datetime.now() - start)
```

```
Time taken:  1:32:26.724197
```

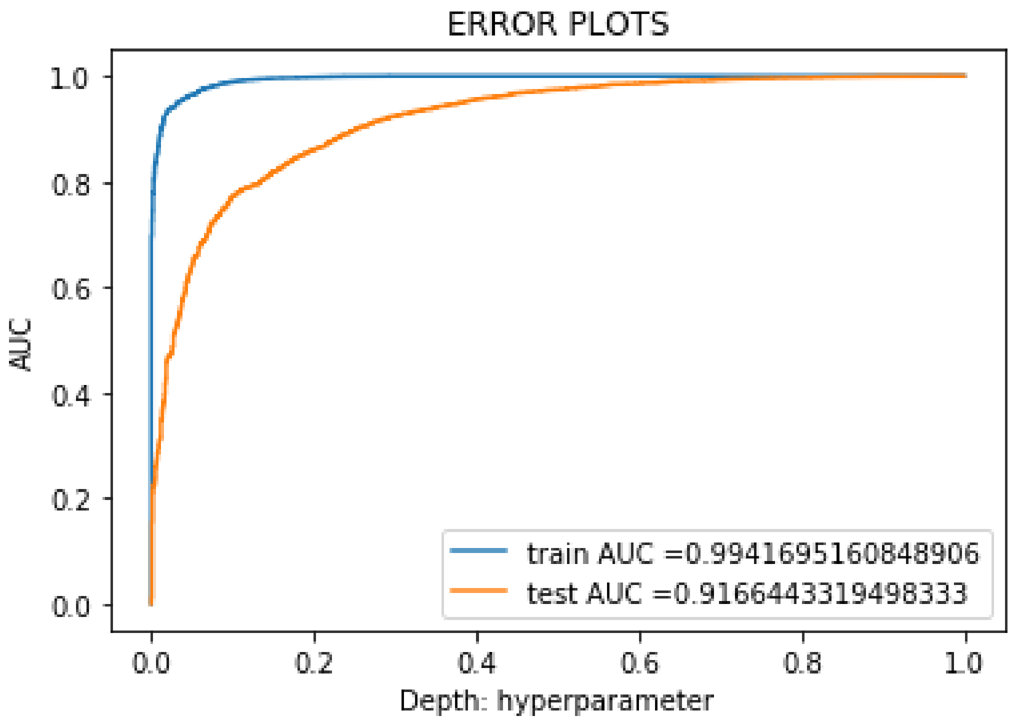
```
In [271]: grid.best_params_
```

```
Out[271]: {'max_depth': 10, 'n_estimators': 275}
```

```
In [272]: optimal_depth5= 10
          optimal_est5  = 200
```

```
In [274]: clf = XGBClassifier(n_estimators= optimal_est5, max_depth=optimal_depth5, class_weig
          clf.fit(x_train_bow, y_train)

          train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_bow)
          test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(x_test_bow)[:
```

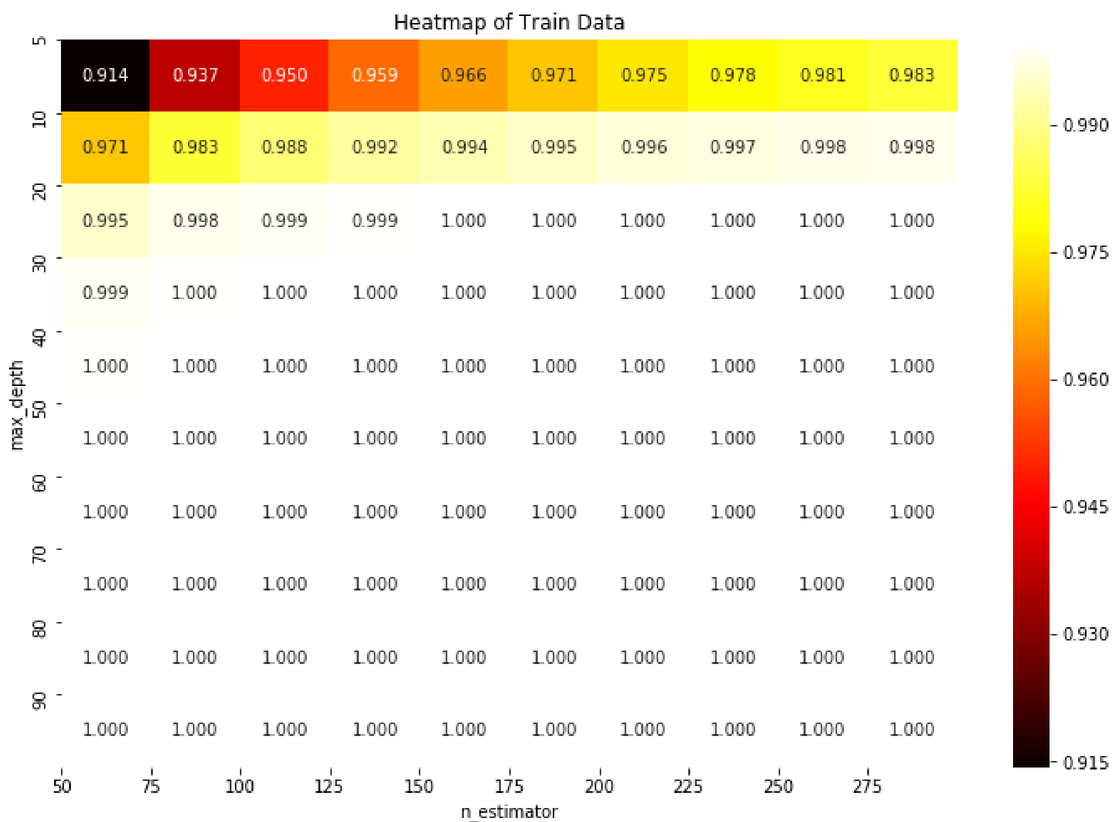
```
plt.plot(train_fpr, train_tpr, label= "train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr,  test_tpr,  label= "test AUC ="+str(auc(test_fpr, test_tpr)))

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



In [359]:
```
clf = XGBClassifier(n_estimators = optimal_est5, max_depth = optimal_depth5)
clf.fit(x_train_bow, y_train)
pred = clf.predict(x_test_bow)

acc_5 = accuracy_score(y_test, pred) * 100
pre_5 = precision_score(y_test, pred) * 100
rec_5 = recall_score(y_test, pred) * 100
f1_5  = f1_score(y_test, pred) * 100

print('\nAccuracy = %f%%' % (acc_5))
print('\nprecision= %f%%' % (pre_5))
print('\nrecall    = %f%%' % (rec_5))
print('\nF1-Score = %f%%' % (f1_5))
```

```
Accuracy = 89.539302%

precision= 90.293160%

recall    = 98.075290%

F1-Score = 94.023472%
```

```
In [275]: heat_map5 = grid.cv_results_['mean_train_score'].reshape(len(estimator),len(depth))
          plt.figure(figsize=(12,8))
          sns.heatmap(heat_map5, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator
          plt.xlabel('n_estimator')
          plt.ylabel('max_depth')
          plt.xticks(np.arange(len(estimator)), estimator)
          plt.yticks(np.arange(len(depth)), depth)
          plt.title('Heatmap of Train Data')
          plt.show()
```

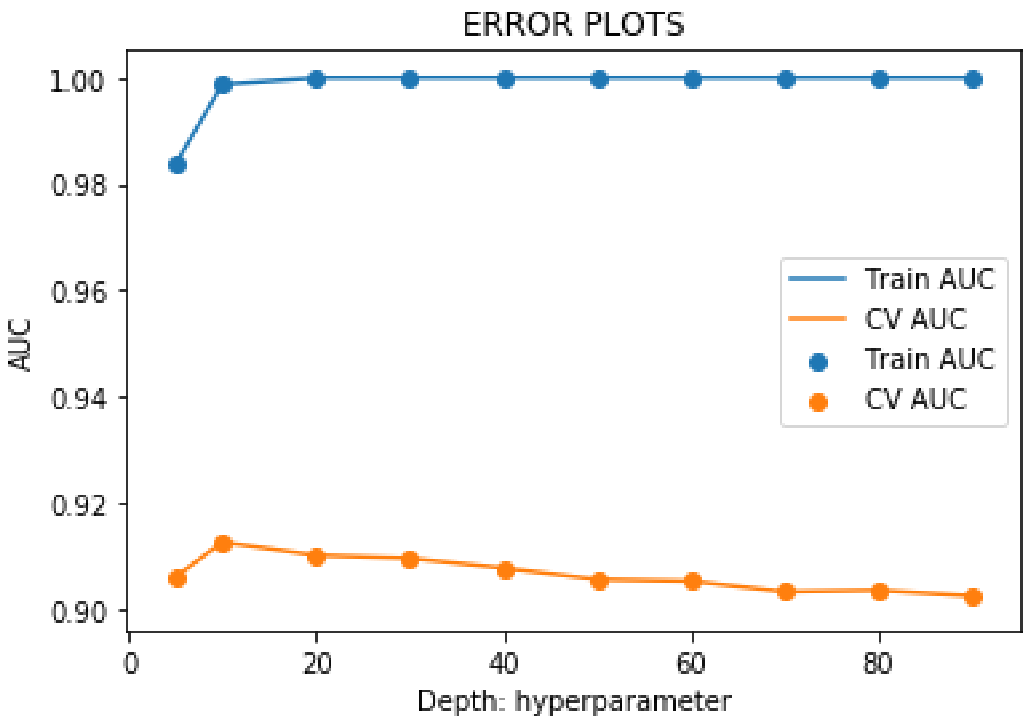


6.7.2 [5.2.2] Applying XGBOOST on TFIDF, SET 2

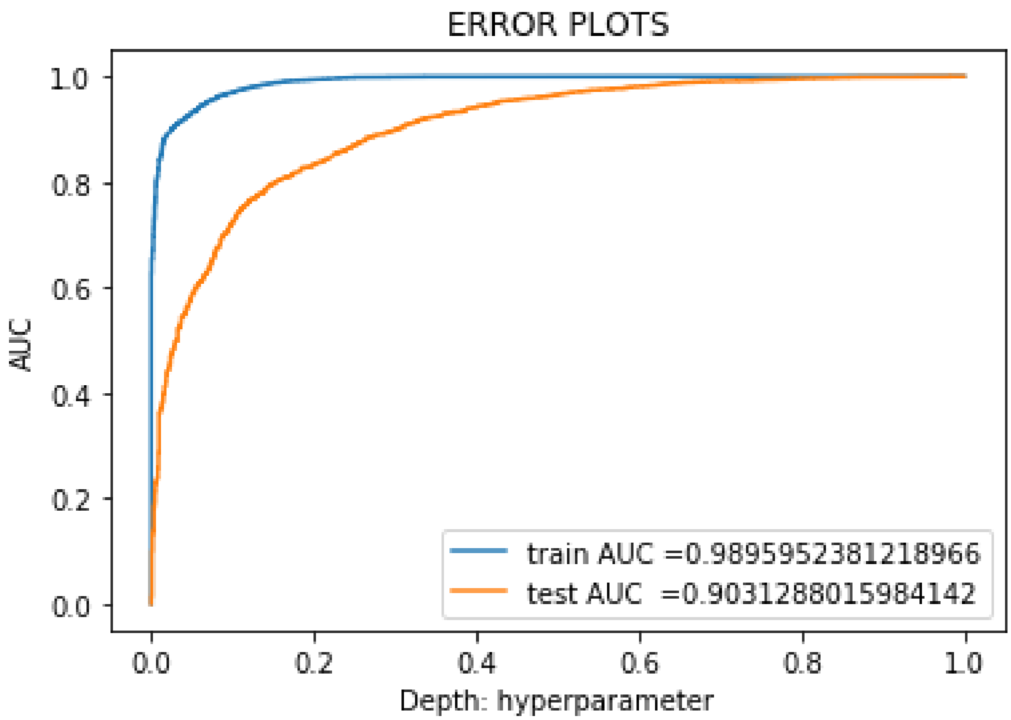
```
In [276]: ############ tuning depth parameter first ###########
          start = datetime.now()
          depth = [5,10,20,30,40,50,60,70,80,90]
          parameters = {'max_depth': depth}
          grid = GridSearchCV(XGBClassifier(booster='gbtree',n_estimators=200), parameters, cv=
          grid.fit(x_train_tf, y_train)

          print("Accuracy on train data = ", grid.best_score_*100)
          optimal_depth6 = grid.best_estimator_.max_depth
          print("The optimal number of depth is : ",optimal_depth6)
          print("Time taken: ", datetime.now() - start)
```

```
Accuracy on train data =  91.26814329849708
The optimal number of depth is :  10
Time taken:  0:17:34.448552
```

```
In [277]: train_auc = grid.cv_results_['mean_train_score']
          cv_auc = grid.cv_results_['mean_test_score']

          plt.plot(depth, train_auc, label='Train AUC')
          plt.scatter(depth, train_auc, label='Train AUC')

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

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



In [279]:
```
clf = XGBClassifier(booster='gbtree',max_depth=optimal_depth6)
clf.fit(x_train_tf, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_tf)[
test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(x_test_tf)[:,

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

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

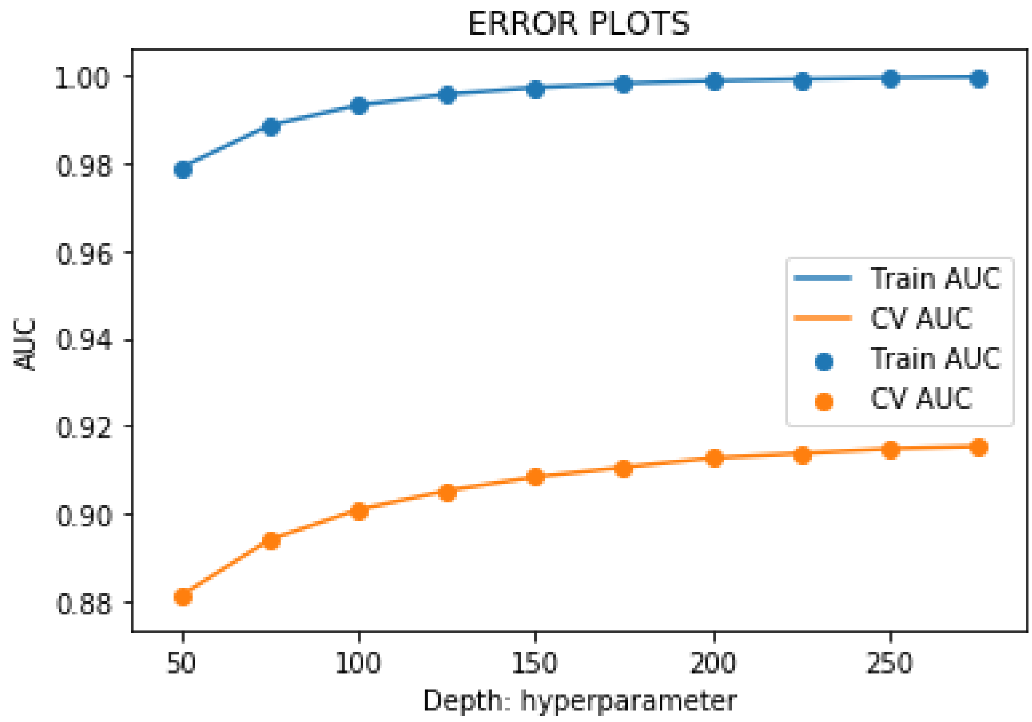


In [280]:
```
########################## tuning for hyperparameter n_estimator ###########

start = datetime.now()
estimator = [50,75,100,125,150,175,200,225,250,275]
parameters = {'n_estimators': estimator}
grid = GridSearchCV(XGBClassifier(booster='gbtree',max_depth = 10), parameters, cv=3
grid.fit(x_train_tf, y_train)

print("Accuracy on train data = ", grid.best_score_*100)
optimal_est6 = grid.best_estimator_.n_estimators
print("The optimal number of depth is : ",optimal_est6)
print("Time taken: ", datetime.now() - start)
```

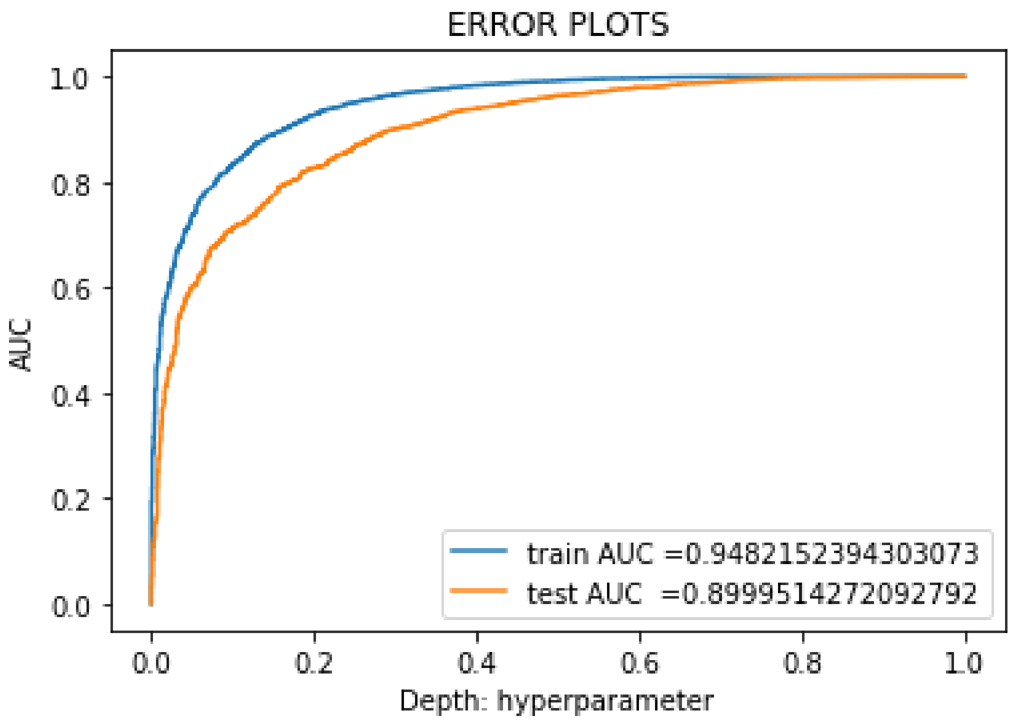
```
Accuracy on train data =  91.52744093723638
The optimal number of depth is :  275
Time taken:  0:06:46.233185
```

In [281]:
```
train_auc = grid.cv_results_['mean_train_score']
cv_auc = grid.cv_results_['mean_test_score']

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

```
plt.plot(estimator, cv_auc, label='CV AUC')
plt.scatter(estimator, cv_auc, label='CV AUC')

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

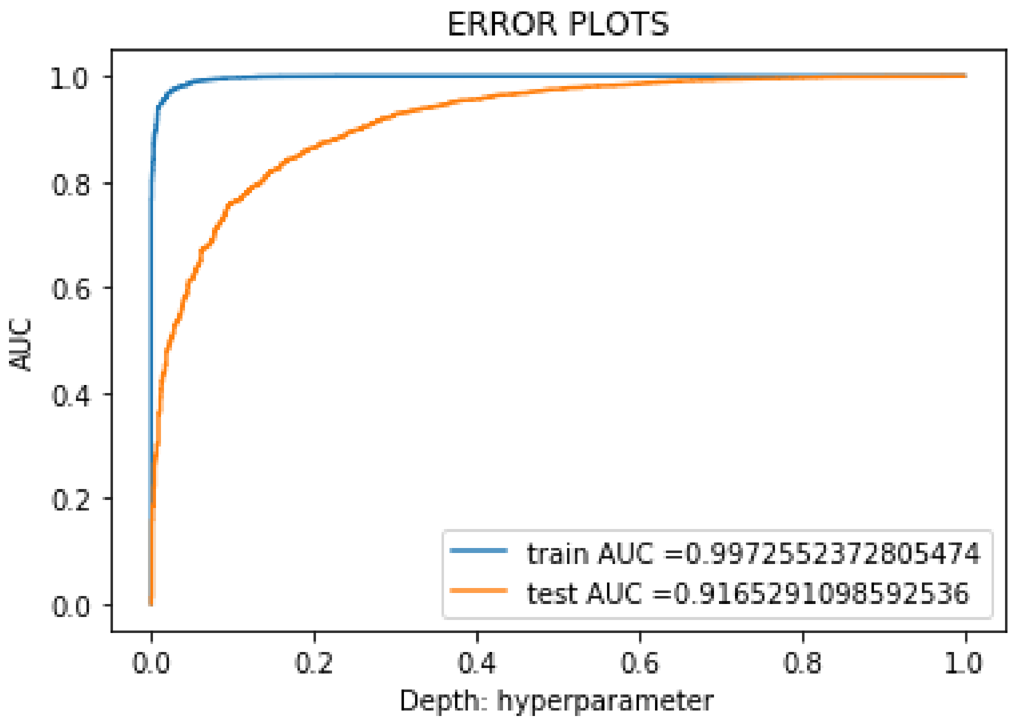


In [284]:
```
clf = XGBClassifier(booster='gbtree',n_estimators=200)
clf.fit(x_train_tf, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_tf)[
test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(x_test_tf)[:,

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

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




```
In [285]: start = datetime.now()
          depth     = [5,10,20,30,40,50,60,70, 80, 90 ]
          estimator = [50,75,100,125,150,175,200,225,250,275]

          parameters = {'n_estimators': estimator,'max_depth': depth}
          grid = GridSearchCV(XGBClassifier(booster='gbtree',class_weight ='balanced'), parame
          grid.fit(x_train_tf, y_train)
          print("Time taken: ", datetime.now() - start)

Time taken:  2:16:10.437806


In [286]: grid.best_params_

Out[286]: {'max_depth': 10, 'n_estimators': 275}

In [287]: optimal_depth6 = 10
          optimal_est6   = 200

In [290]: clf  = XGBClassifier(n_estimators= 200, max_depth=optimal_depth6, class_weight ='balan
          clf.fit(x_train_tf, y_train)

          train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(x_train_tf)[
          test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(x_test_tf)[:,
```

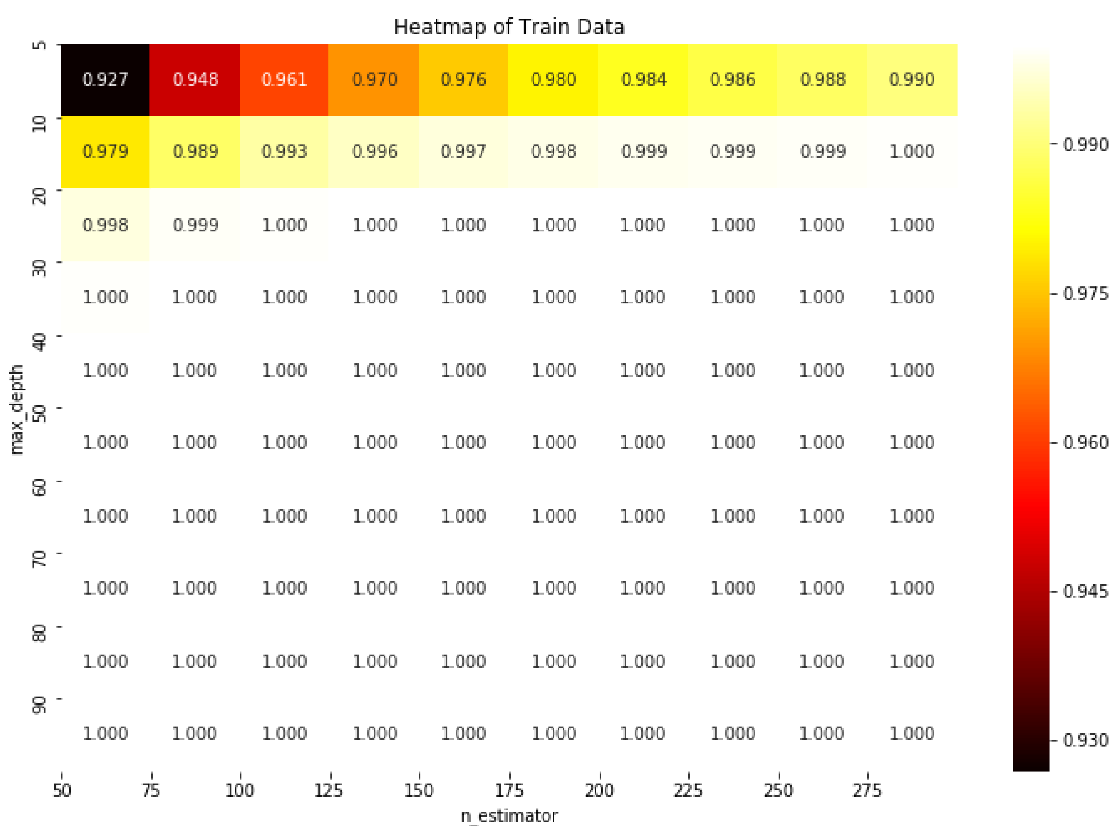
```
plt.plot(train_fpr, train_tpr, label= "train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr,  test_tpr,  label= "test AUC ="+str(auc(test_fpr, test_tpr)))

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




```
In [358]: clf = XGBClassifier(n_estimators = optimal_est6, max_depth = optimal_depth6)
          clf.fit(x_train_tf, y_train)
          pred = clf.predict(x_test_tf)

          acc_6 = accuracy_score(y_test, pred) * 100
          pre_6 = precision_score(y_test, pred) * 100
          rec_6 = recall_score(y_test, pred) * 100
          f1_6  = f1_score(y_test, pred) * 100

          print('\nAccuracy = %f%%' % (acc_6))
          print('\nprecision= %f%%' % (pre_6))
          print('\nrecall    = %f%%' % (rec_6))
          print('\nF1-Score = %f%%' % (f1_6))
```

```
Accuracy = 89.266208%

precision= 89.816490%

recall    = 98.358336%

F1-Score = 93.893542%
```

```
In [291]: heat_map5 = grid.cv_results_['mean_train_score'].reshape(len(estimator),len(depth))
          plt.figure(figsize=(12,8))
          sns.heatmap(heat_map5, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator
          plt.xlabel('n_estimator')
          plt.ylabel('max_depth')
          plt.xticks(np.arange(len(estimator)), estimator)
          plt.yticks(np.arange(len(depth)), depth)
          plt.title('Heatmap of Train Data')
          plt.show()
```

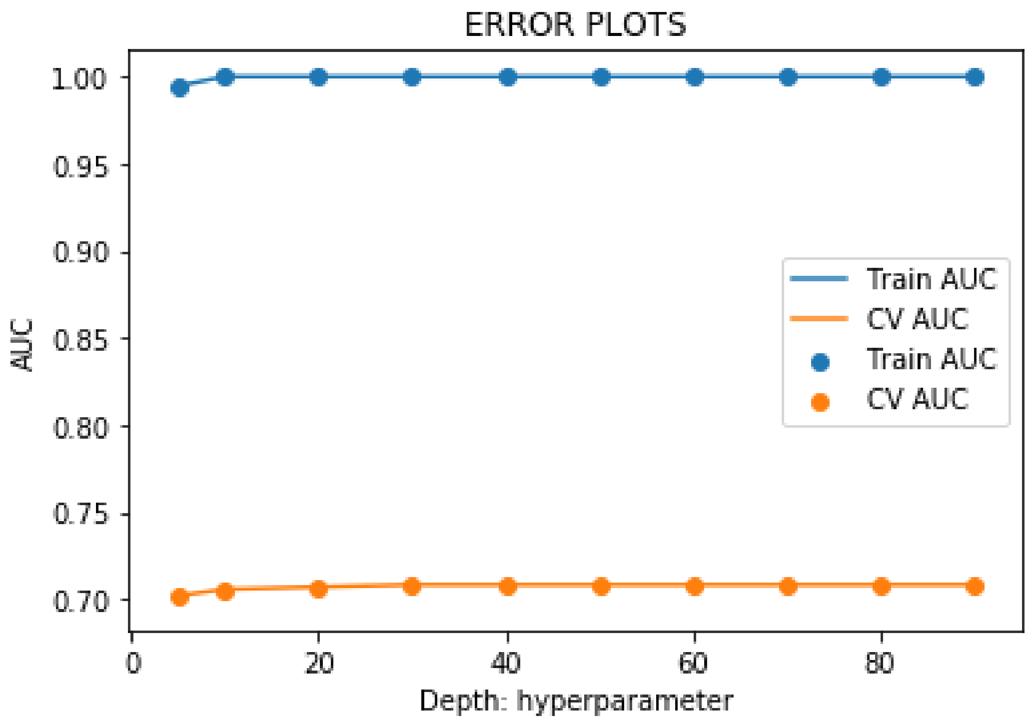


6.7.3 [5.2.3] Applying XGBOOST on AVG W2V, SET 3

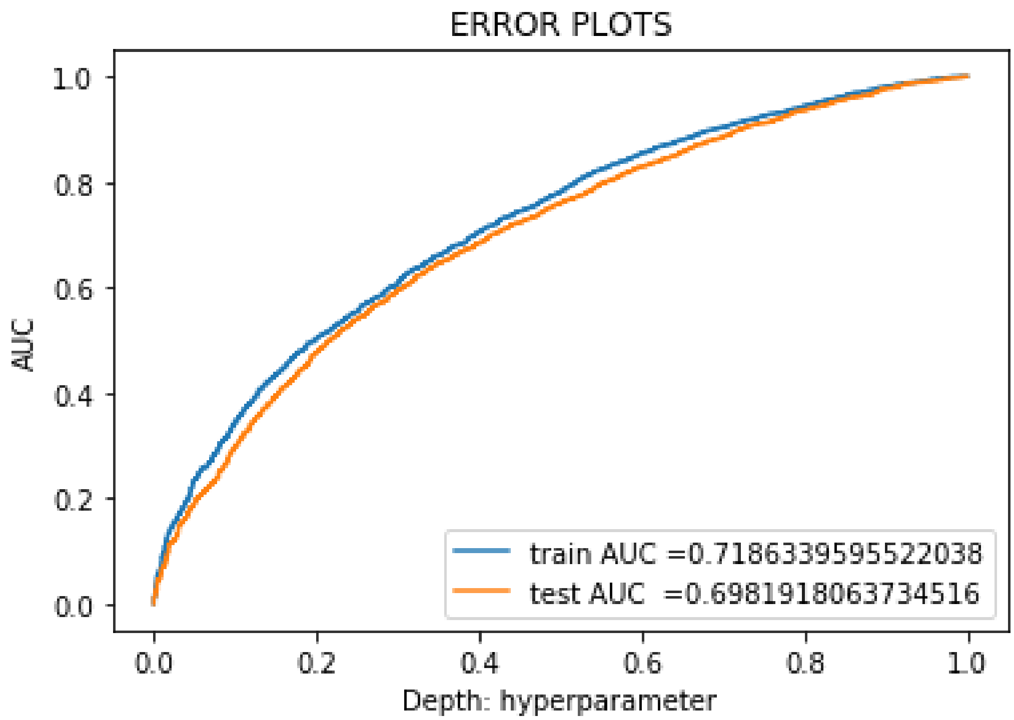
```
In [292]: ############ tuning depth parameter first ###########
          start = datetime.now()
          depth = [5,10,20,30,40,50,60,70,80,90]
          parameters = {'max_depth': depth}
          grid = GridSearchCV(XGBClassifier(booster='gbtree',n_estimators=200), parameters, cv=
          grid.fit(sent_vectors_train, y_train)

          print("Accuracy on train data = ", grid.best_score_*100)
          optimal_depth7 = grid.best_estimator_.max_depth
          print("The optimal number of depth is : ",optimal_depth7)
          print("Time taken: ", datetime.now() - start)
```

```
Accuracy on train data =  70.84431485262648
The optimal number of depth is :  30
Time taken:  0:08:12.008477
```

```
In [293]: train_auc = grid.cv_results_['mean_train_score']
          cv_auc = grid.cv_results_['mean_test_score']

          plt.plot(depth, train_auc, label='Train AUC')
          plt.scatter(depth, train_auc, label='Train AUC')

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

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




```
In [297]: clf = XGBClassifier(booster='gbtree',max_depth=1)
          clf.fit(sent_vectors_train, y_train)

          train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(sent_vectors_
          test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(sent_vectors_t

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

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

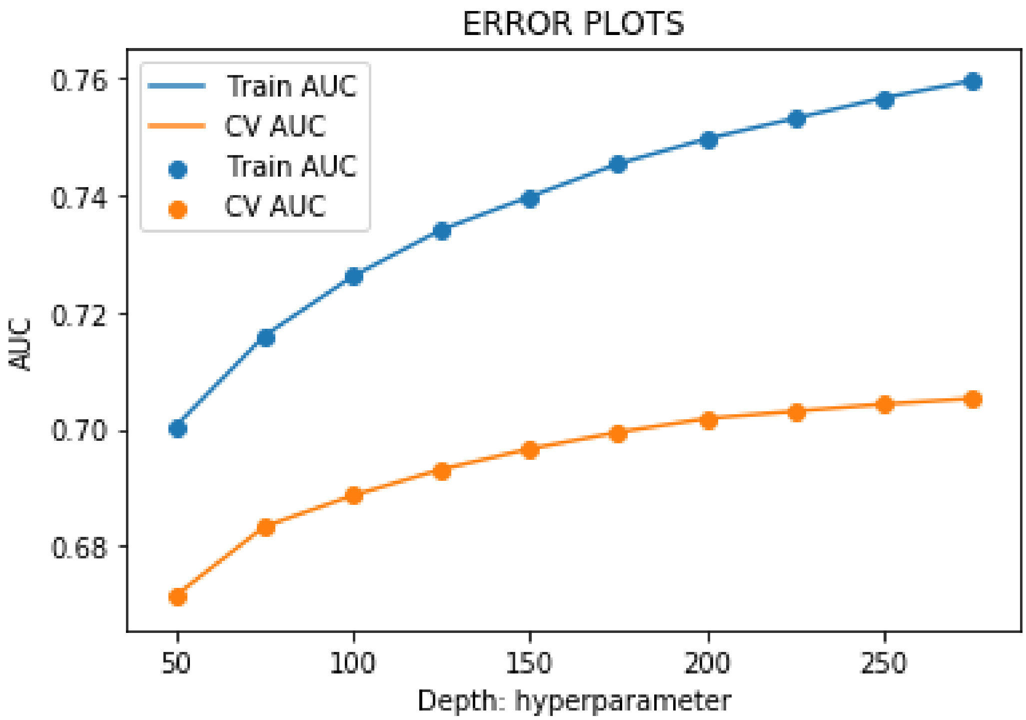


In [298]:
```
########################## tuning for hyperparameter n_estimator ###########

start = datetime.now()
estimator = [50,75,100,125,150,175,200,225,250,275]
parameters = {'n_estimators': estimator}
grid = GridSearchCV(XGBClassifier(booster='gbtree', class_weight ='balanced',max_dep
grid.fit(sent_vectors_train, y_train)

print("Accuracy on train data = ", grid.best_score_*100)
optimal_est7 = grid.best_estimator_.n_estimators
print("The optimal number of depth is : ",optimal_est7)
print("Time taken: ", datetime.now() - start)
```

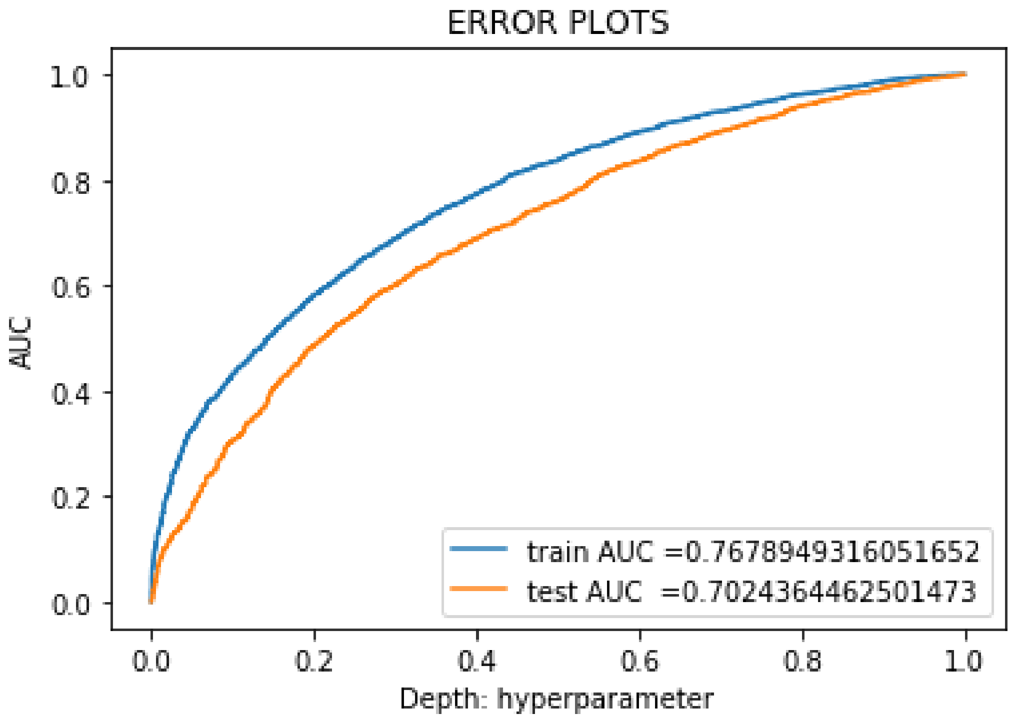
```
Accuracy on train data =  70.52117119204024
The optimal number of depth is :  275
Time taken:  0:00:41.546684
```

In [300]:
```
train_auc = grid.cv_results_['mean_train_score']
cv_auc = grid.cv_results_['mean_test_score']

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

```
plt.plot(estimator, cv_auc, label='CV AUC')
plt.scatter(estimator, cv_auc, label='CV AUC')

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

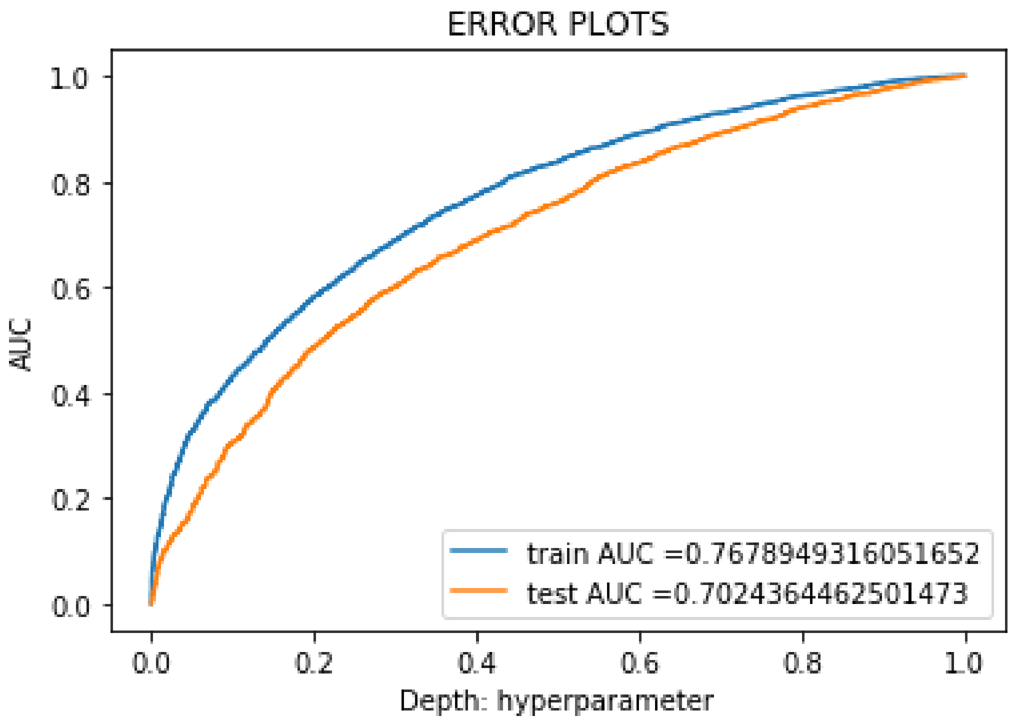


In [307]:
```
clf = XGBClassifier(booster='gbtree',n_estimators=40)
clf.fit(sent_vectors_train, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(sent_vectors_
test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(sent_vectors_t

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

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




```
In [308]: start = datetime.now()
          depth     = [5, 10,20,30,40, 50, 60, 70, 80, 90 ]
          estimator = [30,40,50,75,100,125,150,175,200,225]

          parameters = {'n_estimators': estimator,'max_depth': depth}
          grid = GridSearchCV(XGBClassifier(booster='gbtree',class_weight ='balanced'), paramet
          grid.fit(sent_vectors_train, y_train)
          print("Time taken: ", datetime.now() - start)

Time taken:  0:45:41.868439


In [309]: grid.best_params_

Out[309]: {'max_depth': 30, 'n_estimators': 225}

In [324]: optimal_depth7= 3
          optimal_est7  = 40

In [325]: clf = XGBClassifier(n_estimators= optimal_est7, max_depth=optimal_depth7, class_weig
          clf.fit(sent_vectors_train, y_train)

          train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(sent_vectors_
          test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(sent_vectors_
```

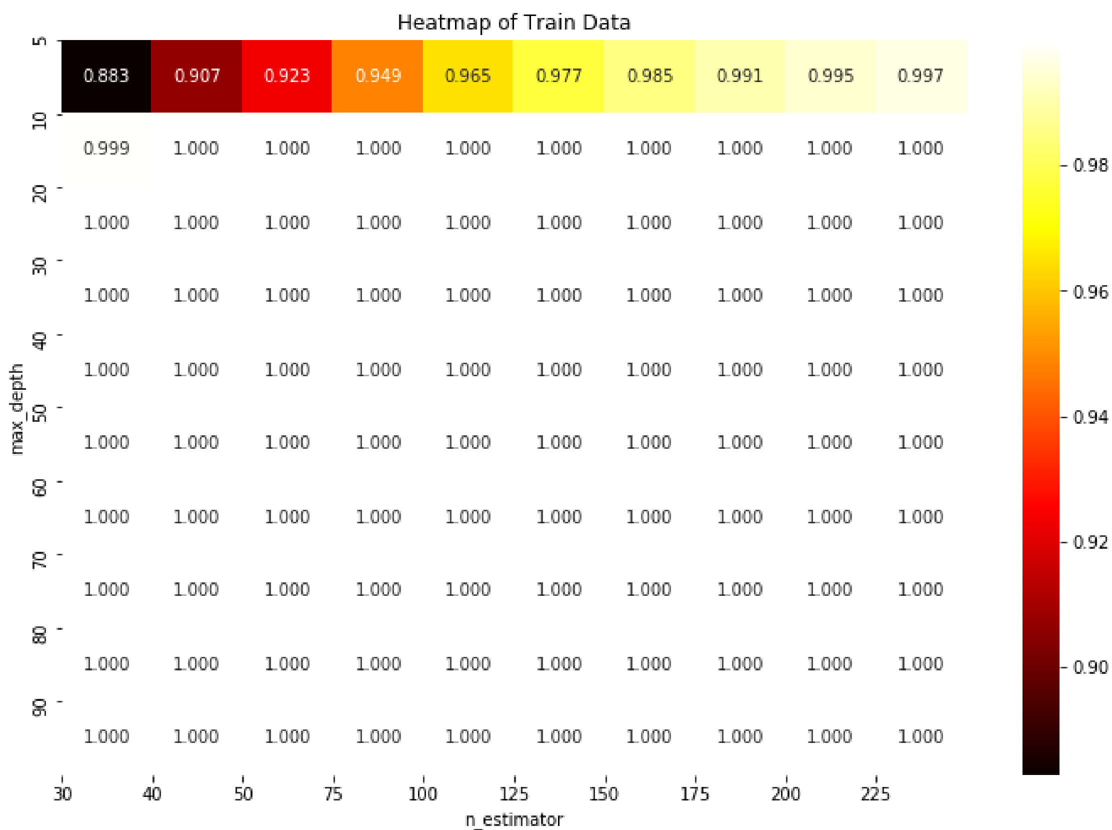
```
plt.plot(train_fpr, train_tpr, label= "train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr,  test_tpr,  label= "test AUC ="+str(auc(test_fpr, test_tpr)))

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



In [357]:
```
clf = XGBClassifier(n_estimators = optimal_est7, max_depth = optimal_depth7)
clf.fit(sent_vectors_train, y_train)
pred = clf.predict(sent_vectors_test)

acc_7 = accuracy_score(y_test, pred) * 100
pre_7 = precision_score(y_test, pred) * 100
rec_7 = recall_score(y_test, pred) * 100
f1_7  = f1_score(y_test, pred) * 100

print('\nAccuracy = %f%%' % (acc_7))
print('\nprecision= %f%%' % (pre_7))
print('\nrecall    = %f%%' % (rec_7))
print('\nF1-Score = %f%%' % (f1_7))
```

```
Accuracy = 83.899311%

precision= 83.899311%

recall    = 100.000000%

F1-Score = 91.244835%
```

```
In [326]: heat_map7 = grid.cv_results_['mean_train_score'].reshape(len(estimator),len(depth))
          plt.figure(figsize=(12,8))
          sns.heatmap(heat_map7, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator
          plt.xlabel('n_estimator')
          plt.ylabel('max_depth')
          plt.xticks(np.arange(len(estimator)), estimator)
          plt.yticks(np.arange(len(depth)), depth)
          plt.title('Heatmap of Train Data')
          plt.show()
```

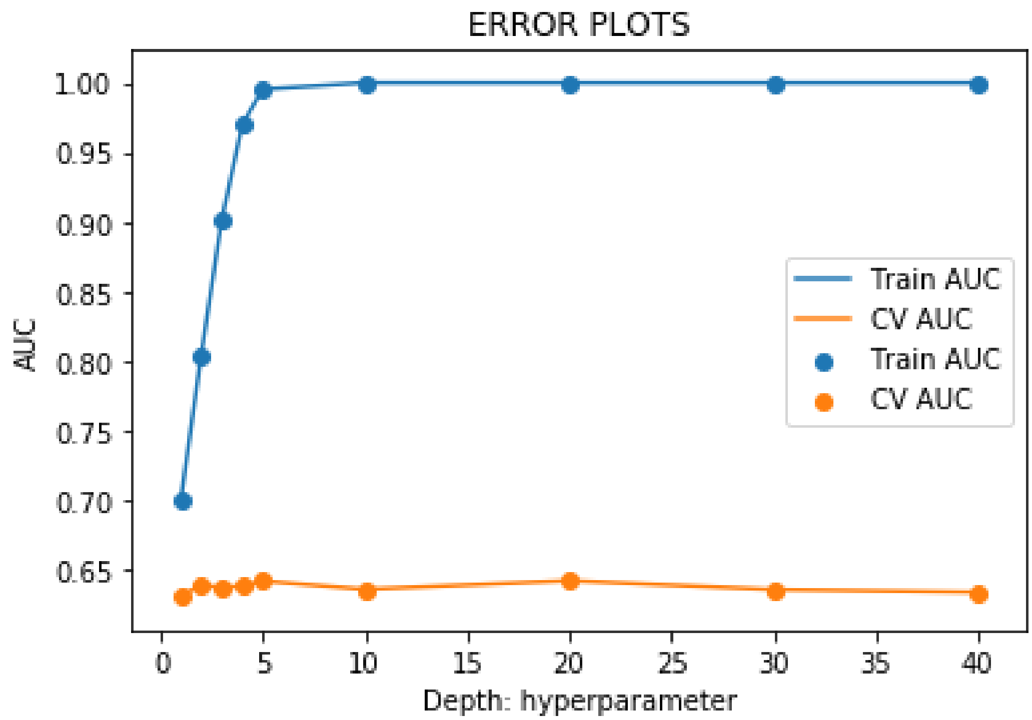


6.7.4 [5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

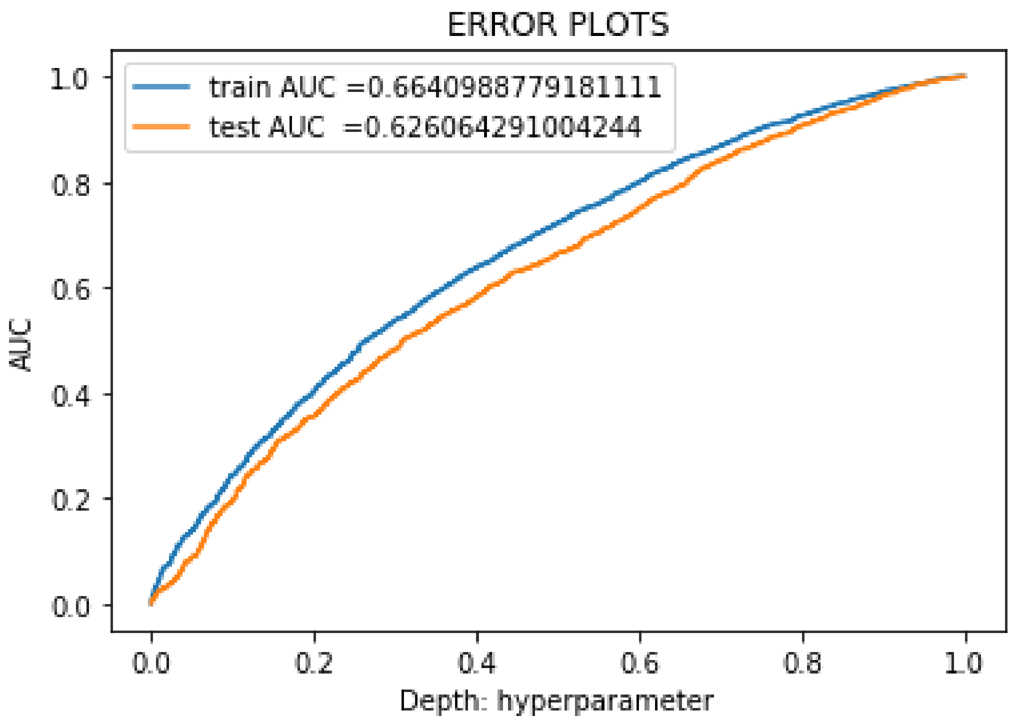
In [334]:
```
############ tuning depth parameter first ###########
start = datetime.now()
depth = [1,2,3,4,5,10,20,30,40,50]
parameters = {'max_depth': depth}
grid = GridSearchCV(XGBClassifier(booster='gbtree',n_estimators=200), parameters, cv=
grid.fit(tfidf_sent_vectors_train, y_train)

print("Accuracy on train data = ", grid.best_score_*100)
optimal_depth8 = grid.best_estimator_.max_depth
print("The optimal number of depth is : ",optimal_depth8)
print("Time taken: ", datetime.now() - start)
```

```
Accuracy on train data =  64.15613529318219
The optimal number of depth is :  20
Time taken:  0:05:08.582615
```

In [335]:
```
train_auc = grid.cv_results_['mean_train_score']
cv_auc = grid.cv_results_['mean_test_score']

plt.plot(depth, train_auc, label='Train AUC')
plt.scatter(depth, train_auc, label='Train AUC')

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

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



In [344]:
```
clf = XGBClassifier(booster='gbtree',max_depth=1)
clf.fit(tfidf_sent_vectors_train, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(tfidf_sent_ve
test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(tfidf_sent_vec

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

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

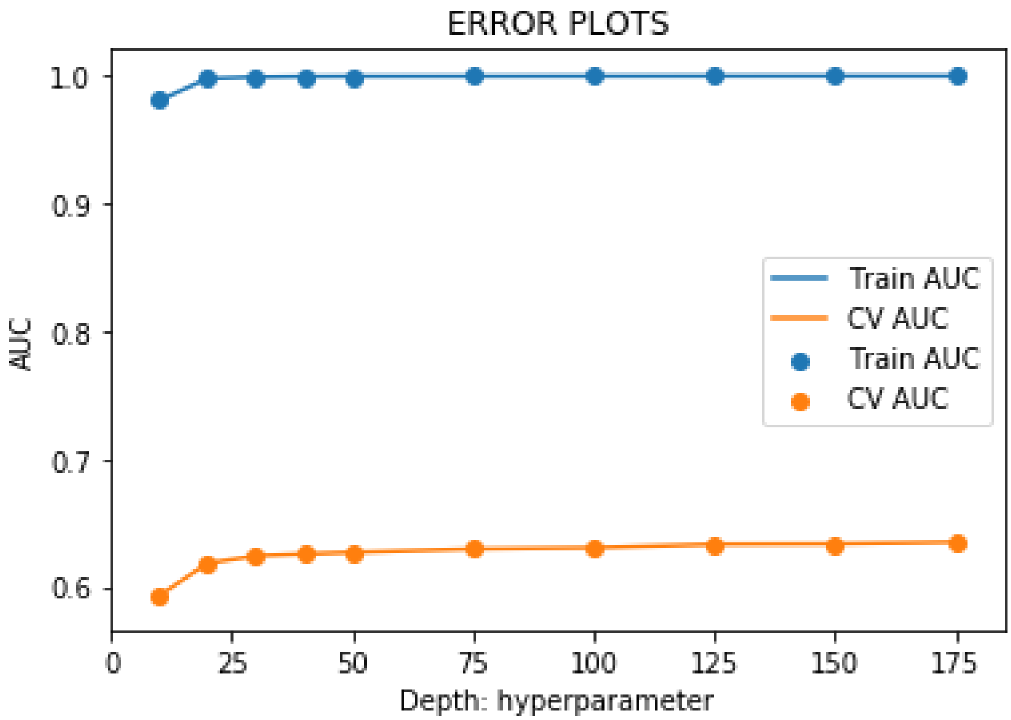


In [345]:
```
########################## tuning for hyperparameter n_estimator ###########

start = datetime.now()
estimator = [10,20,30,40,50,75,100,125,150,175]
parameters = {'n_estimators': estimator}
grid = GridSearchCV(XGBClassifier(booster='gbtree', class_weight ='balanced',max_dep
grid.fit(tfidf_sent_vectors_train, y_train)

print("Accuracy on train data = ", grid.best_score_*100)
optimal_est8 = grid.best_estimator_.n_estimators
print("The optimal number of depth is : ",optimal_est)
print("Time taken: ", datetime.now() - start)
```

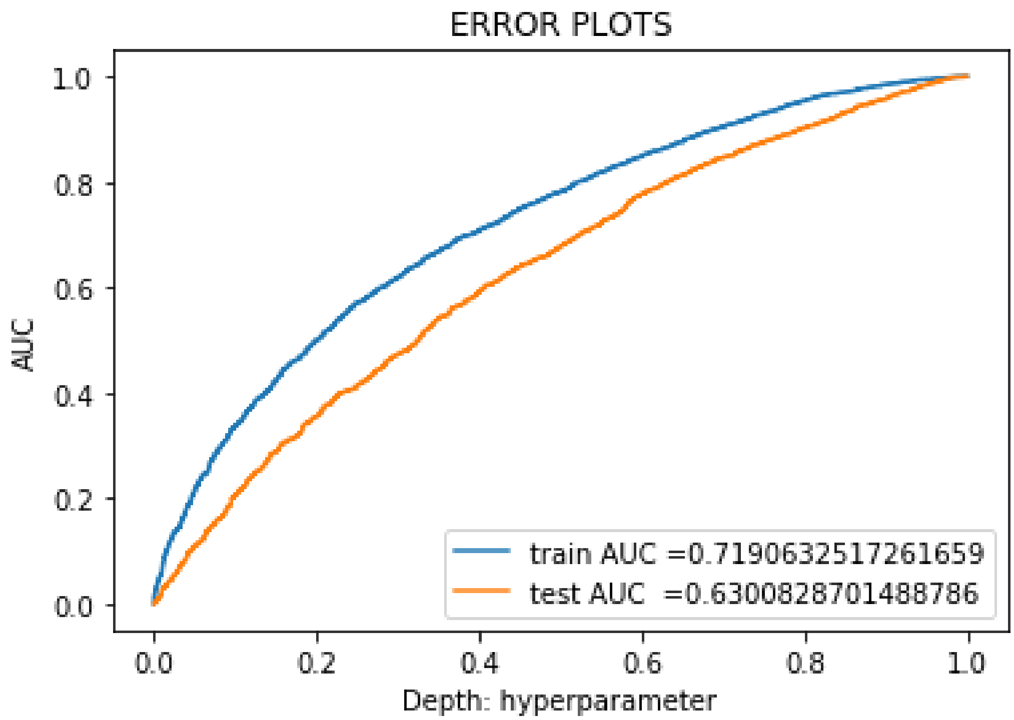
```
Accuracy on train data =  63.488546299093926
The optimal number of depth is :  175
Time taken:  0:03:25.637023
```

In [346]:
```
train_auc = grid.cv_results_['mean_train_score']
cv_auc = grid.cv_results_['mean_test_score']

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

```
plt.plot(estimator, cv_auc, label='CV AUC')
plt.scatter(estimator, cv_auc, label='CV AUC')

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



In [350]:
```
clf = XGBClassifier(booster='gbtree',n_estimators=30)
clf.fit(tfidf_sent_vectors_train, y_train)

train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(tfidf_sent_ve
test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(tfidf_sent_vec

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

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




```
In [351]: start = datetime.now()
          depth     = [1, 2, 3, 4, 5, 10, 20,30, 40, 50]
          estimator = [10,20,30,40,50,75,100,125,150,175]

          parameters = {'n_estimators': estimator,'max_depth': depth}
          grid = GridSearchCV(XGBClassifier(booster='gbtree',class_weight ='balanced'), paramet
          grid.fit(tfidf_sent_vectors_train, y_train)
          print("Time taken: ", datetime.now() - start)
```

```
Time taken:  0:23:23.038941
```

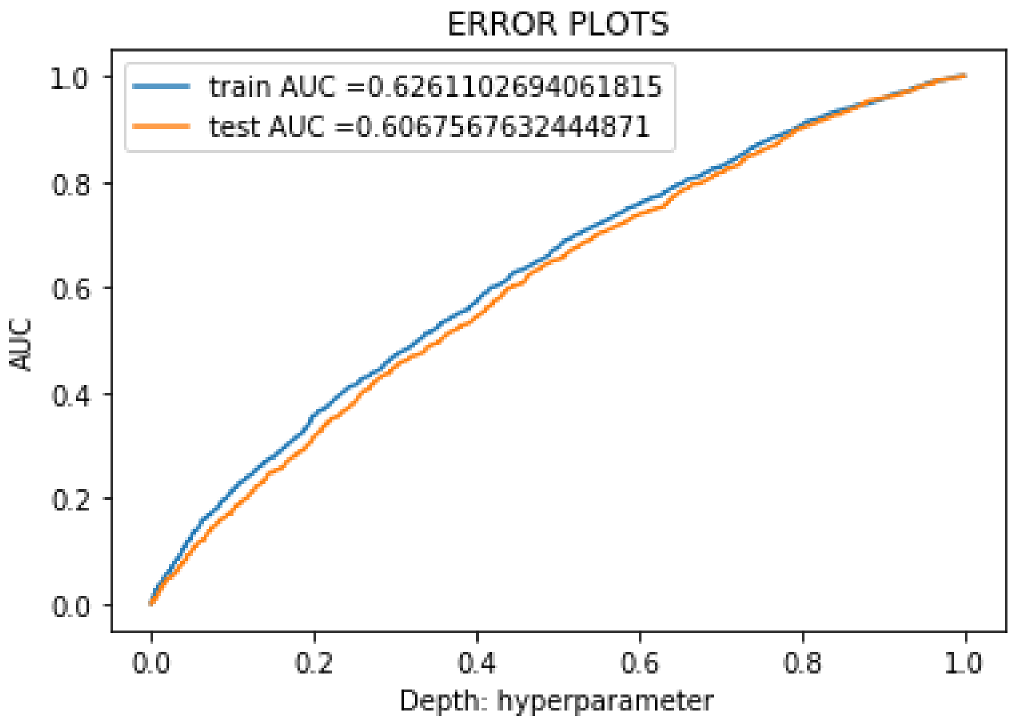
```
In [352]: grid.best_params_
```

```
Out[352]: {'max_depth': 5, 'n_estimators': 175}
```

```
In [353]: optimal_depth8 = 1
          optimal_est8   = 30
```

```
In [354]: clf = XGBClassifier(n_estimators= optimal_est8, max_depth=optimal_depth8, class_weig
          clf.fit(tfidf_sent_vectors_train, y_train)

          train_fpr, train_tpr, thresholds = roc_curve(y_train, clf.predict_proba(tfidf_sent_ve
          test_fpr,  test_tpr,  thresholds = roc_curve(y_test, clf.predict_proba(tfidf_sent_vec
```

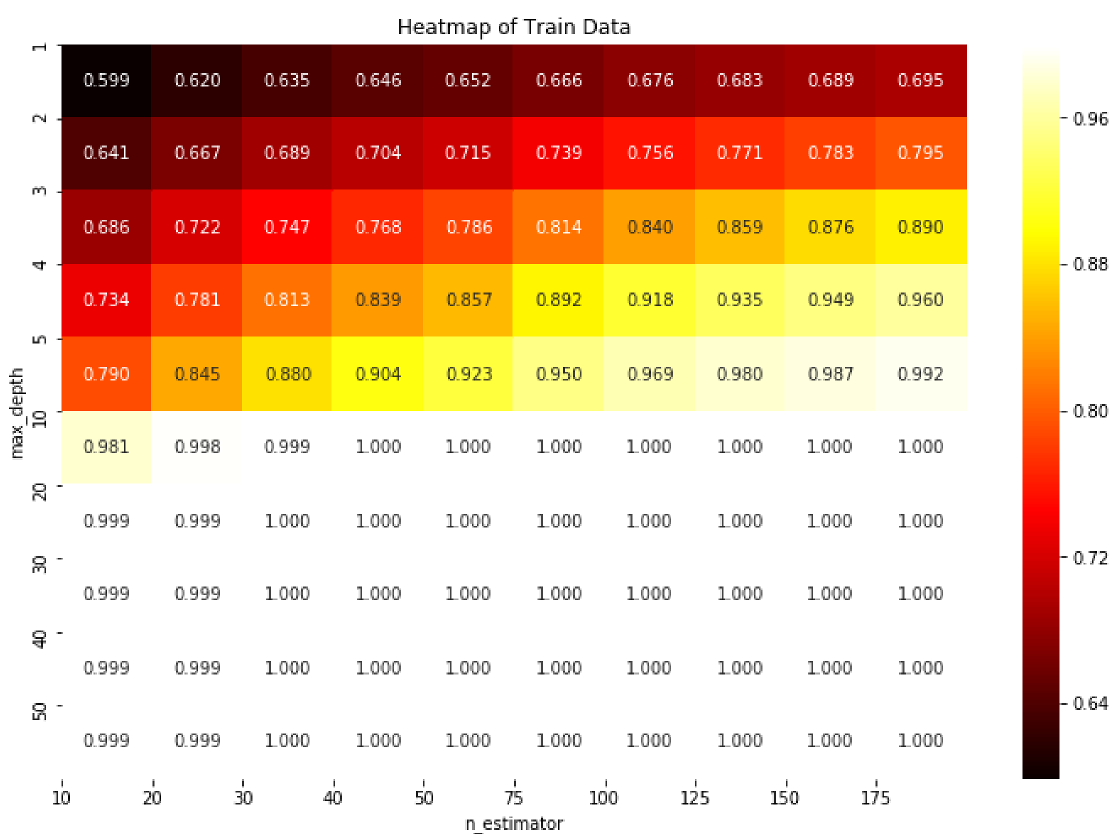
```
plt.plot(train_fpr, train_tpr, label= "train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr,  test_tpr,  label= "test AUC ="+str(auc(test_fpr, test_tpr)))

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



In [355]:
```
clf = XGBClassifier(n_estimators = optimal_est8, max_depth = optimal_depth8)
clf.fit(tfidf_sent_vectors_train, y_train)
pred = clf.predict(tfidf_sent_vectors_test)

acc_8 = accuracy_score(y_test, pred) * 100
pre_8 = precision_score(y_test, pred) * 100
rec_8 = recall_score(y_test, pred) * 100
f1_8  = f1_score(y_test, pred) * 100

print('\nAccuracy = %f%%' % (acc_8))
print('\nprecision= %f%%' % (pre_8))
print('\nrecall   = %f%%' % (rec_8))
print('\nF1-Score = %f%%' % (f1_8))
```

```
Accuracy = 83.899311%

precision= 83.899311%

recall    = 100.000000%

F1-Score = 91.244835%
```

```
In [356]: heat_map8 = grid.cv_results_['mean_train_score'].reshape(len(estimator),len(depth))
          plt.figure(figsize=(12,8))
          sns.heatmap(heat_map8, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator
          plt.xlabel('n_estimator')
          plt.ylabel('max_depth')
          plt.xticks(np.arange(len(estimator)), estimator)
          plt.yticks(np.arange(len(depth)), depth)
          plt.title('Heatmap of Train Data')
          plt.show()
```



7 [6] Conclusions

In [366]:
```
number= [1,2,3,4,5,6,7,8]
feat  = ["Bow", "Tfidf", "Avg W2v", "Tfidf W2v", "Bow" ,"Tfidf","Avg W2v", "Tfidf W2v
model = ["Random Forest","Random Forest","Random Forest","Random Forest", "XGBoost","
depth = [optimal_depth1,optimal_depth2,optimal_depth3,optimal_depth4,optimal_depth5,o
est   = [optimal_est1,optimal_est2,optimal_est3,optimal_est4,optimal_est5,optimal_est
acc   = [acc_1,acc_2,acc_3,acc_4,acc_5,acc_6,acc_7,acc_8]
pre   = [pre_1,pre_2,pre_3,pre_4,pre_5,pre_6,pre_7,pre_8]
#rec   = [rec_1,rec_2,rec_3,rec_4,rec_5,rec_6,rec_7,rec_8]
#f1    = [f1_1,f1_2,f1_3,f1_4,f1_5,f1_6,f1_7,f1_8]


#Initialize Prettytable
pt = PrettyTable()
pt.add_column("Sr.No.", number)
pt.add_column("Featurization", feat)
pt.add_column("Model", model)
pt.add_column("Optimal Depth", depth)
pt.add_column("Optimal Estimator", est)
pt.add_column("Accuracy%", acc)
pt.add_column("Precision%", pre)
#pt.add_column("Recall%", rec)
#pt.add_column("F1%", f1)

print(pt)
```

```
+--------+---------------+---------------+---------------+-------------------+-----------------
| Sr.No. | Featurization |     Model     | Optimal Depth | Optimal Estimator |      Accuracy%
+--------+---------------+---------------+---------------+-------------------+-----------------
|   1    |      Bow      | Random Forest |       20      |         40        |  84.18274343004
|   2    |     Tfidf     | Random Forest |       20      |         80        | 84.6498556888956
|   3    |    Avg W2v    | Random Forest |       3       |         1         | 82.9902779887758
|   4    |   Tfidf W2v   | Random Forest |       3       |         1         | 75.4367309737201
|   5    |      Bow      |    XGBoost    |       10      |        200        |  89.53930182854
|   6    |     Tfidf     |    XGBoost    |       10      |        200        | 89.2662075516504
|   7    |    Avg W2v    |    XGBoost    |       3       |         40        | 83.8993113274750
|   8    |   Tfidf W2v   |    XGBoost    |       1       |         30        | 83.8993113274750
+--------+---------------+---------------+---------------+-------------------+-----------------
```