09affrrf

July 31, 2019

1 Amazon Fine Food Reviews Analysis

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

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

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

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

Attribute Information:

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

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

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

2 [1]. Reading Data

2.1 [1.1] Loading the data

The dataset is available in two forms 1. .csv file 2. SQLite Database

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

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

In [1]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


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

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

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

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

from tqdm import tqdm
import os
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 roc_auc_score
from sklearn.metrics import confusion_matrix
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 sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
```

```
C:\Users\ACER\Anaconda3\lib\site-packages\gensim\utils.py:860: UserWarning: detected Windows; a
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
C:\Users\ACER\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:29: DeprecationWa
  from numpy.core.umath_tests import inner1d
```

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

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 100

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

In [4]:
```
print(display.shape)
display.head()
```

```
(80668, 7)
```

Out[4]:
```
               UserId   ProductId             ProfileName        Time  Score  \
0  #oc-R115TNMSPFT9I7  B007Y59HVM                 Breyton  1331510400      2
1  #oc-R11D9D7SHXIJB9  B005HG9ET0  Louis E. Emory "hoppy"  1342396800      5
2  #oc-R11DNU2NBKQ23Z  B007Y59HVM        Kim Cieszykowski  1348531200      1
3  #oc-R11O5J5ZVQE25C  B005HG9ET0           Penguin Chick  1346889600      5
4  #oc-R12KPBODL2B5ZD  B007OSBE1U   Christopher P. Presta  1348617600      1

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

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

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

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

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

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

3 [2] Exploratory Data Analysis

3.1 [2.1] Data Cleaning: Deduplication

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

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

```
Out[7]:        Id   ProductId         UserId      ProfileName  HelpfulnessNumerator  \
        0   78445  B000HDL1RQ  AR5J8UI46CURR  Geetha Krishnan                     2
        1  138317  B000HDOPYC  AR5J8UI46CURR  Geetha Krishnan                     2
        2  138277  B000HDOPYM  AR5J8UI46CURR  Geetha Krishnan                     2
        3   73791  B000HDOPZG  AR5J8UI46CURR  Geetha Krishnan                     2
        4  155049  B000PAQ75C  AR5J8UI46CURR  Geetha Krishnan                     2

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

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

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

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8) ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

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

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

In [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=Fals
```

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

Out[9]:
```
(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 [14]: # printing some random reviews
         sent_0 = final['Text'].values[0]
         print(sent_0)
         print("="*50)

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

```
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid
==================================================
```

```
In [15]: # remove urls from text python: https://stackoverflow.com/a/40823105/4084039
         sent_0 = re.sub(r"http\S+", "", sent_0)
         sent_1000 = re.sub(r"http\S+", "", sent_1000)
         sent_150 = re.sub(r"http\S+", "", sent_1500)
         sent_4900 = re.sub(r"http\S+", "", sent_4900)

         print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

```
In [16]: # https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all
         from bs4 import BeautifulSoup

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

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

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

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

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
==================================================
The Candy Blocks were a nice visual for the Lego Birthday party but the candy has little taste
==================================================
was way to hot for my blood, took a bite and did a jig  lol
==================================================
My dog LOVES these treats. They tend to have a very strong fish oil smell. So if you are afraid
```

```
In [17]: # https://stackoverflow.com/a/47091490/4084039
         import re
```

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

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

In [18]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
was way to hot for my blood, took a bite and did a jig  lol
==================================================
```

In [19]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
My dogs loves this chicken but its a product from China, so we wont be buying it anymore.  Its
```

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

```
was way to hot for my blood took a bite and did a jig lol
```

In [21]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselve
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'l
```

```
                    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
                    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through
                    '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 [22]: # Combining all the above stundents
         from tqdm import tqdm
         preprocessed_reviews_rf = []
         # tqdm is for printing the status bar
         for sentance in tqdm(final['Text'].values):
             sentance = re.sub(r"http\S+", "", sentance)
             sentance = BeautifulSoup(sentance, 'lxml').get_text()
             sentance = decontracted(sentance)
             sentance = re.sub("\S*\d\S*", "", sentance).strip()
             sentance = re.sub('[^A-Za-z]+', ' ', sentance)
             # https://gist.github.com/sebleier/554280
             sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopw
             preprocessed_reviews_rf.append(sentance.strip())
```

```
100%|| 87773/87773 [01:00<00:00, 1444.42it/s]
```

```
In [23]: preprocessed_reviews_rf[1500]
```

```
Out[23]: 'way hot blood took bite jig lol'
```

4.2 [4] Splitting the data

```
In [24]: X = preprocessed_reviews_rf
         Y = final['Score'].values
```

```
In [25]: # Here we are splitting the data(X ,Y) into train and test data
         # X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=F
         X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30) # this is r
```

5 [4] Featurization

5.1 [4.1] BAG OF WORDS

```
In [26]: #BoW
         vectorizer = CountVectorizer(min_df = 10)
         vectorizer.fit(X_train) # fit has to happen only on train data
         print(vectorizer.get_feature_names()[:20])# printing some feature names
```

```
print("="*50)

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_test_bow = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_bow.shape, Y_train.shape)
print(X_test_bow.shape, Y_test.shape)
```

```
['aa', 'aafco', 'aback', 'abandoned', 'abdominal', 'ability', 'able', 'abroad', 'absence', 'abs
==================================================
After vectorizations
(61441, 9727) (61441,)
(26332, 9727) (26332,)
```

5.2 [4.3] TF-IDF

In [27]:
```
tfidf_vect = TfidfVectorizer(min_df=10)
tfidf_vect.fit(X_train)
print("some sample features ",tfidf_vect.get_feature_names()[0:10])
print('='*50)

# we use the fitted CountVectorizer to convert the text to vector
X_train_tfidf = tfidf_vect.transform(X_train)
X_test_tfidf = tfidf_vect.transform(X_test)

print("After vectorizations")
print(X_train_tfidf.shape, Y_train.shape)
print(X_test_tfidf.shape, Y_test.shape)
```

```
some sample features  ['aa', 'aafco', 'aback', 'abandoned', 'abdominal', 'ability', 'able', 'al
==================================================
After vectorizations
(61441, 9727) (61441,)
(26332, 9727) (26332,)
```

5.3 [4.4] Word2Vec

In [28]:
```
# 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 [29]:
```
# this line of code trains your w2v model on the give list of sentances, fitting the
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=-1)
```

```
In [30]: 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  14804
sample words  ['first', 'hawaiian', 'salt', 'ive', 'ever', 'tried', 'cooking', 'gave', 'stars'
```

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

[4.4.1.1] Avg W2v

5.4.1 Converting Train data set

```
In [31]: # average Word2Vec
         # compute average word2vec for each review.
         sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this lis
         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 t
             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%|| 61441/61441 [02:51<00:00, 358.64it/s]


(61441, 50)
[ 1.49530279e-03  1.74013253e-03 -3.35474421e-04 -1.79215600e-03
 -1.22944877e-03  6.37771201e-05  1.57838220e-03  1.83518851e-03
 -2.46359594e-04  1.59830931e-03 -2.31750775e-04 -4.86755654e-05
  1.20638629e-03  2.72470262e-04  1.84368356e-03  1.17688291e-04
 -1.32211400e-04  4.92880523e-05  1.96743069e-04  9.99466279e-04
 -2.17257638e-04  1.82484241e-03  1.28306261e-03  1.18840378e-03
  6.58293802e-04  1.55421265e-03 -2.73246577e-04 -1.88914748e-04
  2.19925632e-03 -1.72848888e-04  3.90295542e-04 -5.58386260e-05
 -6.23318934e-04 -1.24313771e-03 -5.19351171e-04 -2.93402546e-04
 -1.77312948e-03  1.65710942e-03 -5.65482641e-04  2.46362106e-03
 -2.24579809e-04  7.06118126e-04 -1.34266445e-03  5.98191188e-04
 -7.05843118e-04 -9.57805030e-04 -1.14978465e-03 -6.45412965e-05
```

```
 -5.61554718e-04 -1.21379622e-04]
```

```
In [32]: type(sent_vectors_train)
```

```
Out[32]: numpy.ndarray
```

5.4.2 Converting Test data set

```
In [33]: list_of_sentance_test=[]
         for sentance in X_test:
             list_of_sentance_test.append(sentance.split())
```

```
In [34]: # average Word2Vec
         # compute average word2vec for each review.
         sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
         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 t
             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%|| 26332/26332 [01:13<00:00, 356.15it/s]


(26332, 50)
[ 1.11712633e-03  9.86252709e-04 -3.88459998e-04 -4.34413278e-04
  8.44555456e-04  1.32795192e-03 -8.84551686e-04 -1.13731602e-03
 -7.31953090e-04 -1.30945456e-03 -2.53102247e-04 -2.67113570e-04
 -2.64294749e-04  1.68484774e-03  1.14786019e-03 -6.89245567e-04
 -8.69215923e-05 -1.17180864e-03 -2.58920362e-03 -1.13680045e-03
 -4.75744506e-04 -6.86671648e-04  1.06466102e-03 -2.07969540e-04
  1.29443252e-03  2.73572570e-04 -1.47164530e-03  7.85484186e-04
 -1.95110830e-04  1.06518659e-03 -1.65947218e-03  5.17210330e-04
  2.34779015e-03  1.02078152e-03  1.28233555e-03 -4.07889093e-05
  8.94802575e-04  7.73024989e-05 -1.50300679e-04 -8.53352848e-04
  1.47817630e-03 -8.51270536e-04 -1.74862290e-04 -3.90027578e-04
  7.50398346e-04 -6.70166485e-04 -1.86333669e-03  5.60554299e-04
  1.90450745e-03 -5.34504875e-04]
```

[4.4.1.2] TFIDF weighted W2v

5.4.3 Converting Train data set

In [35]:
```
# 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_)))
```

In [36]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_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%|| 61441/61441 [39:04<00:00, 26.21it/s]
```

In [37]:
```
tfidf_sent_vectors_train1 = np.asarray(tfidf_sent_vectors_train)
type(tfidf_sent_vectors_train1)
```

Out[37]: numpy.ndarray

In [39]:
```
tfidf_sent_vectors_test1 = np.asarray(tfidf_sent_vectors_test)
type(tfidf_sent_vectors_test1)
```

Out[39]: numpy.ndarray

5.4.4 Converting Test data set

In [38]:
```python
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored in t
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%|| 26332/26332 [07:50<00:00, 55.99it/s]
```

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 vector
        <li><font color='red'>SET 2:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 3:</font>Review text, preprocessed one converted into vector
        <li><font color='red'>SET 4:</font>Review text, preprocessed one converted into vector
    </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.appliedaico
<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.
<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.

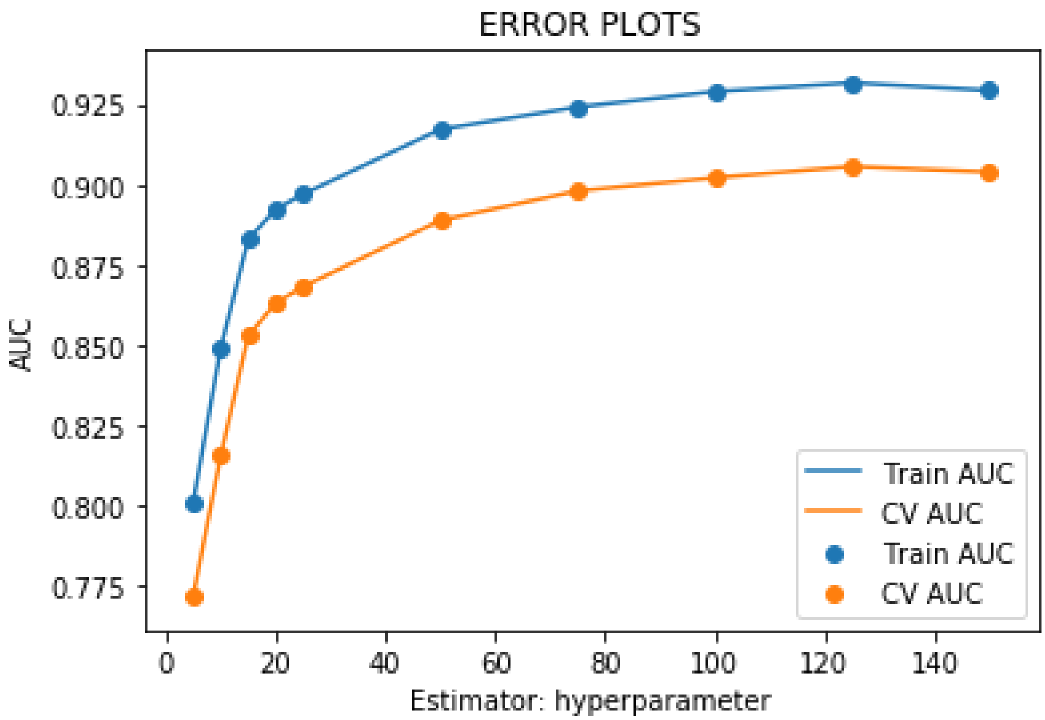
6.1 [5.1] Applying RF

6.1.1 [5.1.1] Applying Random Forests on BOW, SET 1

6.1.2 Hyperparameter tuning using GridSearch

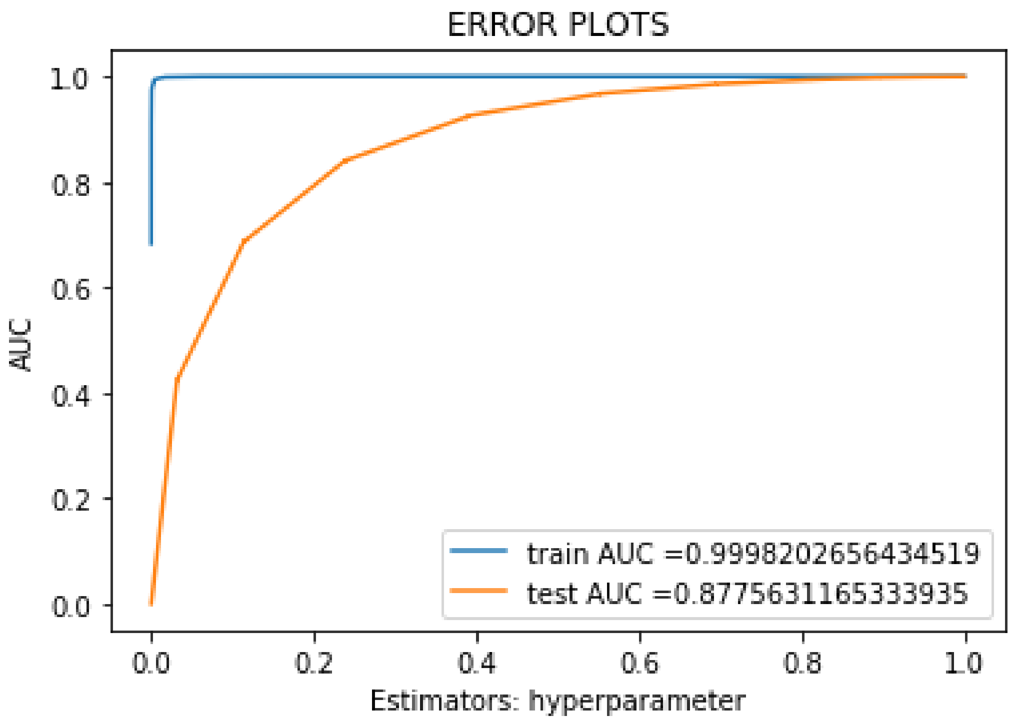
In [40]:
```
#clf = RandomForestClassifier()
# For Estimator
estimator = [5,10,15,20,25,50,75,100,125,150]
parameters = {'n_estimators':[5,10,15,20,25,50,75,100,125,150]}
grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',max_depth = 10), p
grid.fit(X_train_bow, Y_train)

print("best estimator = ", grid.best_params_)

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()
```

```
best estimator =  {'n_estimators': 125}
```



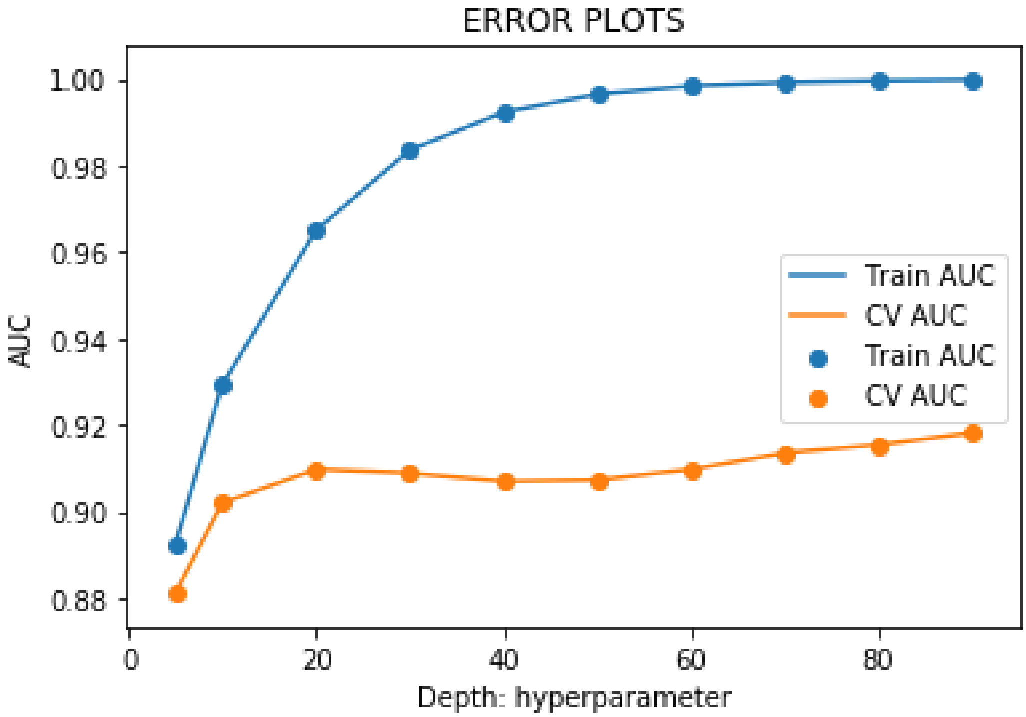
6.1.3 Testing with Test data

```
In [40]: clf = RandomForestClassifier(n_estimators = 10, class_weight ='balanced')
         clf.fit(X_train_bow, Y_train)

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

         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()
```



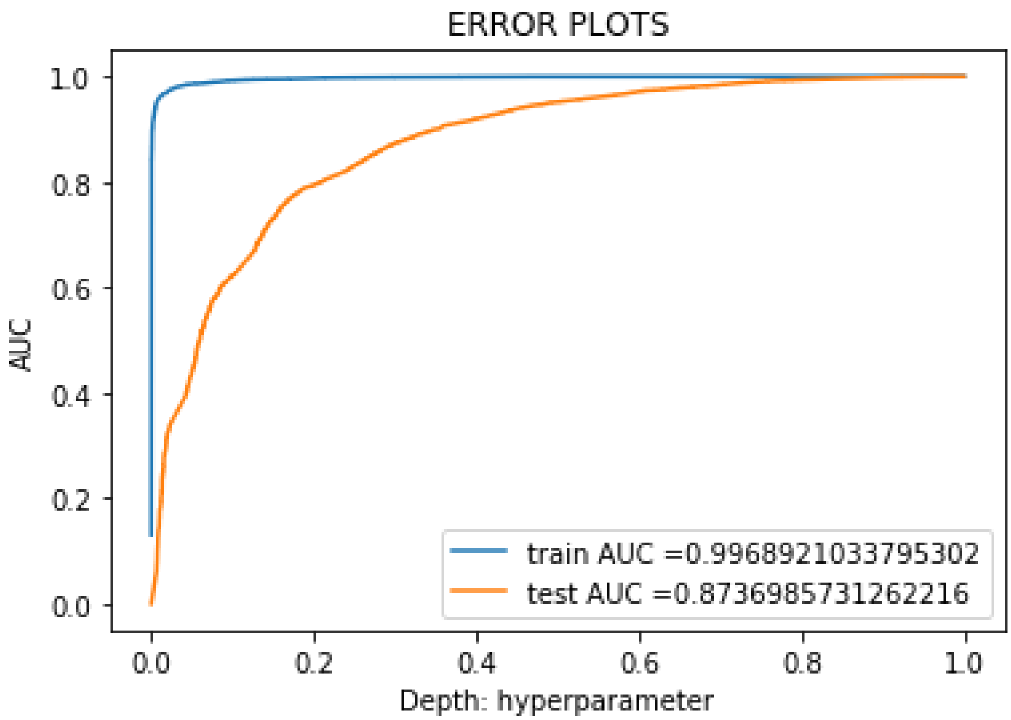
In [41]:
```
#clf = RandomForestClassifier()
# For Depth
depth = [5,10,20,30,40,50,60,70,80,90]
parameters = {'max_depth':[5,10,20,30,40,50,60,70,80,90]}
grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',n_estimators = 10(
grid.fit(X_train_bow, Y_train)

print("best depth = ", grid.best_params_)

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()
```

```
best depth =  {'max_depth': 90}
```



6.1.4 Testing with Test Data

```
In [42]: clf = RandomForestClassifier(max_depth = 90, class_weight ='balanced')
         clf.fit(X_train_bow, Y_train)

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

         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()
```



1. Here best value for n_estimator = 100 because after 100, AUC value is not changing much.
2. Here best value for max_depth = 90 because after 90, AUC value is not changing much.

In [41]:
```
# clf = RandomForestClassifier()
estimator = [5,10,15,20,25,50,75,100,125,150]
depth = [5,10,20,30,40,50,60,70,80,90]

parameters = {'n_estimators': estimator,'max_depth': depth}
grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',max_features='sqr
grid.fit(X_train_bow, Y_train)
```

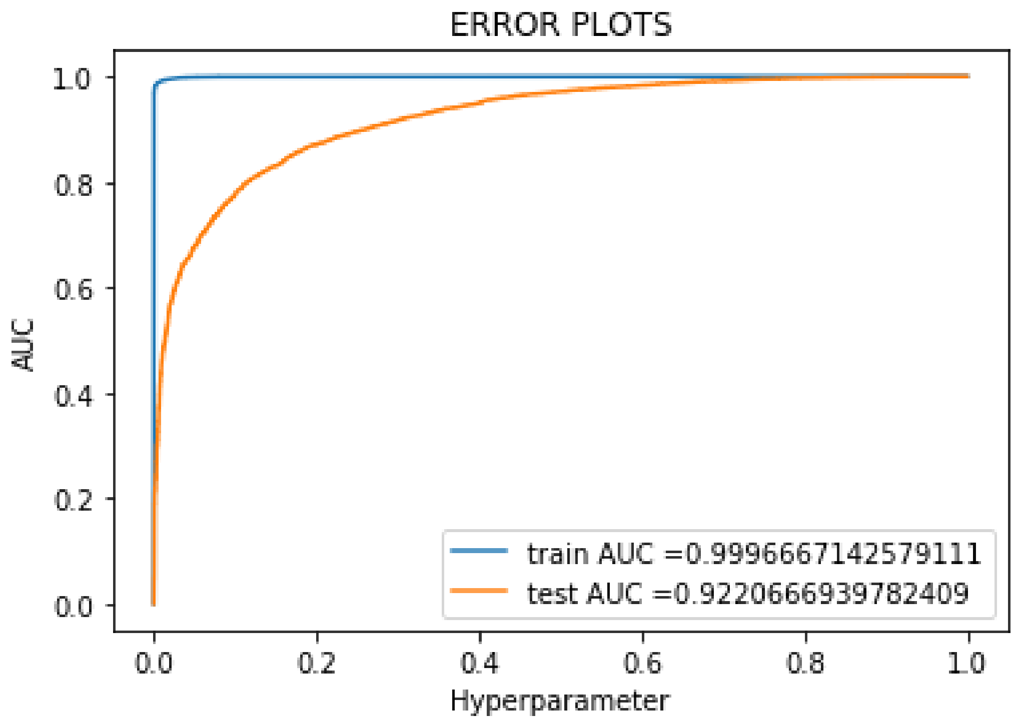
Out[41]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='sqrt',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'n_estimators': [5, 10, 15, 20, 25, 50, 75, 100, 125, 150], 'max_de
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

```
In [42]: optimal_estimator = grid.best_estimator_.n_estimators
         print("Value for optimal estimator : ",optimal_estimator)

         optimal_depth = grid.best_estimator_.max_depth
         print("Value for optimal depth : ",optimal_depth)
```

```
Value for optimal estimator :  150
Value for optimal depth :  90
```

6.1.5 Testing with Test data

```
In [43]: clf = RandomForestClassifier(max_depth = optimal_depth, n_estimators= optimal_estimat
         clf.fit(X_train_bow, Y_train)

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

         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 [72]:
```
clf = RandomForestClassifier(n_estimators = optimal_estimator, max_depth = optimal_dep
clf.fit(X_train_bow, Y_train)
predb1 = clf.predict(X_test_bow)

accb1 = accuracy_score(Y_test, predb1) * 100
preb1 = precision_score(Y_test, predb1) * 100
recb1 = recall_score(Y_test, predb1) * 100
f1b1 = f1_score(Y_test, predb1) * 100

print('\nAccuracy=%f%%' % (accb1))
print('\nprecision=%f%%' % (preb1))
print('\nrecall=%f%%' % (recb1))
print('\nF1-Score=%f%%' % (f1b1))
```

```
Accuracy=87.118335%

precision=86.861574%

recall=99.677668%

F1-Score=92.829359%
```

6.1.6 [5.1.2] Wordcloud of top 20 important features from SET 1

```
In [55]: # Calculate feature importances from decision trees
         importances = clf.feature_importances_

         # Sort feature importances in descending order
         indices = list(np.argsort(importances)[::-1][:20])
         print(indices)
```

```
[5693, 3770, 4992, 715, 3693, 4114, 9596, 2438, 6157, 6573, 4997, 548, 4048, 2241, 527, 8660, 
```

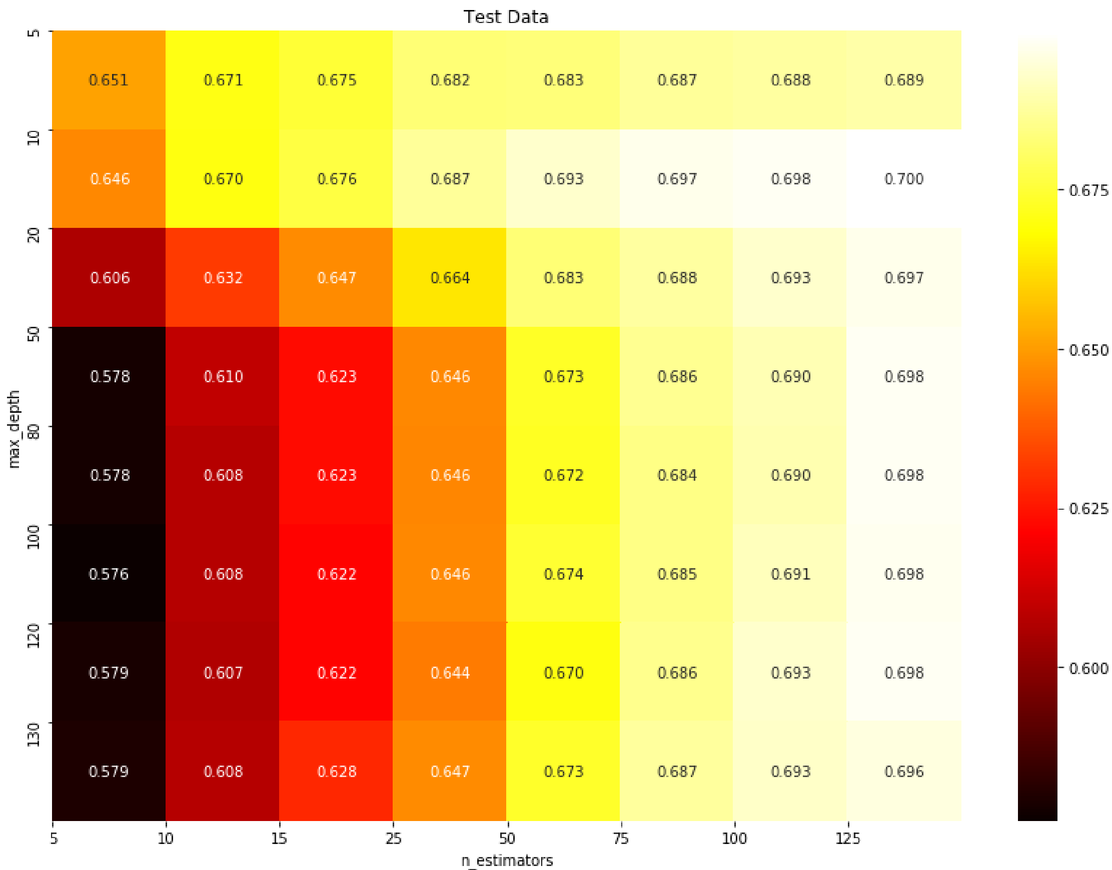
```
In [56]: names = np.array(vectorizer.get_feature_names())
         print(names[indices])
```

```
['not' 'great' 'love' 'best' 'good' 'horrible' 'worst' 'disappointed'
 'perfect' 'product' 'loves' 'bad' 'highly' 'delicious' 'awful' 'terrible'
 'excellent' 'thought' 'money' 'favorite']
```

```
In [72]: text = str(names[indices])

         # Create and generate a word cloud image:
         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()
```

6.1.7 [5.3.1] Heatmap on Test Data

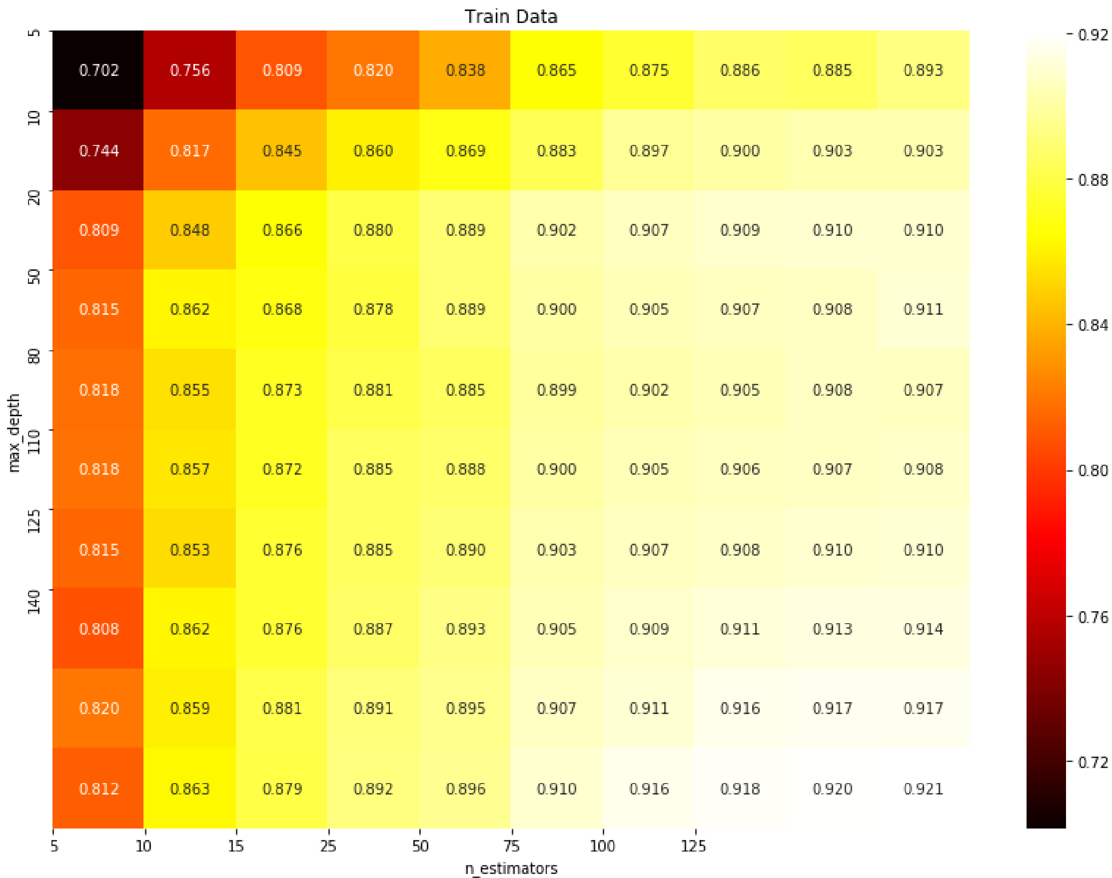
In [95]:
```
scores = grid.cv_results_['mean_test_score'].reshape(len(estimator),len(depth))
plt.figure(figsize=(14,10))
sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator, yt
plt.xlabel('n_estimators')
plt.ylabel('max_depth')
plt.xticks(np.arange(len(estimator)), estimator)
plt.yticks(np.arange(len(depth)), depth)
plt.title('Test Data')
plt.show()
```



6.1.8 [5.3.2] Heatmap on Train Data

In [82]:
```
scores1 = grid.cv_results_['mean_train_score'].reshape(len(estimator),len(depth))
plt.figure(figsize=(14,10))
sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator, yt
plt.xlabel('n_estimators')
plt.ylabel('max_depth')
plt.xticks(np.arange(len(estimator)), estimator)
```

```
plt.yticks(np.arange(len(depth)), depth)
plt.title('Train Data')
plt.show()
```



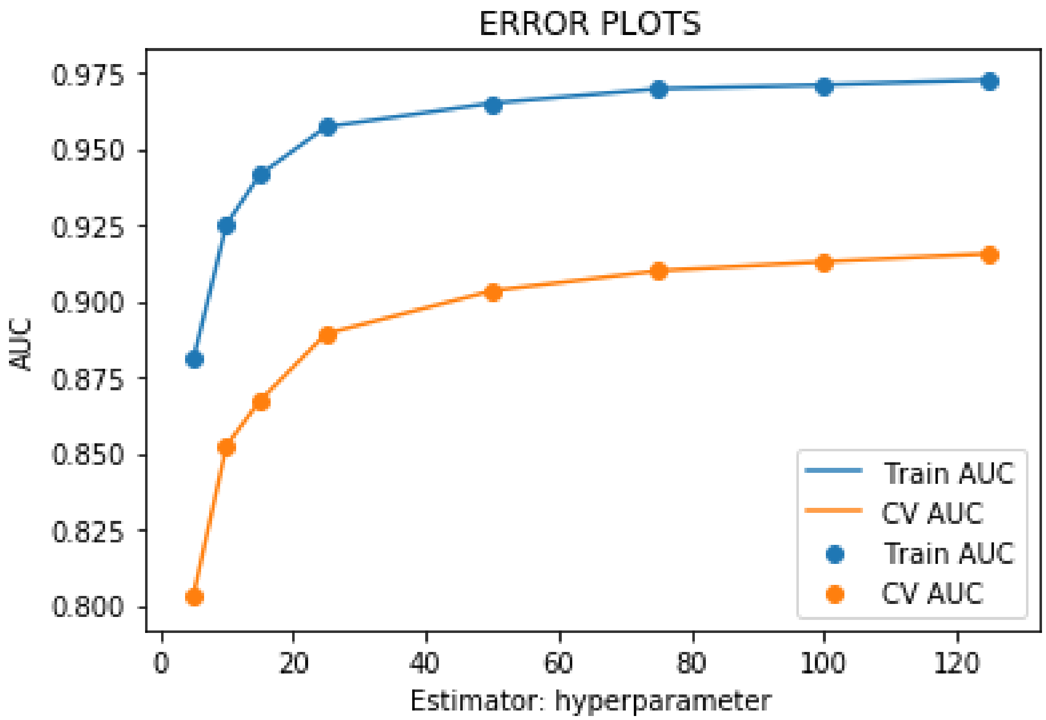
6.1.9 [5.1.3] Applying Random Forests on TFIDF, SET 2

6.1.10 Hyperparameter tuning using GridSearch

In [43]:
```
# clf = RandomForestClassifier()
# for Estimators in Random Forest
estimator = [5,10,15,25,50,75,100,125]
parameters = {'n_estimators':[5,10,15,25,50,75,100,125]}
grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced', max_depth=20), pa
grid.fit(X_train_tfidf, Y_train)

print("best estimator = ", grid.best_params_)

train_auc_tfidf = grid.cv_results_['mean_train_score']
cv_auc_tfidf = grid.cv_results_['mean_test_score']
```

```
plt.plot(estimator, train_auc_tfidf, label='Train AUC')
plt.scatter(estimator, train_auc_tfidf, label='Train AUC')
plt.plot(estimator, cv_auc_tfidf, label='CV AUC')
plt.scatter(estimator, cv_auc_tfidf, label='CV AUC')

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

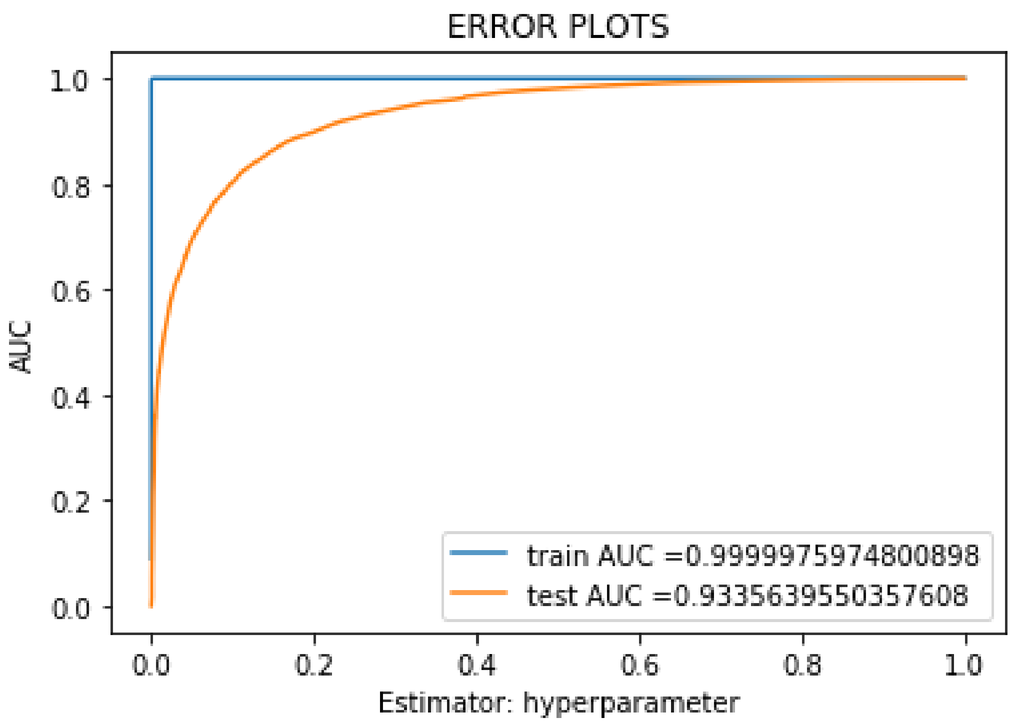
```
best estimator =  {'n_estimators': 125}
```



6.1.11 Testing with Test Data

```
In [44]: clf = RandomForestClassifier(n_estimators = 125, class_weight ='balanced')
         clf.fit(X_train_tfidf, Y_train)

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

         train_fpr_tfidf, train_tpr_tfidf, thresholds_tfidf = roc_curve(Y_train, clf.predict_p
         test_fpr_tfidf, test_tpr_tfidf, thresholds_tfidf = roc_curve(Y_test, clf.predict_prob
```

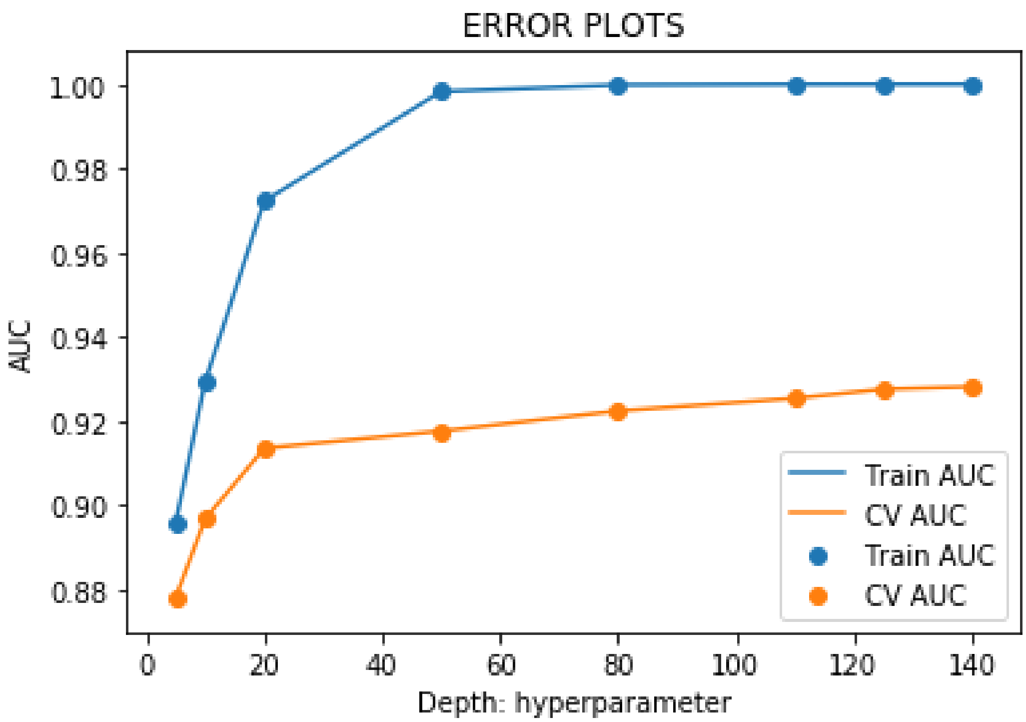
```
plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="train AUC ="+str(auc(train_fpr_tfid
plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="test AUC ="+str(auc(test_fpr_tfidf, te
plt.legend()
plt.xlabel("Estimator: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [45]:
```
# clf = RandomForestClassifier()
# for Maximum Depth in Random Forest
depth = [5,10,20,50,80,110,125,140]
parameters = {'max_depth':[5,10,20,50,80,110,125,140]}
grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced', n_estimators=100)
grid.fit(X_train_tfidf, Y_train)

print("best depth = ", grid.best_params_)

train_auc_tfidf = grid.cv_results_['mean_train_score']
cv_auc_tfidf = grid.cv_results_['mean_test_score']

plt.plot(depth, train_auc_tfidf, label='Train AUC')
plt.scatter(depth, train_auc_tfidf, label='Train AUC')
plt.plot(depth, cv_auc_tfidf, label='CV AUC')
```

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

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

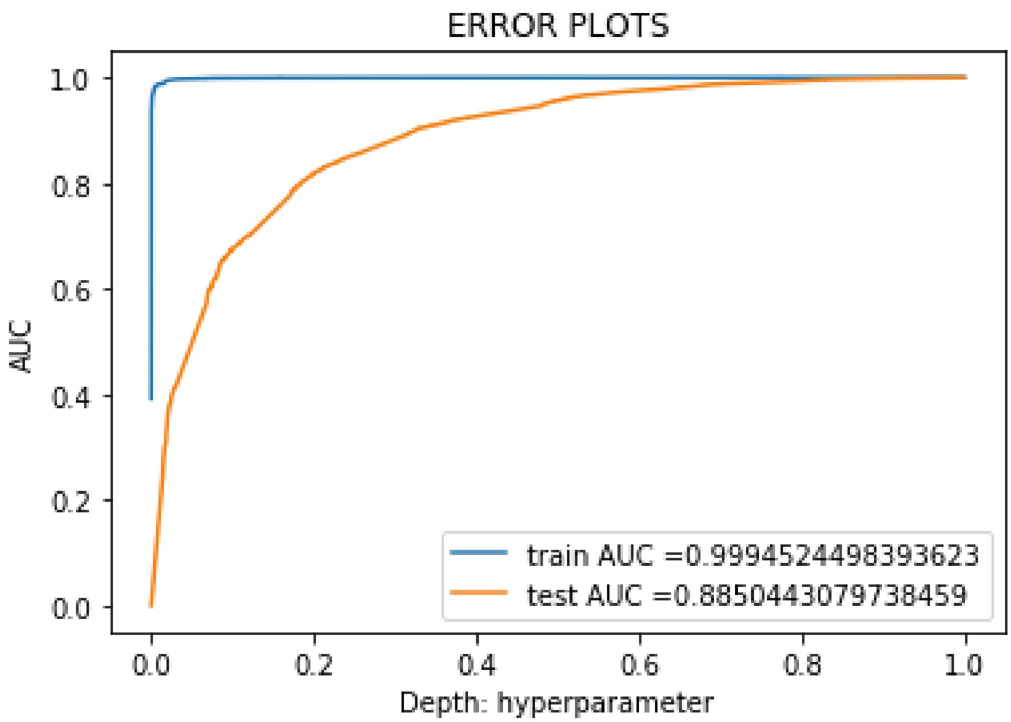
```
best depth =  {'max_depth': 140}
```



6.1.12 Testing with Test data

```
In [46]: clf = RandomForestClassifier(max_depth = 140, class_weight ='balanced')
         clf.fit(X_train_tfidf, Y_train)

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

         train_fpr_tfidf, train_tpr_tfidf, thresholds_tfidf = roc_curve(Y_train, clf.predict_p
         test_fpr_tfidf, test_tpr_tfidf, thresholds_tfidf = roc_curve(Y_test, clf.predict_prob

         plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="train AUC ="+str(auc(train_fpr_tfid
         plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="test AUC ="+str(auc(test_fpr_tfidf, te
```

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




```
In [44]: # clf = RandomForestClassifier()
         estimator = [5,10,15,25,50,75,100,125]
         depth = [5,10,20,50,80,110,125,140]

         parameters = {'n_estimators': estimator,'max_depth': depth}
         grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',max_features='sqr
         grid.fit(X_train_tfidf, Y_train)

Out[44]: GridSearchCV(cv=3, error_score='raise',
                estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
                     criterion='gini', max_depth=None, max_features='sqrt',
                     max_leaf_nodes=None, min_impurity_decrease=0.0,
                     min_impurity_split=None, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
                     verbose=0, warm_start=False),
                fit_params=None, iid=True, n_jobs=-1,
                param_grid={'n_estimators': [5, 10, 15, 25, 50, 75, 100, 125], 'max_depth': [5
```

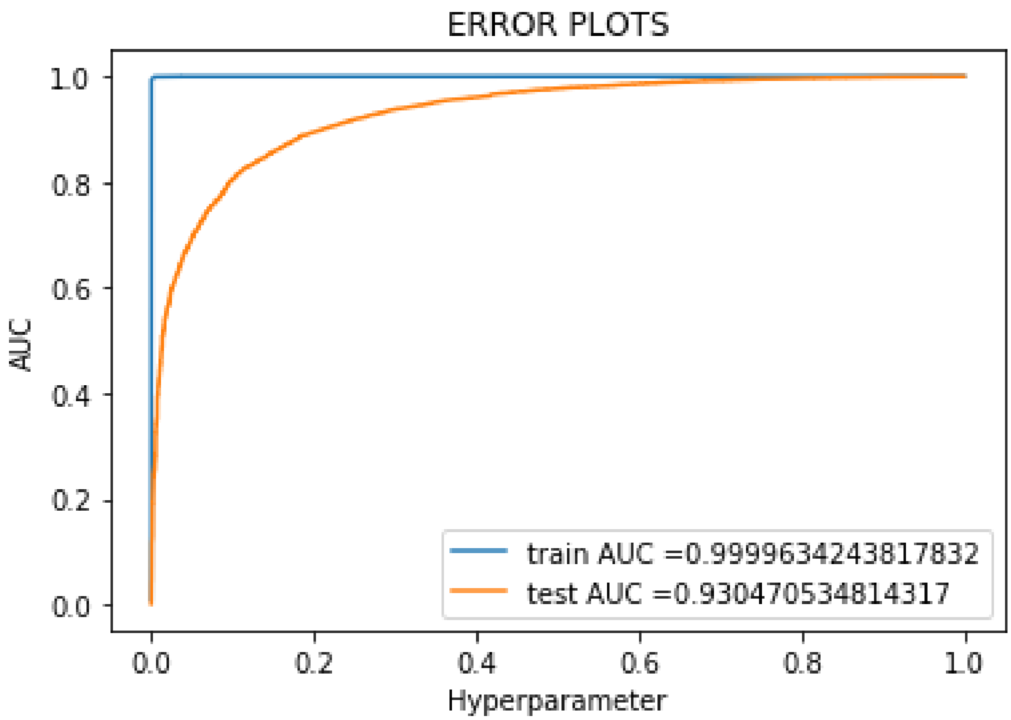
```
            pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
            scoring='roc_auc', verbose=0)
```

```
In [45]: optimal_estimator = grid.best_estimator_.n_estimators
         print("Value for optimal estimator : ",optimal_estimator)

         optimal_depth = grid.best_estimator_.max_depth
         print("Value for optimal depth : ",optimal_depth)
```

```
Value for optimal estimator :  125
Value for optimal depth :  140
```

6.1.13 Testing with Test data

```
In [46]: clf = RandomForestClassifier(max_depth = optimal_depth, n_estimators= optimal_estimat
         clf.fit(X_train_tfidf, Y_train)

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

         train_fpr_tfidf, train_tpr_tfidf, thresholds_tfidf = roc_curve(Y_train, clf.predict_pr
         test_fpr_tfidf, test_tpr_tfidf, thresholds_tfidf = roc_curve(Y_test, clf.predict_proba

         plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="train AUC ="+str(auc(train_fpr_tfidf
         plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="test AUC ="+str(auc(test_fpr_tfidf, te
         plt.legend()
         plt.xlabel("Hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```



In [81]:
```
clf = RandomForestClassifier(n_estimators = optimal_estimator, max_depth = optimal_dep
clf.fit(X_train_tfidf, Y_train)
predt = clf.predict(X_test_tfidf)

acct = accuracy_score(Y_test, predt) * 100
pret = precision_score(Y_test, predt) * 100
rect = recall_score(Y_test, predt) * 100
f1t = f1_score(Y_test, predt) * 100

print('\nAccuracy=%f%%' % (acct))
print('\nprecision=%f%%' % (pret))
print('\nrecall=%f%%' % (rect))
print('\nF1-Score=%f%%' % (f1t))
```

```
Accuracy=87.650008%

precision=87.381038%

recall=99.623190%

F1-Score=93.101400%
```

6.1.14 [5.1.4] Wordcloud of top 20 important features from SET 2

In [171]:
```
# Calculate feature importances from decision trees
importances = clf.feature_importances_

# Sort feature importances in descending order
indices = list(np.argsort(importances)[::-1][:20])
print(indices)
```

```
[3770, 5693, 715, 2438, 3141, 9596, 7851, 5457, 2241, 523, 9389, 8660, 4997, 5200, 3693, 4114,
```

In [172]:
```
names = np.array(vectorizer.get_feature_names())
print(names[indices])
```

```
['great' 'not' 'best' 'disappointed' 'favorite' 'worst' 'snack' 'money'
 'delicious' 'away' 'waste' 'terrible' 'loves' 'maybe' 'good' 'horrible'
 'disappointing' 'tasted' 'reviews' 'thought']
```

In [173]:
```
text = str(names[indices])

# Create and generate a word cloud image:
wordcloud = WordCloud(max_font_size=50, max_words=30, background_color="white").gene

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

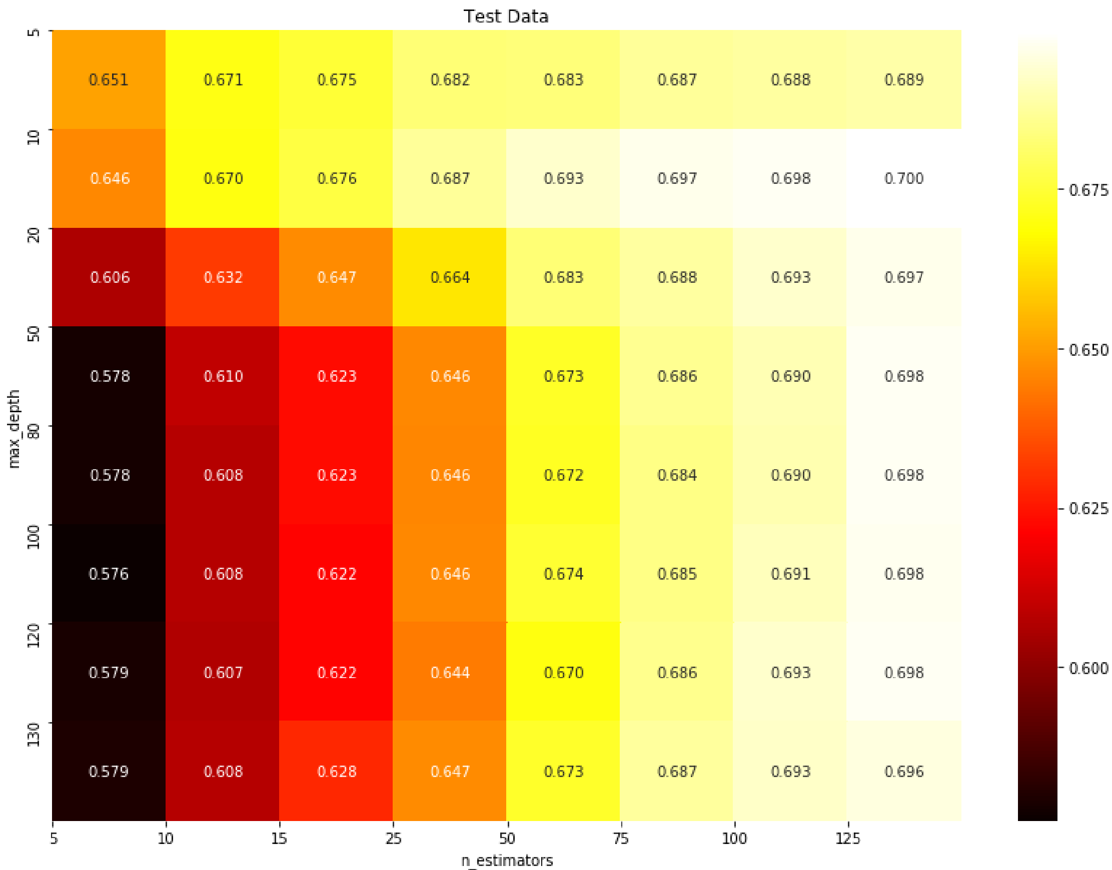
6.1.15 [5.4.1] Heatmap on Train Data

```
In [84]: scores = grid.cv_results_['mean_train_score'].reshape(len(estimator),len(depth))
         plt.figure(figsize=(14,10))
         sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator, yt
         plt.xlabel('n_estimators')
         plt.ylabel('max_depth')
         plt.xticks(np.arange(len(estimator)), estimator)
         plt.yticks(np.arange(len(depth)), depth)
         plt.title('Train Data')
         plt.show()
```

6.1.16 [5.4.2] Heatmap on Test Data

```
In [94]: scores1 = grid.cv_results_['mean_test_score'].reshape(len(estimator),len(depth))
         plt.figure(figsize=(14,10))
         sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator, yt
         plt.xlabel('n_estimators')
         plt.ylabel('max_depth')
         plt.xticks(np.arange(len(estimator)), estimator)
```

```
plt.yticks(np.arange(len(depth)), depth)
plt.title('Test Data')
plt.show()
```



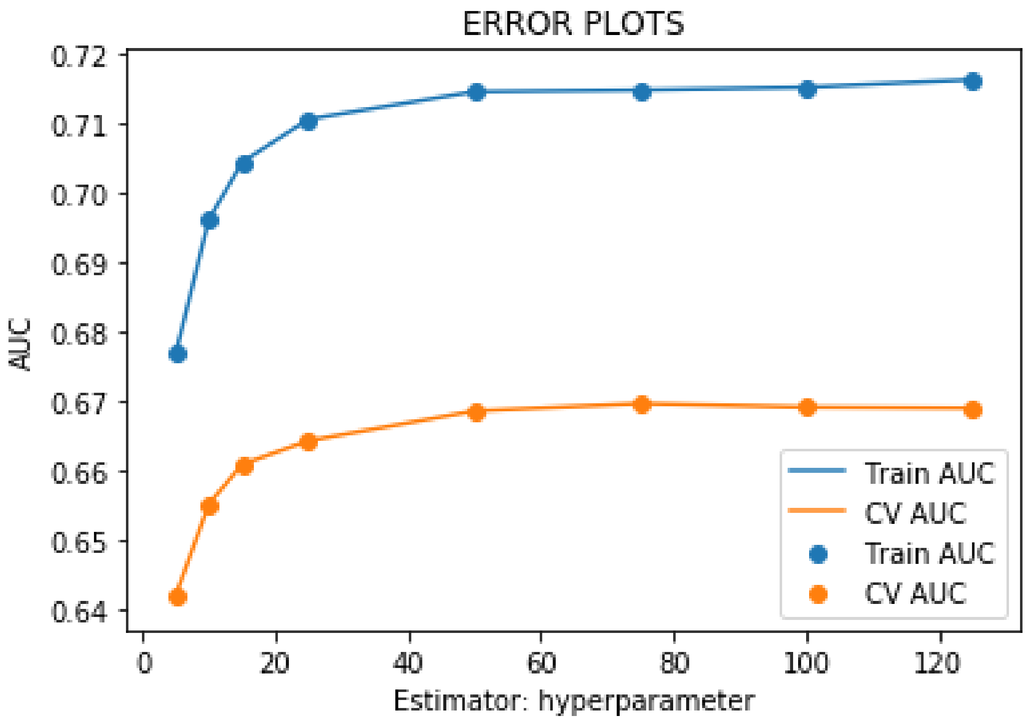
6.1.17 [5.1.5] Applying Random Forests on AVG W2V, SET 3

6.1.18 Hyperparameter tuning using GridSearch

```
In [64]: # clf = RandomForestClassifier()
         # for Estimators in Random Forest
         estimator = [5,10,15,25,50,75,100,125]
         parameters = {'n_estimators':[5,10,15,25,50,75,100,125]}
         grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced', max_depth=5), pa
         grid.fit(sent_vectors_train, Y_train)

         print("best estimator = ", grid.best_params_)

         train_auc_aw2v = grid.cv_results_['mean_train_score']
         cv_auc_aw2v = grid.cv_results_['mean_test_score']
```

```
plt.plot(estimator, train_auc_aw2v, label='Train AUC')
plt.scatter(estimator, train_auc_aw2v, label='Train AUC')
plt.plot(estimator, cv_auc_aw2v, label='CV AUC')
plt.scatter(estimator, cv_auc_aw2v, label='CV AUC')

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

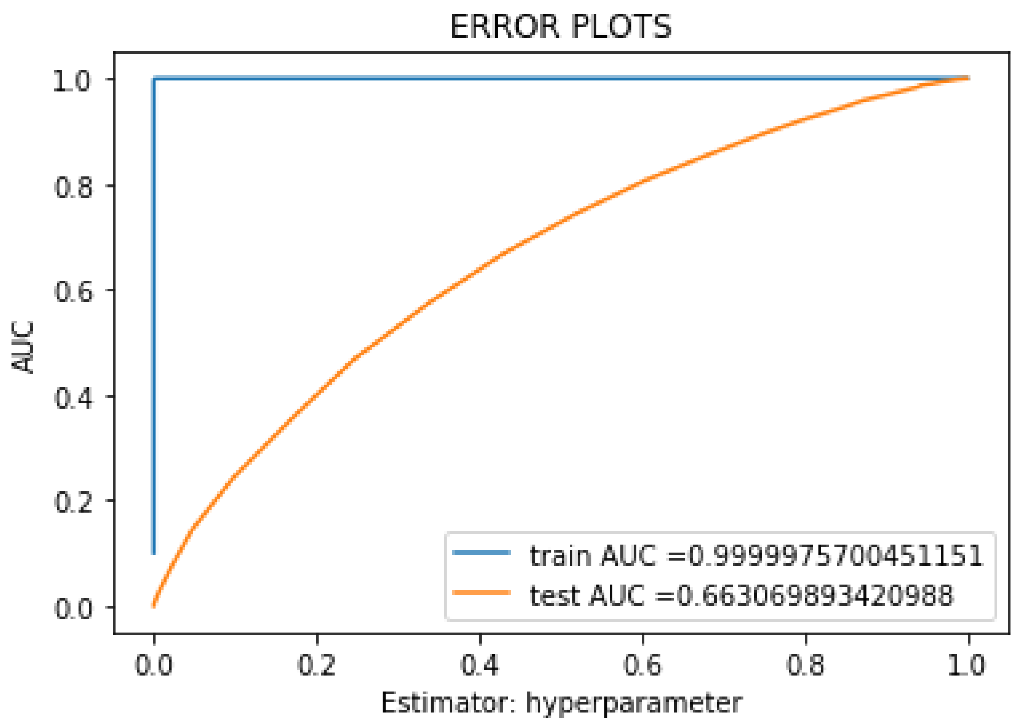
```
best estimator =  {'n_estimators': 75}
```



6.1.19 Testing with Test data

```
In [94]: clf = RandomForestClassifier(n_estimators= 50,class_weight='balanced')
         clf.fit(sent_vectors_train, Y_train)

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

         train_fpr_aw2v, train_tpr_aw2v, thresholds_aw2v = roc_curve(Y_train, clf.predict_proba
         test_fpr_aw2v, test_tpr_aw2v, thresholds_aw2v = roc_curve(Y_test, clf.predict_proba(se
```

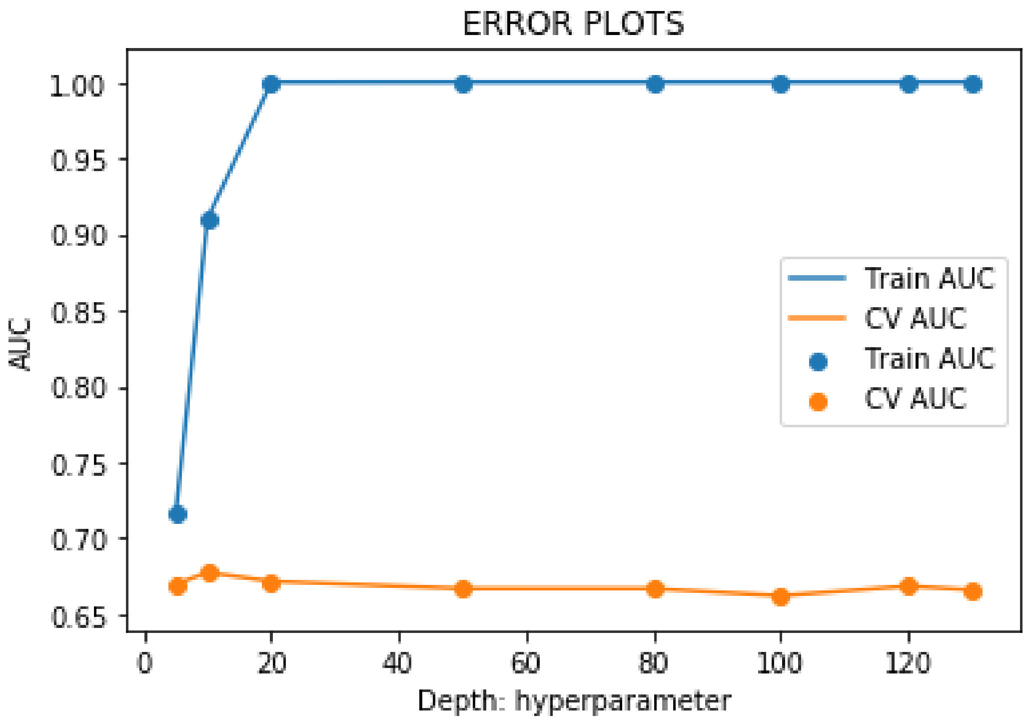
```
plt.plot(train_fpr_aw2v, train_tpr_aw2v, label="train AUC ="+str(auc(train_fpr_aw2v, t
plt.plot(test_fpr_aw2v, test_tpr_aw2v, label="test AUC ="+str(auc(test_fpr_aw2v, test_
plt.legend()
plt.xlabel("Estimator: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```




```
In [69]: # clf = RandomForestClassifier()
         # for Maximum Depth in Random Forest
         depth = [5,10,20,50,80,100,120,130]
         parameters = {'max_depth':[5,10,20,50,80,100,120,130]}
         grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced', n_estimators=80)
         grid.fit(sent_vectors_train, Y_train)

         print("best depth = ", grid.best_params_)

         train_auc_aw2v = grid.cv_results_['mean_train_score']
         cv_auc_aw2v = grid.cv_results_['mean_test_score']

         plt.plot(depth, train_auc_aw2v, label='Train AUC')
         plt.scatter(depth, train_auc_aw2v, label='Train AUC')
         plt.plot(depth, cv_auc_aw2v, label='CV AUC')
```

```
plt.scatter(depth, cv_auc_aw2v, label='CV AUC')

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

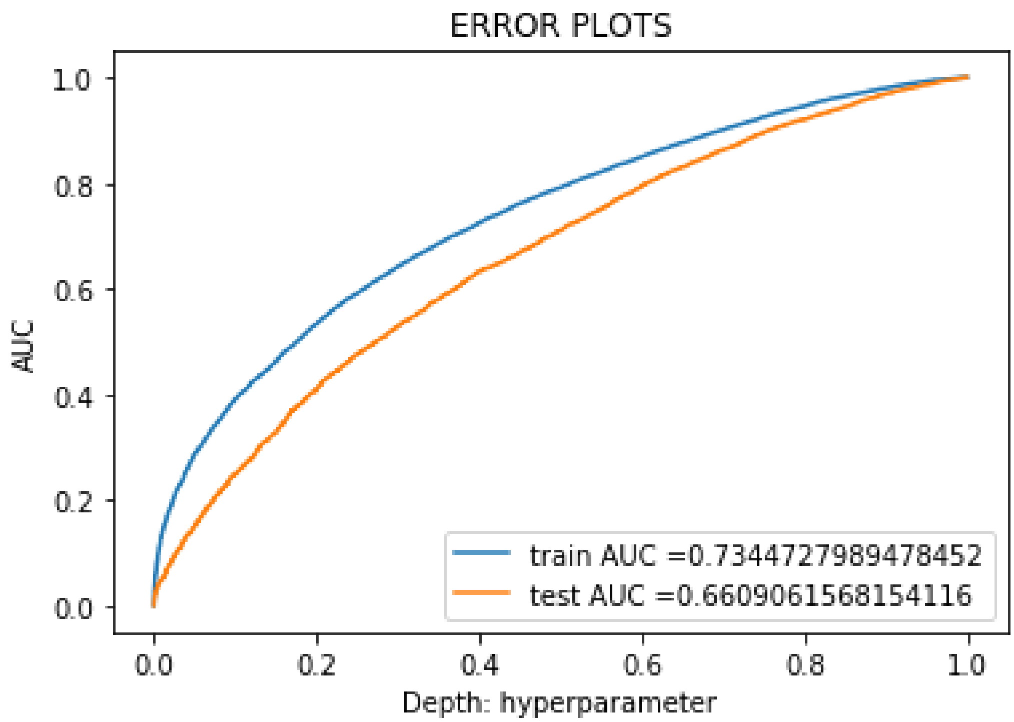
```
best depth =  {'max_depth': 10}
```



6.1.20 Testing with Test Data

```
In [74]: clf = RandomForestClassifier(max_depth= 7, class_weight='balanced')
         clf.fit(sent_vectors_train, Y_train)

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

         train_fpr_aw2v, train_tpr_aw2v, thresholds_aw2v = roc_curve(Y_train, clf.predict_proba
         test_fpr_aw2v, test_tpr_aw2v, thresholds_aw2v = roc_curve(Y_test, clf.predict_proba(se

         plt.plot(train_fpr_aw2v, train_tpr_aw2v, label="train AUC ="+str(auc(train_fpr_aw2v, t
         plt.plot(test_fpr_aw2v, test_tpr_aw2v, label="test AUC ="+str(auc(test_fpr_aw2v, test_
```

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




```
In [47]: # clf = RandomForestClassifier()
         estimator = [5,10,15,25,50,75,100,125]
         depth = [5,10,20,50,80,100,120,130]

         parameters = {'n_estimators': estimator,'max_depth': depth}
         grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',max_features='sqrt
         grid.fit(sent_vectors_train, Y_train)

Out[47]: GridSearchCV(cv=3, error_score='raise',
                estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
                     criterion='gini', max_depth=None, max_features='sqrt',
                     max_leaf_nodes=None, min_impurity_decrease=0.0,
                     min_impurity_split=None, min_samples_leaf=1,
                     min_samples_split=2, min_weight_fraction_leaf=0.0,
                     n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
                     verbose=0, warm_start=False),
                fit_params=None, iid=True, n_jobs=-1,
                param_grid={'n_estimators': [5, 10, 15, 25, 50, 75, 100, 125], 'max_depth': [5
```

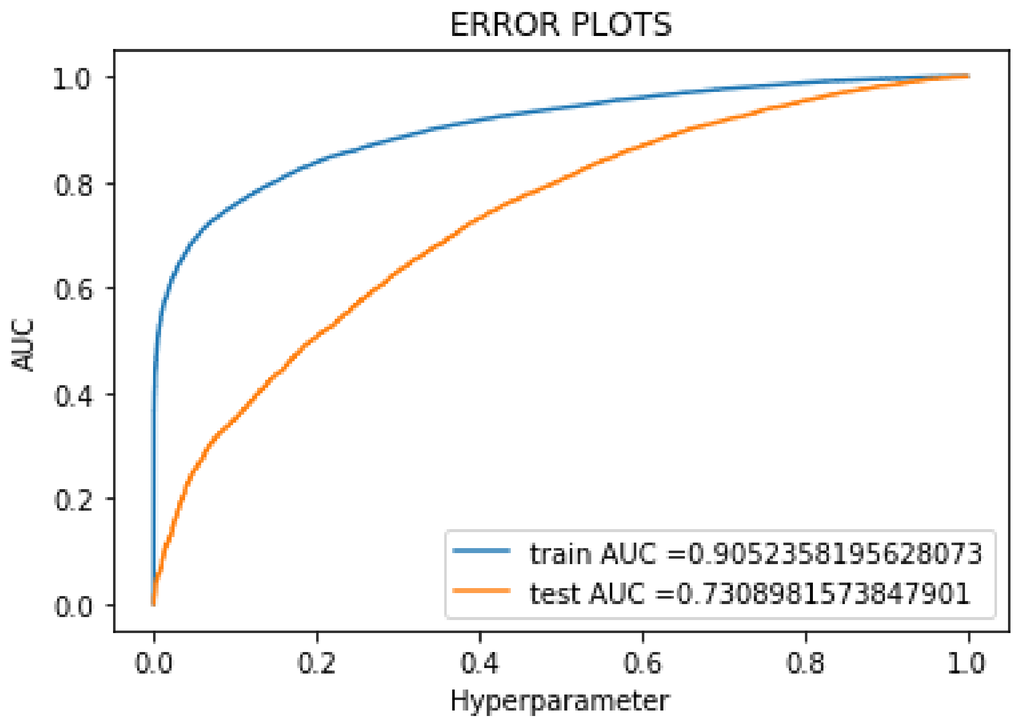
```
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring='roc_auc', verbose=0)
```

```
In [48]: optimal_estimator = grid.best_estimator_.n_estimators
         print("Value for optimal estimator : ",optimal_estimator)

         optimal_depth = grid.best_estimator_.max_depth
         print("Valur for optimal depth : ",optimal_depth)
```

```
Value for optimal estimator :  125
Valur for optimal depth :  10
```

6.1.21 Testing with Test data

```
In [49]: clf = RandomForestClassifier(max_depth= optimal_depth, n_estimators= optimal_estimato
         clf.fit(sent_vectors_train, Y_train)

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

         train_fpr_aw2v, train_tpr_aw2v, thresholds_aw2v = roc_curve(Y_train, clf.predict_proba
         test_fpr_aw2v, test_tpr_aw2v, thresholds_aw2v = roc_curve(Y_test, clf.predict_proba(se

         plt.plot(train_fpr_aw2v, train_tpr_aw2v, label="train AUC ="+str(auc(train_fpr_aw2v, t
         plt.plot(test_fpr_aw2v, test_tpr_aw2v, label="test AUC ="+str(auc(test_fpr_aw2v, test_
         plt.legend()
         plt.xlabel("Hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```

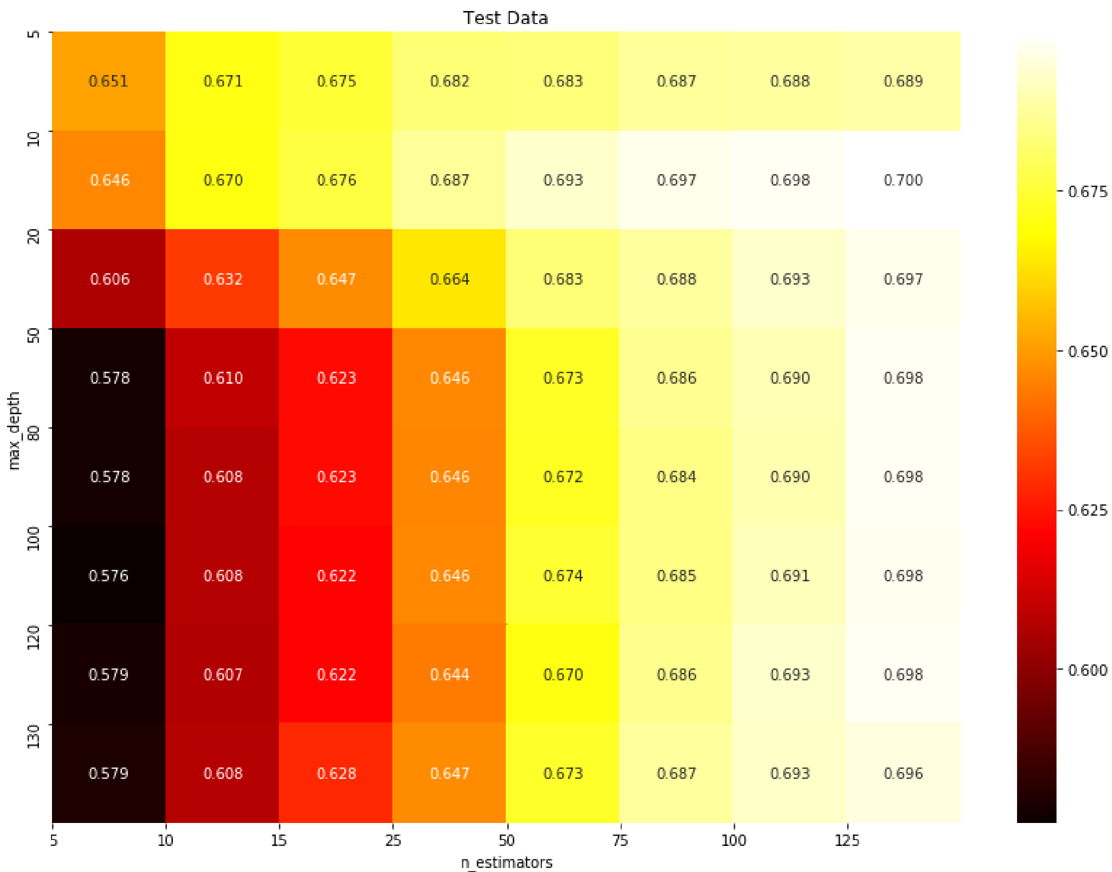



```
In [91]: clf = RandomForestClassifier(n_estimators = optimal_estimator, max_depth = optimal_dep
         clf.fit(sent_vectors_train, Y_train)
         preda = clf.predict(sent_vectors_test)

         acca = accuracy_score(Y_test, preda) * 100
         prea = precision_score(Y_test, preda) * 100
         reca = recall_score(Y_test, preda) * 100
         f1a = f1_score(Y_test, preda) * 100

         print('\nAccuracy=%f%%' % (acca))
         print('\nprecision=%f%%' % (prea))
         print('\nrecall=%f%%' % (reca))
         print('\nF1-Score=%f%%' % (f1a))
```

```
Accuracy=83.658666%

precision=83.657425%

recall=100.000000%

F1-Score=91.101599%
```

6.1.22 [5.5] Heatmap on Test Data

```
In [93]: scores = grid.cv_results_['mean_test_score'].reshape(len(estimator),len(depth))
         plt.figure(figsize=(14,10))
         sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator, yti
         plt.xlabel('n_estimators')
         plt.ylabel('max_depth')
         plt.xticks(np.arange(len(estimator)), estimator)
         plt.yticks(np.arange(len(depth)), depth)
         plt.title('Test Data')
         plt.show()
```



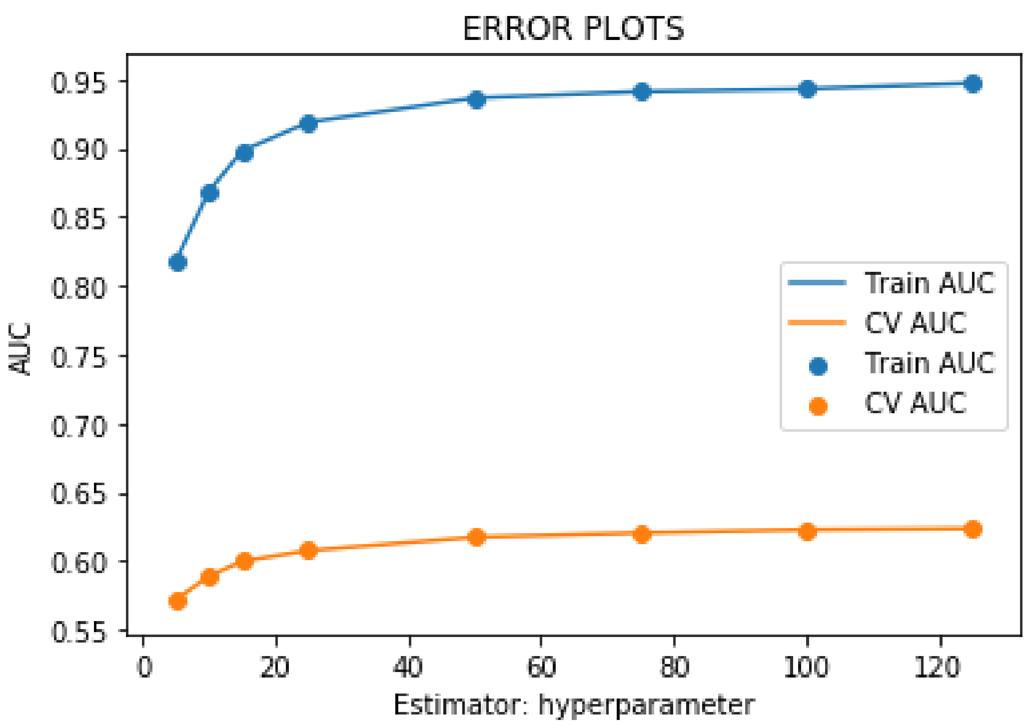
6.1.23 [5.1.6] Applying Random Forests on TFIDF W2V, SET 4

6.1.24 Hyperparameter tuning using GridSearch

```
In [78]: # clf = RandomForestClassifier()
         # for Estimators in Random Forest
         estimator = [5,10,15,25,50,75,100,125]
         parameters = {'n_estimators':[5,10,15,25,50,75,100,125]}
         grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced', max_depth=10), pa
```

```
grid.fit(tfidf_sent_vectors_train, Y_train)

print("best estimator = ", grid.best_params_)

train_auc_tfw2v = grid.cv_results_['mean_train_score']
cv_auc_tfw2v = grid.cv_results_['mean_test_score']

plt.plot(estimator, train_auc_tfw2v, label='Train AUC')
plt.scatter(estimator, train_auc_tfw2v, label='Train AUC')
plt.plot(estimator, cv_auc_tfw2v, label='CV AUC')
plt.scatter(estimator, cv_auc_tfw2v, label='CV AUC')

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

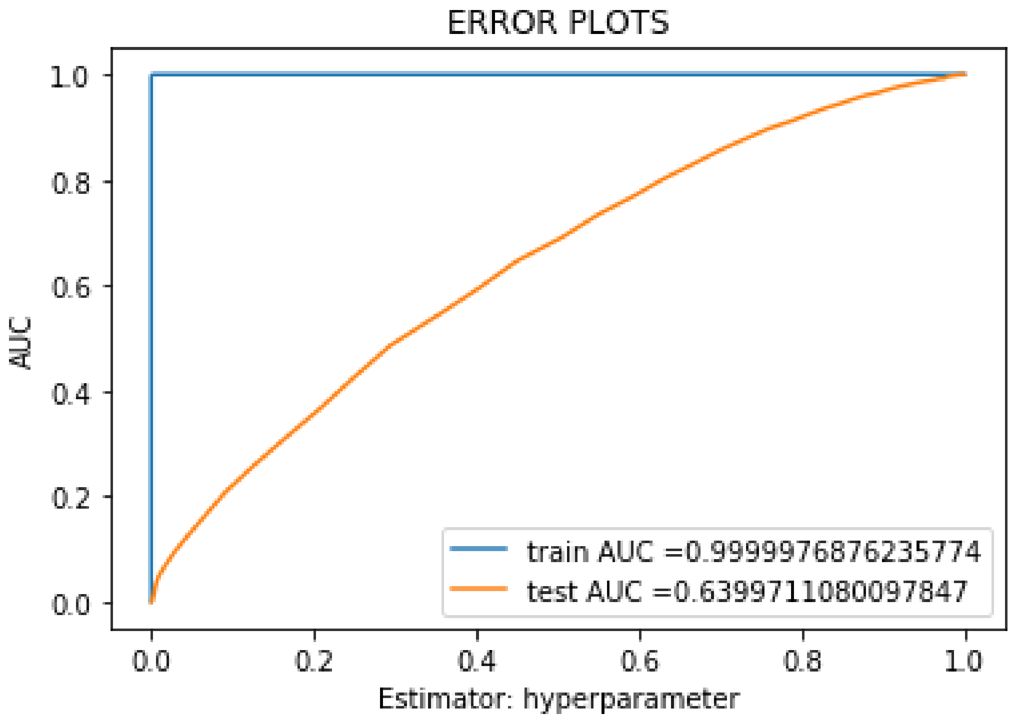
```
best estimator =  {'n_estimators': 125}
```



6.1.25 Testing with Test data

```
In [109]: clf = RandomForestClassifier(n_estimators = 150, class_weight ='balanced')
          clf.fit(tfidf_sent_vectors_train, Y_train)
```

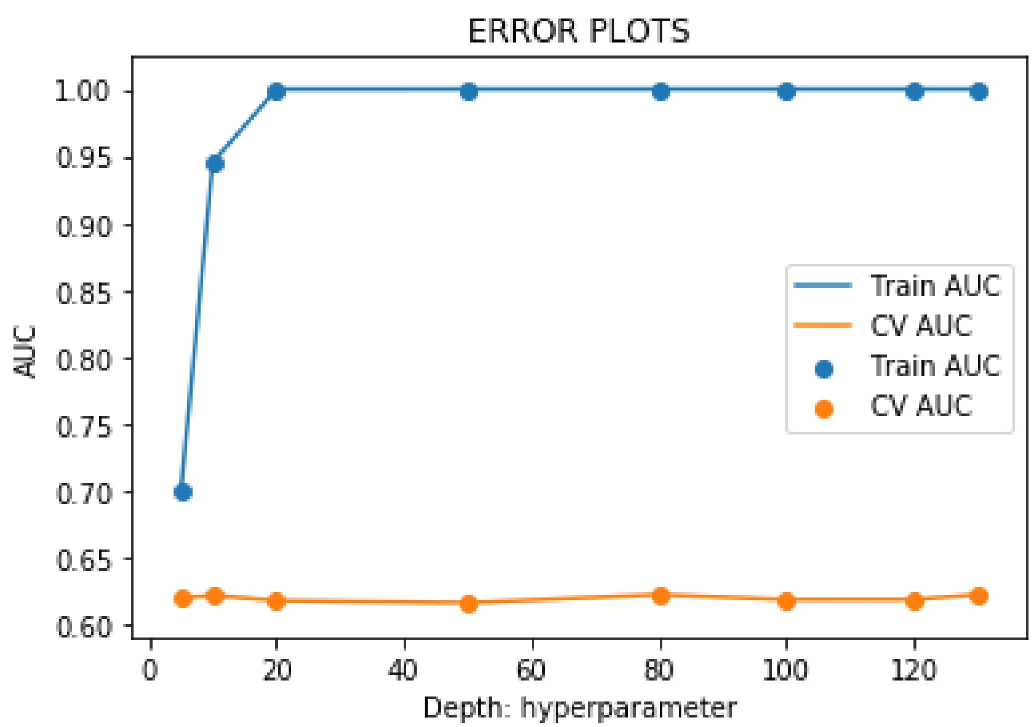
```python
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates o
# not the predicted outputs

train_fpr_tfw2v, train_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_train, clf.predict_p
test_fpr_tfw2v, test_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_test, clf.predict_prol

plt.plot(train_fpr_tfw2v, train_tpr_tfw2v, label="train AUC ="+str(auc(train_fpr_tfw
plt.plot(test_fpr_tfw2v, test_tpr_tfw2v, label="test AUC ="+str(auc(test_fpr_tfw2v,
plt.legend()
plt.xlabel("Estimator: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [83]:
```python
# clf = RandomForestClassifier()
# for Maximum Depth in Random Forest
depth = [5,10,20,50,80,100,120,130]
parameters = {'max_depth':[5,10,20,50,80,100,120,130]}
grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced', n_estimators=100)
grid.fit(tfidf_sent_vectors_train, Y_train)

print("best depth = ", grid.best_params_)
```

```
train_auc_tfw2v = grid.cv_results_['mean_train_score']
cv_auc_tfw2v = grid.cv_results_['mean_test_score']

plt.plot(depth, train_auc_tfw2v, label='Train AUC')
plt.scatter(depth, train_auc_tfw2v, label='Train AUC')
plt.plot(depth, cv_auc_tfw2v, label='CV AUC')
plt.scatter(depth, cv_auc_tfw2v, label='CV AUC')

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

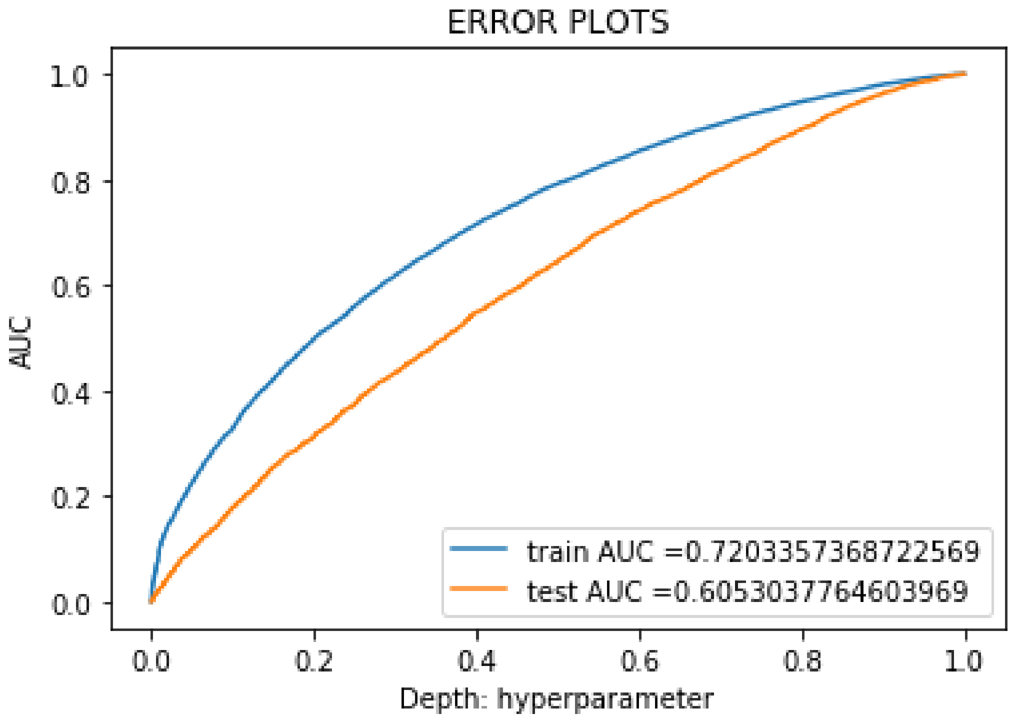
```
best depth =  {'max_depth': 130}
```



6.1.26 Testing with Test data

```
In [89]: clf = RandomForestClassifier(max_depth = 7, class_weight ='balanced')
         clf.fit(tfidf_sent_vectors_train, Y_train)

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

```
train_fpr_tfw2v, train_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_train, clf.predict_pr
test_fpr_tfw2v, test_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_test, clf.predict_proba

plt.plot(train_fpr_tfw2v, train_tpr_tfw2v, label="train AUC ="+str(auc(train_fpr_tfw2v
plt.plot(test_fpr_tfw2v, test_tpr_tfw2v, label="test AUC ="+str(auc(test_fpr_tfw2v, te
plt.legend()
plt.xlabel("Depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [50]:
```
# clf = RandomForestClassifier()
estimator = [5,10,15,25,50,75,100,125]
depth = [5,10,20,50,80,100,120,130]

parameters = {'n_estimators': estimator,'max_depth': depth}
grid = GridSearchCV(RandomForestClassifier(class_weight ='balanced',max_features='sqrt
grid.fit(tfidf_sent_vectors_train, Y_train)
```

Out[50]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
            criterion='gini', max_depth=None, max_features='sqrt',
            max_leaf_nodes=None, min_impurity_decrease=0.0,
```

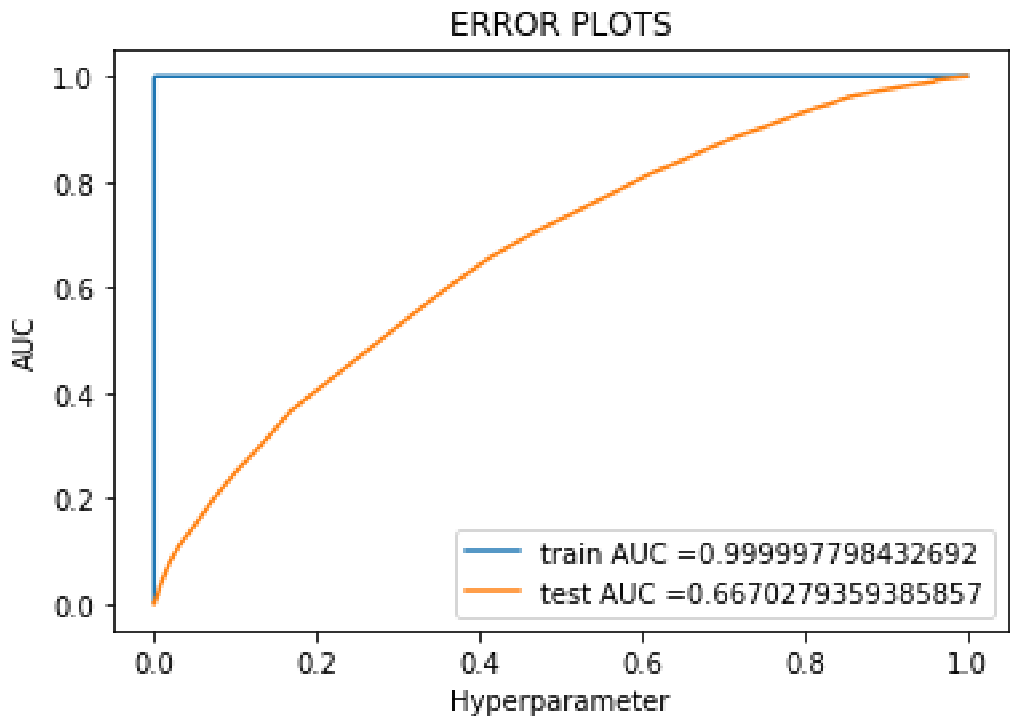
```
                        min_impurity_split=None, min_samples_leaf=1,
                        min_samples_split=2, min_weight_fraction_leaf=0.0,
                        n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
                        verbose=0, warm_start=False),
                fit_params=None, iid=True, n_jobs=-1,
                param_grid={'n_estimators': [5, 10, 15, 25, 50, 75, 100, 125], 'max_depth': [5
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring='roc_auc', verbose=0)
```

```
In [51]: optimal_estimator = grid.best_estimator_.n_estimators
         print("Value for optimal estimator : ",optimal_estimator)

         optimal_depth = grid.best_estimator_.max_depth
         print("Value for optimal depth : ",optimal_depth)
```

```
Value for optimal estimator :  125
Value for optimal depth :  100
```

6.1.27 Testing with Test data

```
In [52]: clf = RandomForestClassifier(max_depth = optimal_depth, n_estimators= optimal_estimat
         clf.fit(tfidf_sent_vectors_train, Y_train)

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

         train_fpr_tfw2v, train_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_train, clf.predict_p
         test_fpr_tfw2v, test_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_test, clf.predict_proba

         plt.plot(train_fpr_tfw2v, train_tpr_tfw2v, label="train AUC ="+str(auc(train_fpr_tfw2
         plt.plot(test_fpr_tfw2v, test_tpr_tfw2v, label="test AUC ="+str(auc(test_fpr_tfw2v, t
         plt.legend()
         plt.xlabel("Hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```



Here difference between Train AUC and Test AUC score is too much

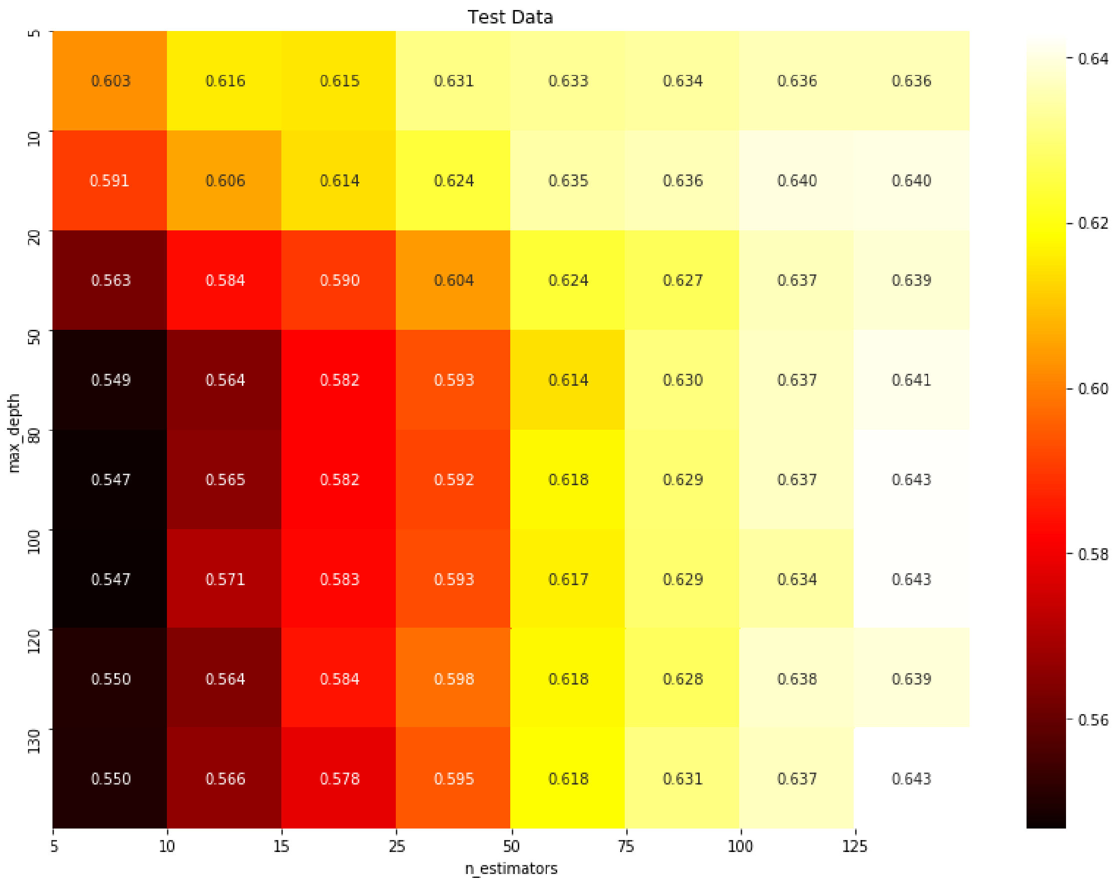
```
In [102]: clf = RandomForestClassifier(n_estimators = optimal_estimator, max_depth = optimal_de
          clf.fit(tfidf_sent_vectors_train, Y_train)
          predw = clf.predict(tfidf_sent_vectors_test)

          accw = accuracy_score(Y_test, predw) * 100
          prew = precision_score(Y_test, predw) * 100
          recw = recall_score(Y_test, predw) * 100
          f1w = f1_score(Y_test, predw) * 100

          print('\nAccuracy=%f%%' % (accw))
          print('\nprecision=%f%%' % (prew))
          print('\nrecall=%f%%' % (recw))
          print('\nF1-Score=%f%%' % (f1w))
```

```
Accuracy=83.742215%

precision=83.763318%

recall=99.936442%
```

F1-Score=91.137930%

6.1.28 [5.5] Heatmap on Test Data

```
In [103]: scores = grid.cv_results_['mean_test_score'].reshape(len(estimator),len(depth))
          plt.figure(figsize=(14,10))
          sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator, y
          plt.xlabel('n_estimators')
          plt.ylabel('max_depth')
          plt.xticks(np.arange(len(estimator)), estimator)
          plt.yticks(np.arange(len(depth)), depth)
          plt.title('Test Data')
          plt.show()
```



NOTE:

1. In Avg-W2V and Tfidf-W2V for n_estimators, difference between Train AUC and Test AUC score is too much. I have tried all the values(5,20,50,100,400,750,1000) for n_estimators while testing on test data. It does not change.
2. Similarly for max_depth, in Avg-W2v and Tfidf-W2v we should take value less than 10 to lower difference between Train AUC and Test AUC score.

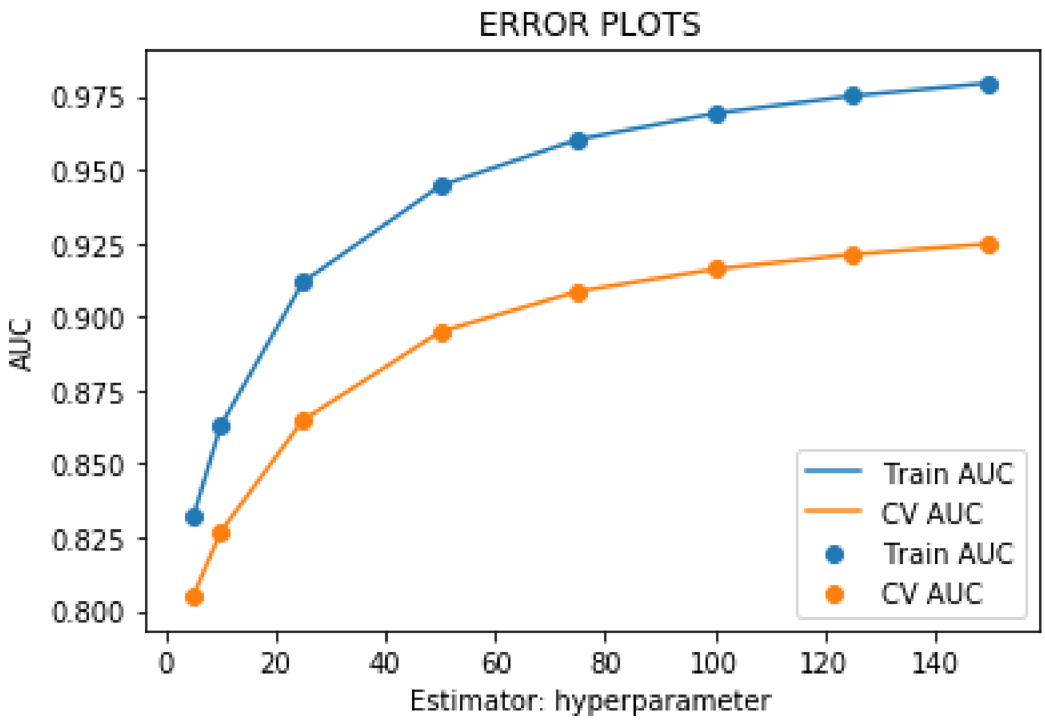
6.2 [5.2] Applying GBDT using XGBOOST

6.2.1 [5.2.1] Applying XGBOOST on BOW, SET 1

6.2.2 Hyperparameter tuning using GridSearch

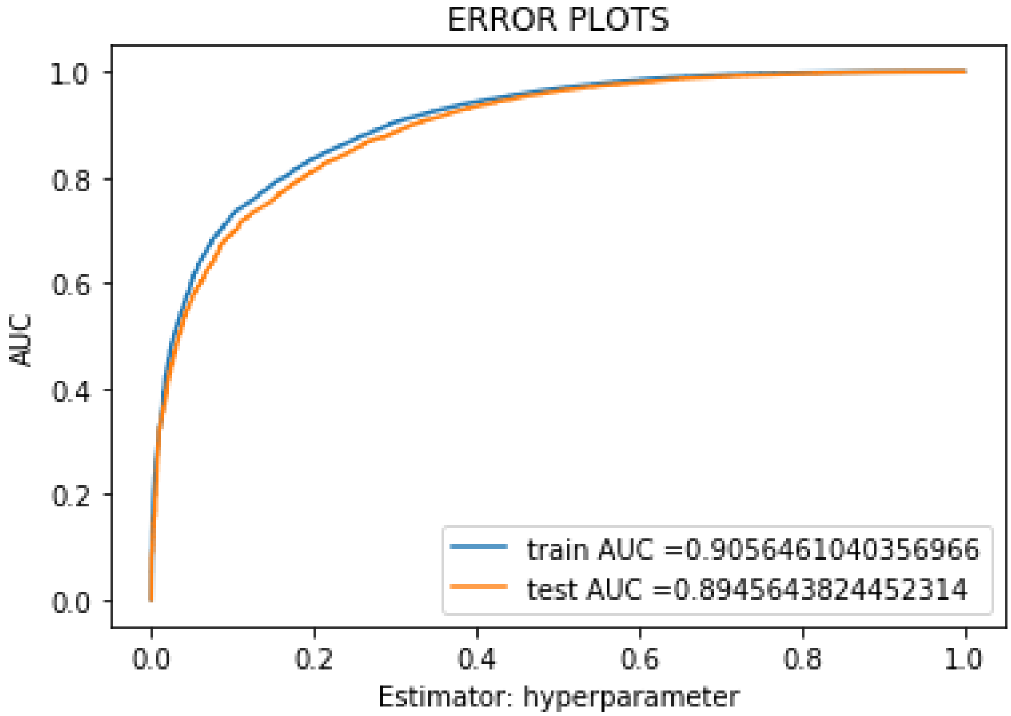
In [95]:
```
#clf = XGBClassifier()
# For Estimator
estimator = [5,10,25,50,75,100,125,150]
parameters = {'n_estimators':[5,10,25,50,75,100,125,150]}
grid = GridSearchCV(XGBClassifier(booster='gbtree',max_depth = 10), parameters, cv=3,
grid.fit(X_train_bow, Y_train)

print("best estimator = ", grid.best_params_)

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()
```

```
best estimator =  {'n_estimators': 150}
```



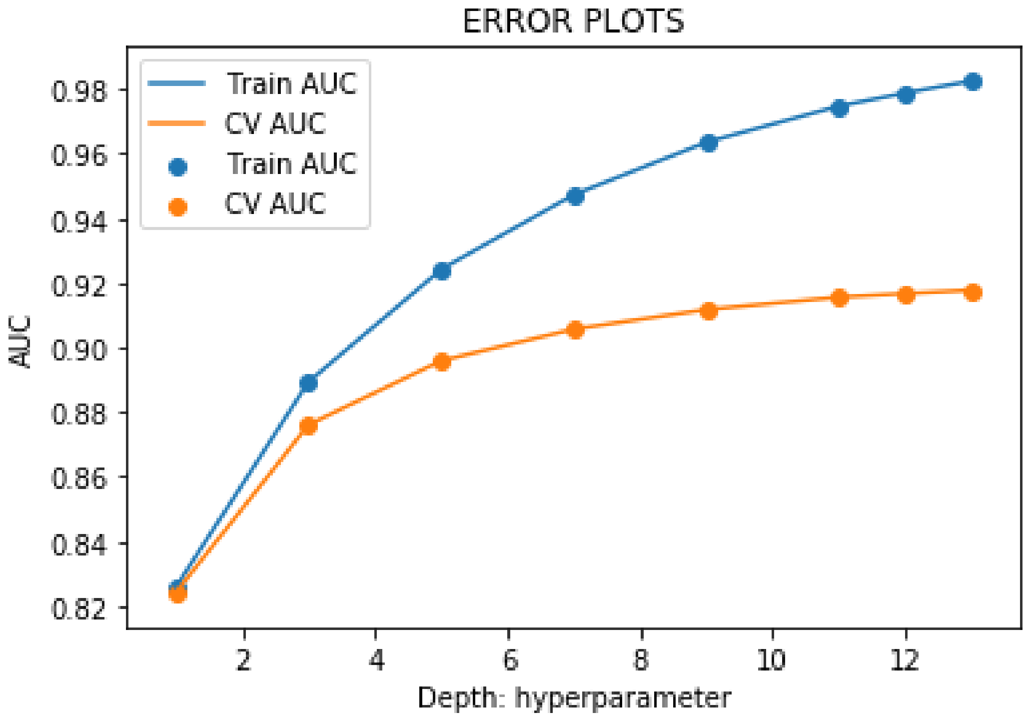
6.2.3 Testing with Test Data

In [96]:
```
clf = XGBClassifier(n_estimators = 150, class_weight ='balanced')
clf.fit(X_train_bow, Y_train)

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

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



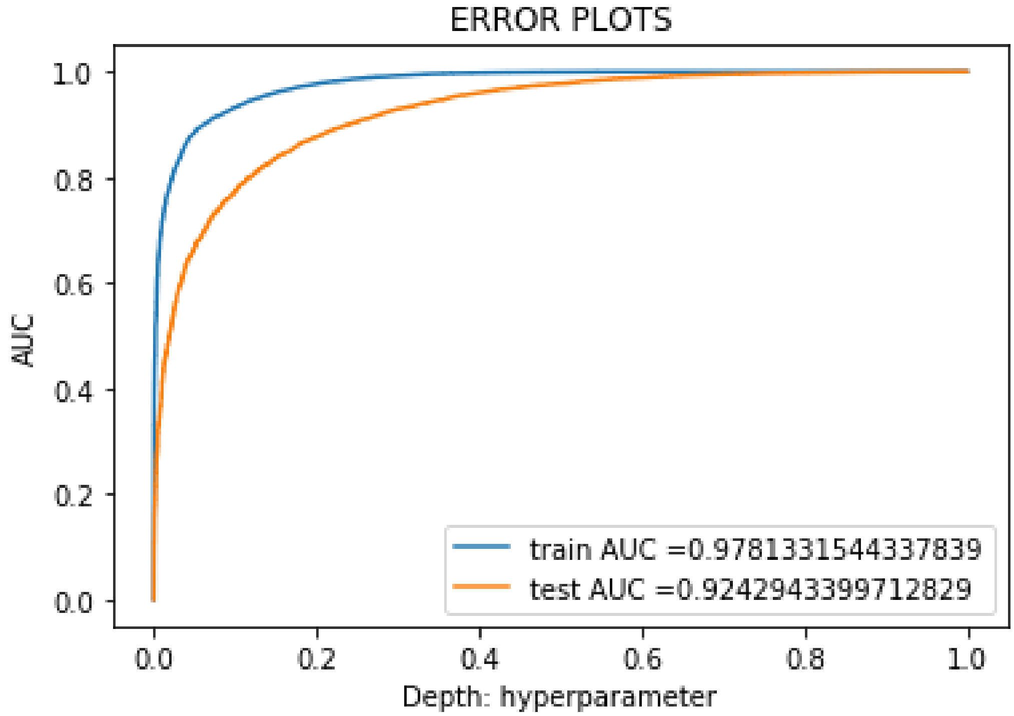

```
In [113]: #clf = XGBClassifier()
          # For Depth
          depth = [1,3,5,7,9,11,12,13]
          parameters = {'max_depth':[1,3,5,7,9,11,12,13]}
          grid = GridSearchCV(XGBClassifier(booster='gbtree',n_estimators = 100), parameters, c
          grid.fit(X_train_bow, Y_train)

          print("best depth = ", grid.best_params_)

          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()
```

```
best depth =  {'max_depth': 13}
```



6.2.4 Testing with Test data

In [97]:
```
clf = XGBClassifier(max_depth = 13, class_weight ='balanced')
clf.fit(X_train_bow, Y_train)

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

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



In [53]:
```
# clf = RandomForestClassifier()
estimator = [5,10,25,50,75,100,125,150]
depth = [1,3,5,7,9,11,12,13]

parameters = {'n_estimators': estimator,'max_depth': depth}
grid = GridSearchCV(XGBClassifier(max_features='sqrt'), parameters, cv=3, scoring='ro
grid.fit(X_train_bow, Y_train)
```

Out[53]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, max_features='sqrt',
       min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
       nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'n_estimators': [5, 10, 25, 50, 75, 100, 125, 150], 'max_depth': [
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

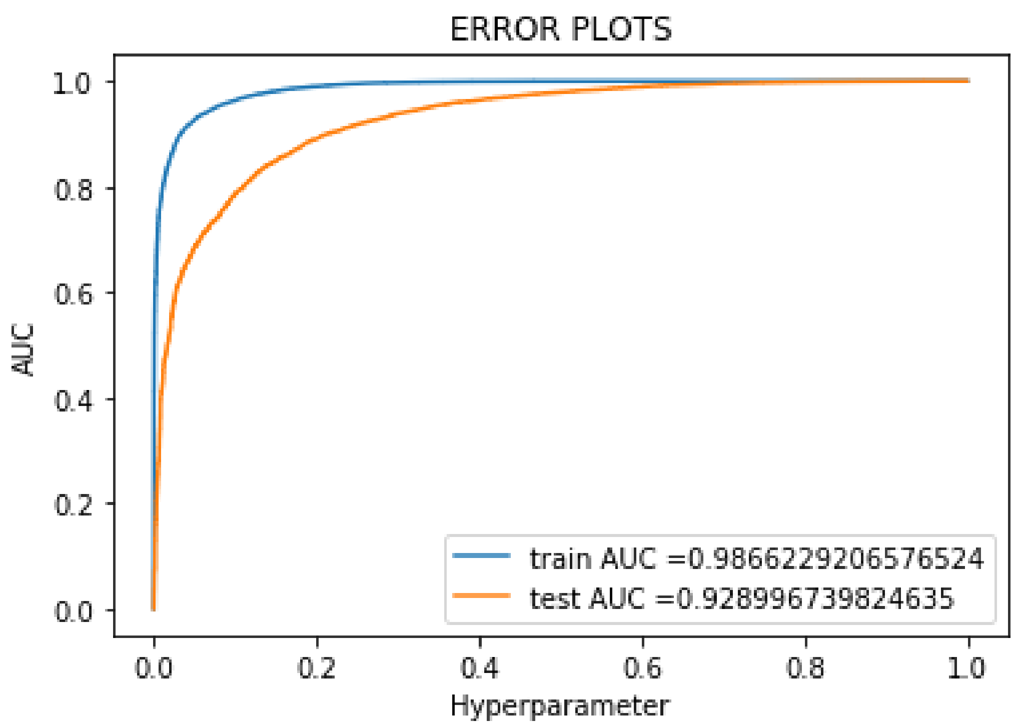
In [54]:
```
optimal_estimator = grid.best_estimator_.n_estimators
print("Value for optimal estimator : ",optimal_estimator)
```

```
optimal_depth = grid.best_estimator_.max_depth
print("Value for optimal depth : ",optimal_depth)
```

```
Value for optimal estimator :  150
Value for optimal depth :  13
```

6.2.5 Testing with Test data

In [55]:
```
clf = XGBClassifier(max_depth = optimal_depth, n_estimators= optimal_estimator, class_
clf.fit(X_train_bow, Y_train)

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

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()
```

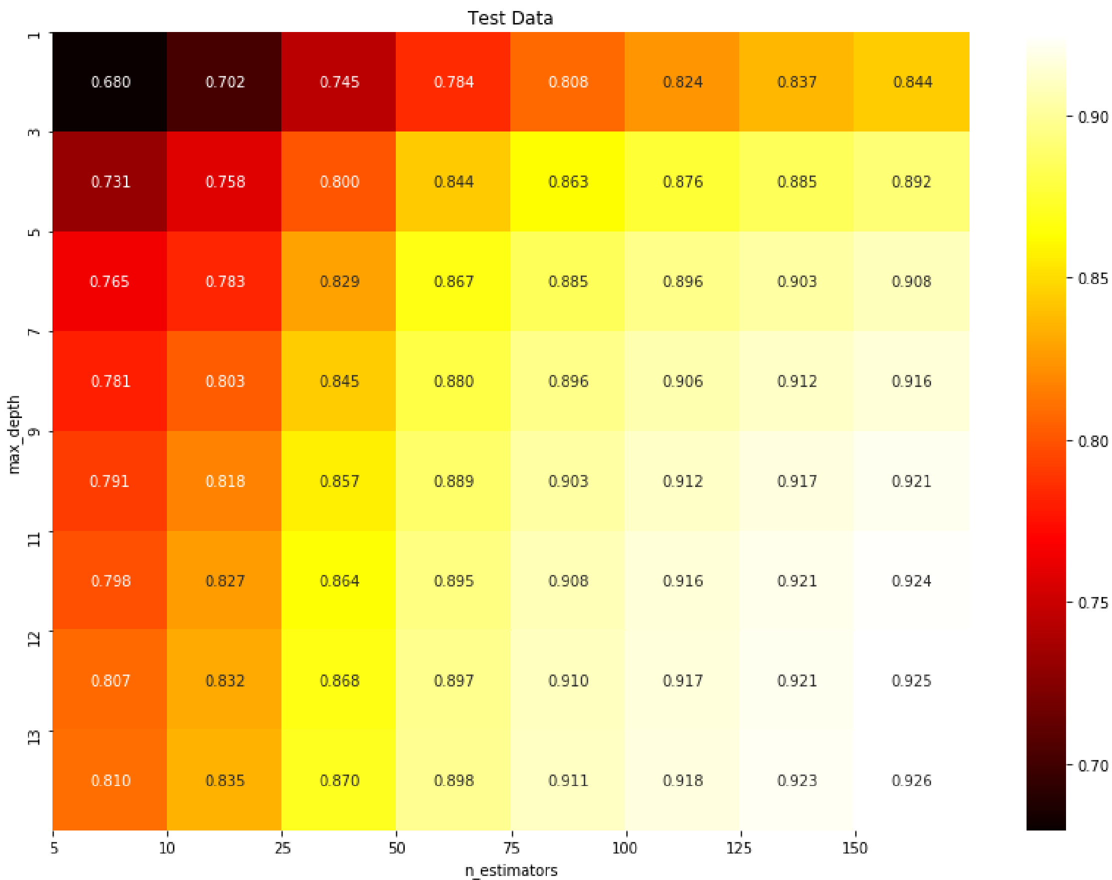


In [117]:
```
clf = XGBClassifier(n_estimators = optimal_estimator, max_depth = optimal_depth)
clf.fit(X_train_bow, Y_train)
predb = clf.predict(X_test_bow)

accb = accuracy_score(Y_test, predb) * 100
preb = precision_score(Y_test, predb) * 100
recb = recall_score(Y_test, predb) * 100
f1b = f1_score(Y_test, predb) * 100

print('\nAccuracy=%f%%' % (accb))
print('\nprecision=%f%%' % (preb))
print('\nrecall=%f%%' % (recb))
print('\nF1-Score=%f%%' % (f1b))
```

```
Accuracy=90.361537%

precision=91.048486%

recall=98.125028%

F1-Score=94.454398%
```

6.2.6 Heatmap on Test data

In [118]:
```
scores = grid.cv_results_['mean_test_score'].reshape(len(estimator),len(depth))
plt.figure(figsize=(14,10))
sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator, y
plt.xlabel('n_estimators')
plt.ylabel('max_depth')
plt.xticks(np.arange(len(estimator)), estimator)
plt.yticks(np.arange(len(depth)), depth)
plt.title('Test Data')
plt.show()
```



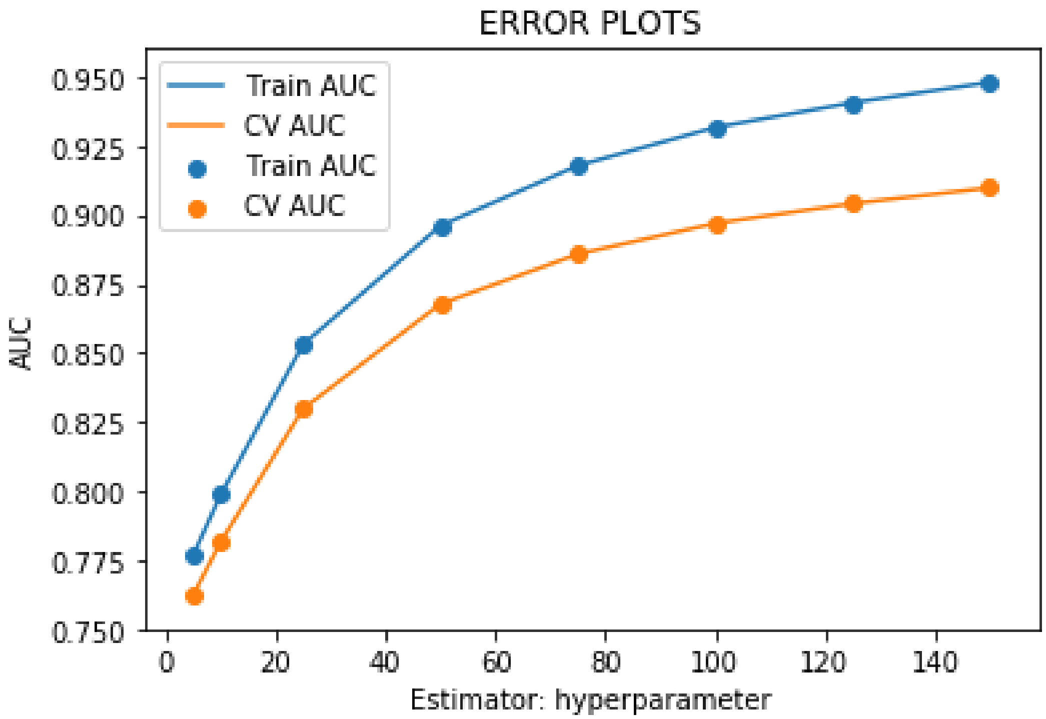
6.2.7 [5.2.2] Applying XGBOOST on TFIDF, SET 2

6.2.8 Hyperparameter tuning using GridSearch

```
In [119]: #clf = XGBClassifier()
          # For Estimator
          estimator = [5,10,25,50,75,100,125,150]
          parameters = {'n_estimators':[5,10,25,50,75,100,125,150]}
          grid = GridSearchCV(XGBClassifier(booster='gbtree',max_depth = 5), parameters, cv=3,
          grid.fit(X_train_tfidf, Y_train)

          print("best estimator = ", grid.best_params_)

          train_auc_tfidf = grid.cv_results_['mean_train_score']
          cv_auc_tfidf = grid.cv_results_['mean_test_score']

          plt.plot(estimator, train_auc_tfidf, label='Train AUC')
          plt.scatter(estimator, train_auc_tfidf, label='Train AUC')
          plt.plot(estimator, cv_auc_tfidf, label='CV AUC')
          plt.scatter(estimator, cv_auc_tfidf, label='CV AUC')
```

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

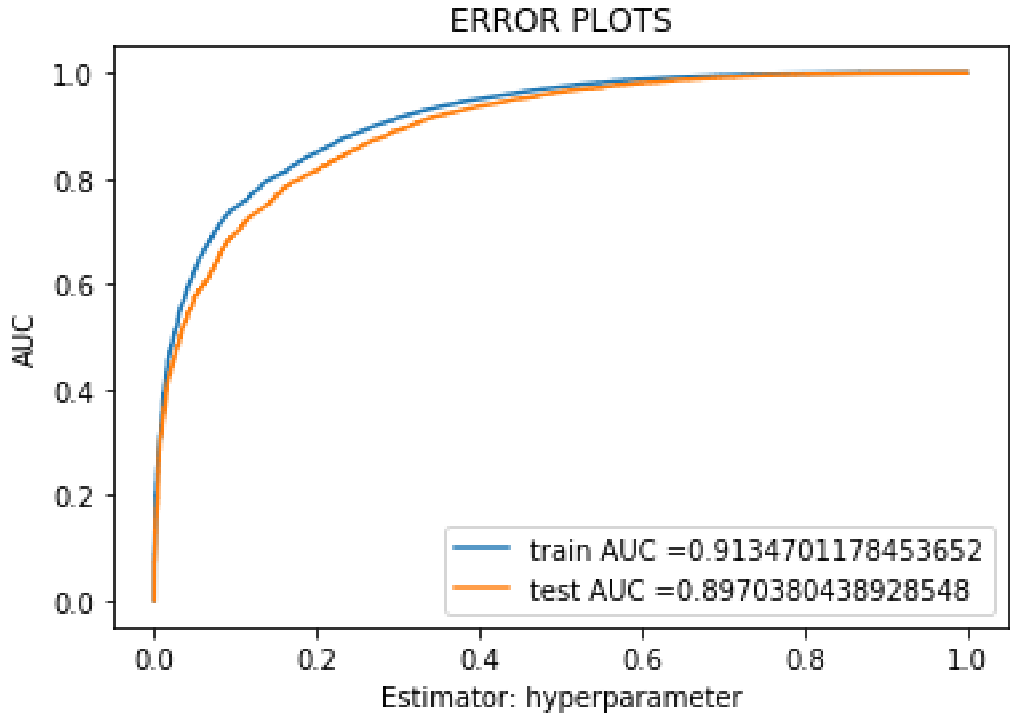
```
best estimator =  {'n_estimators': 150}
```



6.2.9 Testing with Test data

```
In [98]: clf = XGBClassifier(n_estimators = 150, class_weight ='balanced')
         clf.fit(X_train_tfidf, Y_train)

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

         train_fpr_tfidf, train_tpr_tfidf, thresholds_tfidf = roc_curve(Y_train, clf.predict_p
         test_fpr_tfidf, test_tpr_tfidf, thresholds_tfidf = roc_curve(Y_test, clf.predict_proba

         plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="train AUC ="+str(auc(train_fpr_tfid
         plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="test AUC ="+str(auc(test_fpr_tfidf, te
         plt.legend()
```

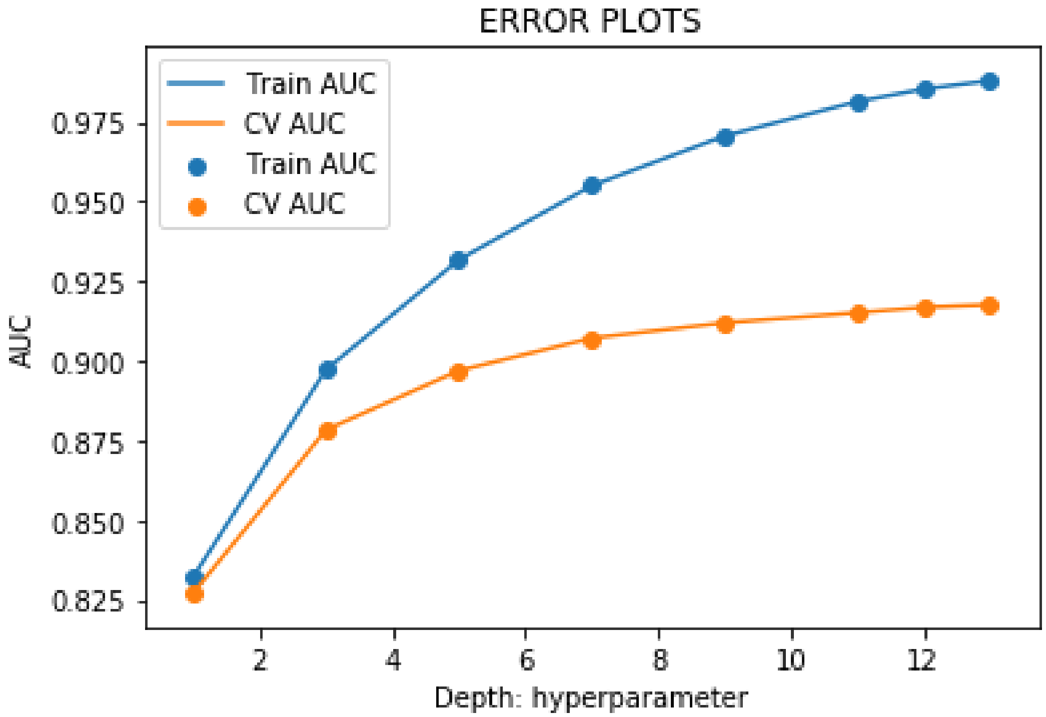
```
plt.xlabel("Estimator: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```




```
In [120]: #clf = XGBClassifier()
          # For Depth
          depth = [1,3,5,7,9,11,12,13]
          parameters = {'max_depth':[1,3,5,7,9,11,12,13]}
          grid = GridSearchCV(XGBClassifier(booster='gbtree',n_estimators = 100), parameters, c
          grid.fit(X_train_tfidf, Y_train)

          print("best depth = ", grid.best_params_)

          train_auc_tfidf = grid.cv_results_['mean_train_score']
          cv_auc_tfidf = grid.cv_results_['mean_test_score']

          plt.plot(depth, train_auc_tfidf, label='Train AUC')
          plt.scatter(depth, train_auc_tfidf, label='Train AUC')
          plt.plot(depth, cv_auc_tfidf, label='CV AUC')
          plt.scatter(depth, cv_auc_tfidf, label='CV AUC')

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

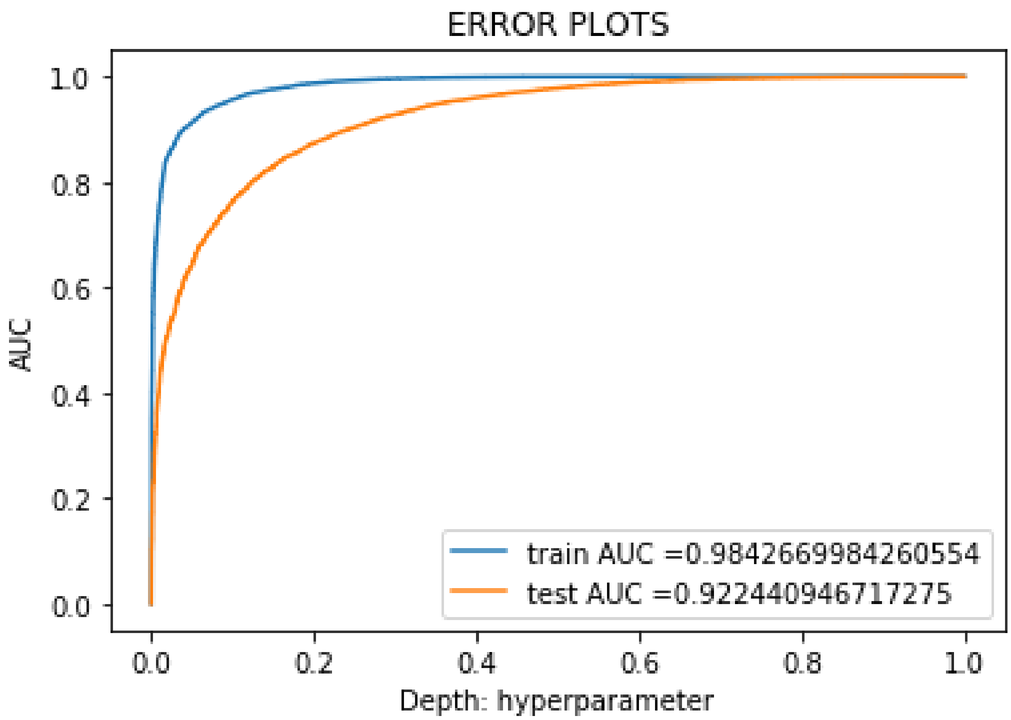
```
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.show()
```

```
best depth =  {'max_depth': 13}
```



6.2.10 Testing with Test data

```
In [99]: clf = XGBClassifier(max_depth = 13, class_weight ='balanced')
         clf.fit(X_train_tfidf, Y_train)

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

         train_fpr_tfidf, train_tpr_tfidf, thresholds_tfidf = roc_curve(Y_train, clf.predict_p
         test_fpr_tfidf, test_tpr_tfidf, thresholds_tfidf = roc_curve(Y_test, clf.predict_proba

         plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="train AUC ="+str(auc(train_fpr_tfid
         plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="test AUC ="+str(auc(test_fpr_tfidf, t
         plt.legend()
         plt.xlabel("Depth: hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```



In [56]:
```
# clf = XGBClassifier()
estimator = [5,10,25,50,75,100,125,150]
depth = [1,3,5,7,9,11,12,13]

parameters = {'n_estimators': estimator,'max_depth': depth}
grid = GridSearchCV(XGBClassifier(max_features='sqrt'), parameters, cv=3, scoring='roc
grid.fit(X_train_tfidf, Y_train)
```

Out[56]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, max_features='sqrt',
       min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
       nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'n_estimators': [5, 10, 25, 50, 75, 100, 125, 150], 'max_depth': [
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

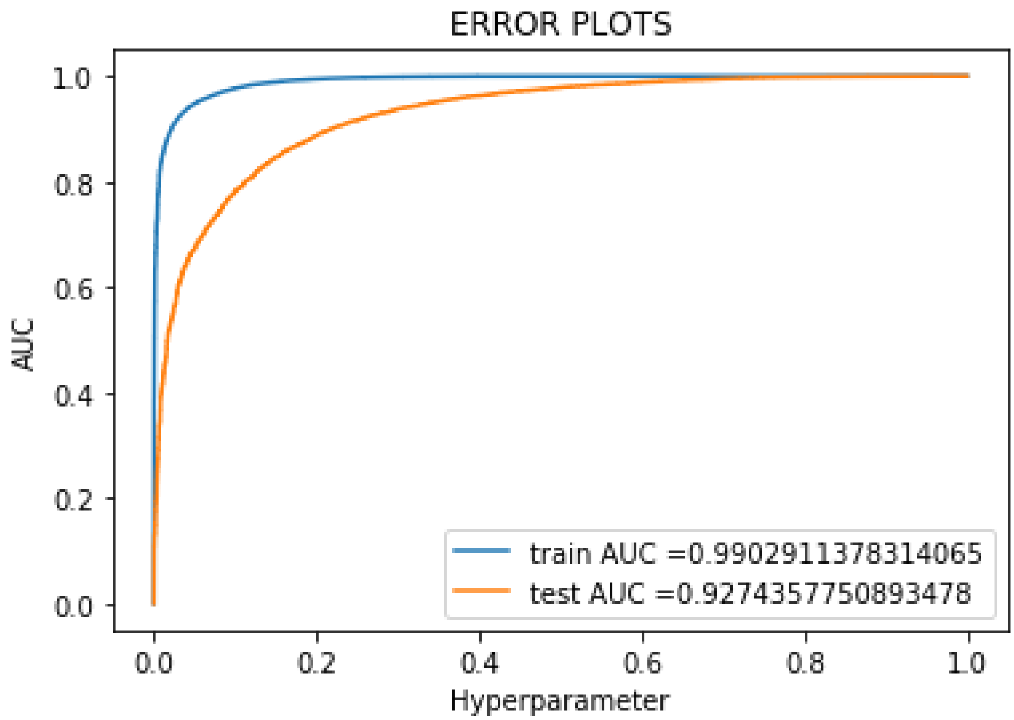
In [57]:
```
optimal_estimator = grid.best_estimator_.n_estimators
print("Value for optimal estimator : ",optimal_estimator)
```

```
        optimal_depth = grid.best_estimator_.max_depth
        print("Value for optimal depth : ",optimal_depth)
```

```
Value for optimal estimator :  150
Value for optimal depth :  13
```

6.2.11 Testing with Test data

```
In [58]: clf = XGBClassifier(max_depth = optimal_depth, n_estimators= optimal_estimator, class_
         clf.fit(X_train_tfidf, Y_train)

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

         train_fpr_tfidf, train_tpr_tfidf, thresholds_tfidf = roc_curve(Y_train, clf.predict_p
         test_fpr_tfidf, test_tpr_tfidf, thresholds_tfidf = roc_curve(Y_test, clf.predict_proba

         plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="train AUC ="+str(auc(train_fpr_tfid
         plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="test AUC ="+str(auc(test_fpr_tfidf, te
         plt.legend()
         plt.xlabel("Hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```

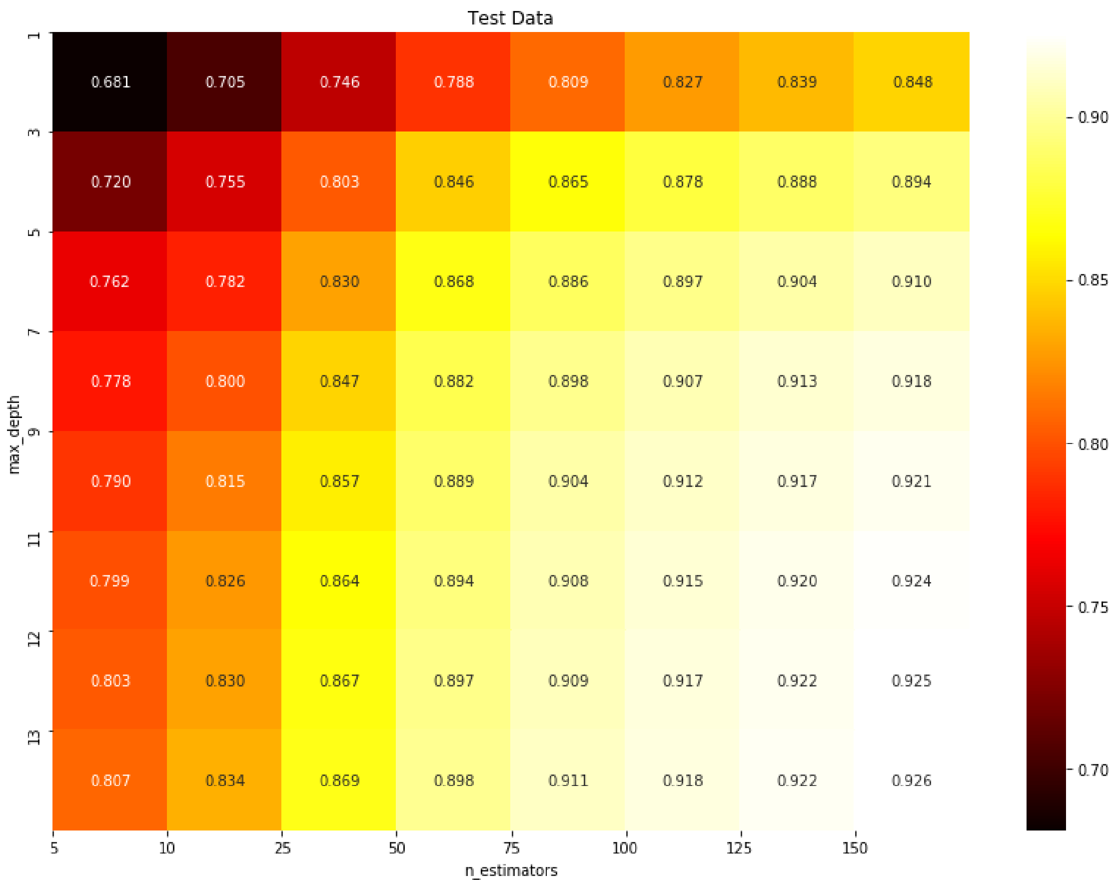


```
In [125]: clf = XGBClassifier(n_estimators = optimal_estimator, max_depth = optimal_depth)
          clf.fit(X_train_tfidf, Y_train)
          predt1 = clf.predict(X_test_tfidf)

          acct1 = accuracy_score(Y_test, predt1) * 100
          pret1 = precision_score(Y_test, predt1) * 100
          rect1 = recall_score(Y_test, predt1) * 100
          f1t1 = f1_score(Y_test, predt1) * 100

          print('\nAccuracy=%f%%' % (acct1))
          print('\nprecision=%f%%' % (pret1))
          print('\nrecall=%f%%' % (rect1))
          print('\nF1-Score=%f%%' % (f1t1))
```

```
Accuracy=90.236214%

precision=90.908326%

recall=98.143188%

F1-Score=94.387321%
```

6.2.12 Heatmap on Test data

```
In [126]: scores = grid.cv_results_['mean_test_score'].reshape(len(estimator),len(depth))
          plt.figure(figsize=(14,10))
          sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator, y
          plt.xlabel('n_estimators')
          plt.ylabel('max_depth')
          plt.xticks(np.arange(len(estimator)), estimator)
          plt.yticks(np.arange(len(depth)), depth)
          plt.title('Test Data')
          plt.show()
```



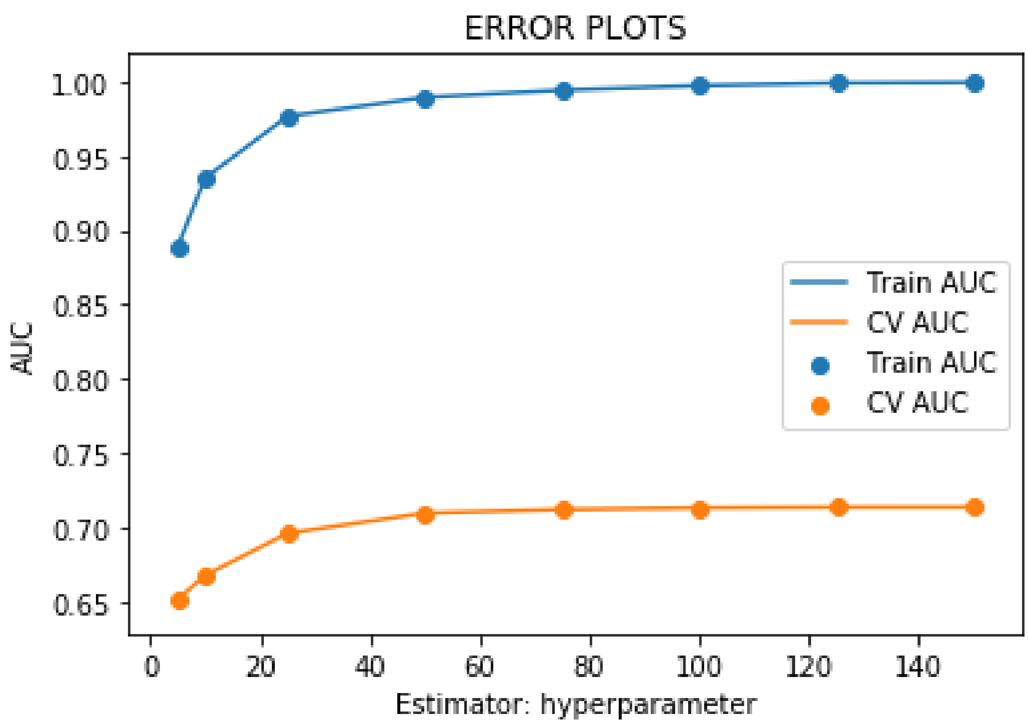
6.2.13 [5.2.3] Applying XGBOOST on AVG W2V, SET 3

6.2.14 Hyperparameter tuning using GridSearch

```
In [128]: #clf = XGBClassifier()
          # For Estimator
          estimator = [5,10,25,50,75,100,125,150]
          parameters = {'n_estimators':[5,10,25,50,75,100,125,150]}
          grid = GridSearchCV(XGBClassifier(booster='gbtree',max_depth = 10), parameters, cv=3
          grid.fit(sent_vectors_train, Y_train)

          print("best estimator = ", grid.best_params_)

          train_auc_aw2v = grid.cv_results_['mean_train_score']
          cv_auc_aw2v = grid.cv_results_['mean_test_score']

          plt.plot(estimator, train_auc_aw2v, label='Train AUC')
          plt.scatter(estimator, train_auc_aw2v, label='Train AUC')
          plt.plot(estimator, cv_auc_aw2v, label='CV AUC')
          plt.scatter(estimator, cv_auc_aw2v, label='CV AUC')
```

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

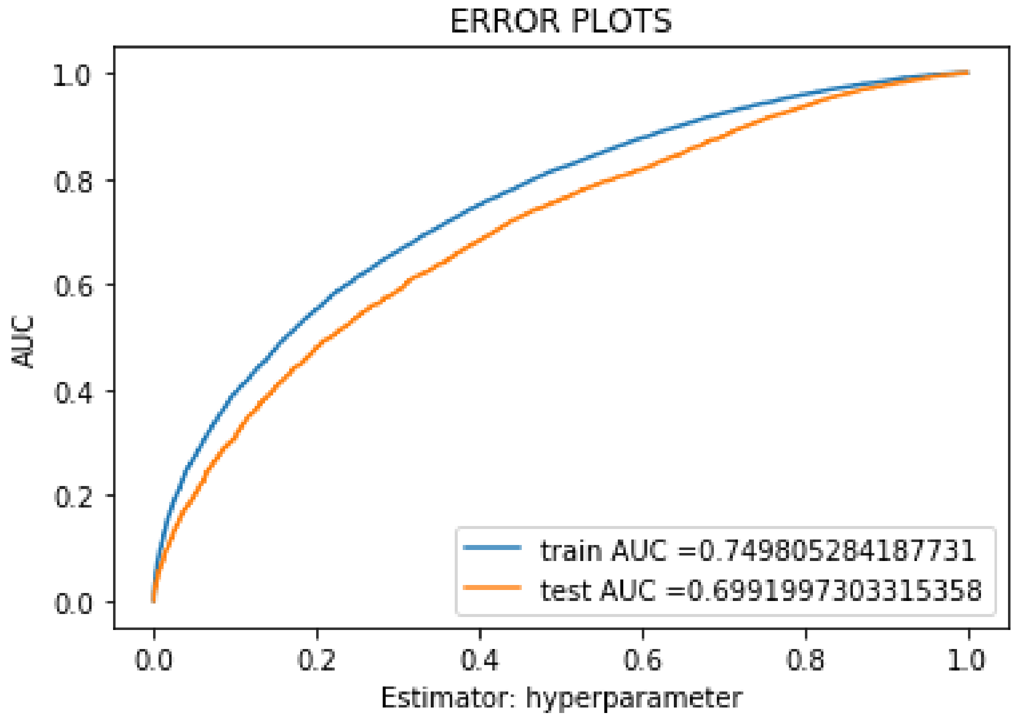
```
best estimator =  {'n_estimators': 150}
```



6.2.15 Testing with Test data

```
In [100]: clf = XGBClassifier(n_estimators = 150, class_weight ='balanced')
          clf.fit(sent_vectors_train, Y_train)

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

          train_fpr_aw2v, train_tpr_aw2v, thresholds_aw2v = roc_curve(Y_train, clf.predict_prol
          test_fpr_aw2v, test_tpr_aw2v, thresholds_aw2v = roc_curve(Y_test, clf.predict_proba(s

          plt.plot(train_fpr_aw2v, train_tpr_aw2v, label="train AUC ="+str(auc(train_fpr_aw2v,
          plt.plot(test_fpr_aw2v, test_tpr_aw2v, label="test AUC ="+str(auc(test_fpr_aw2v, tes
          plt.legend()
```

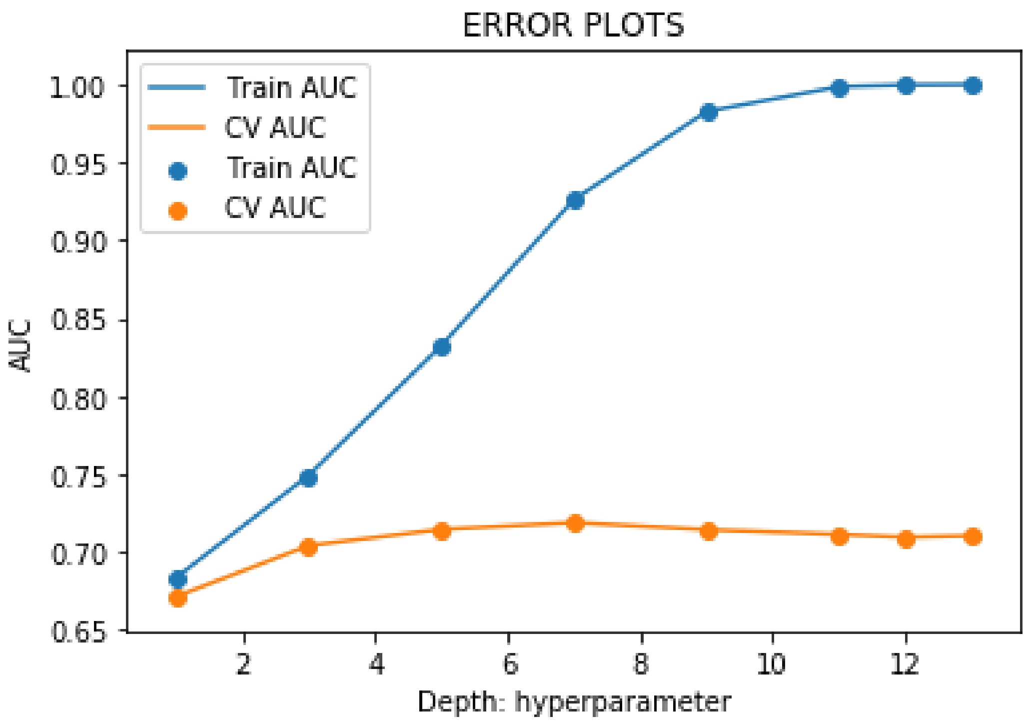
```
plt.xlabel("Estimator: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



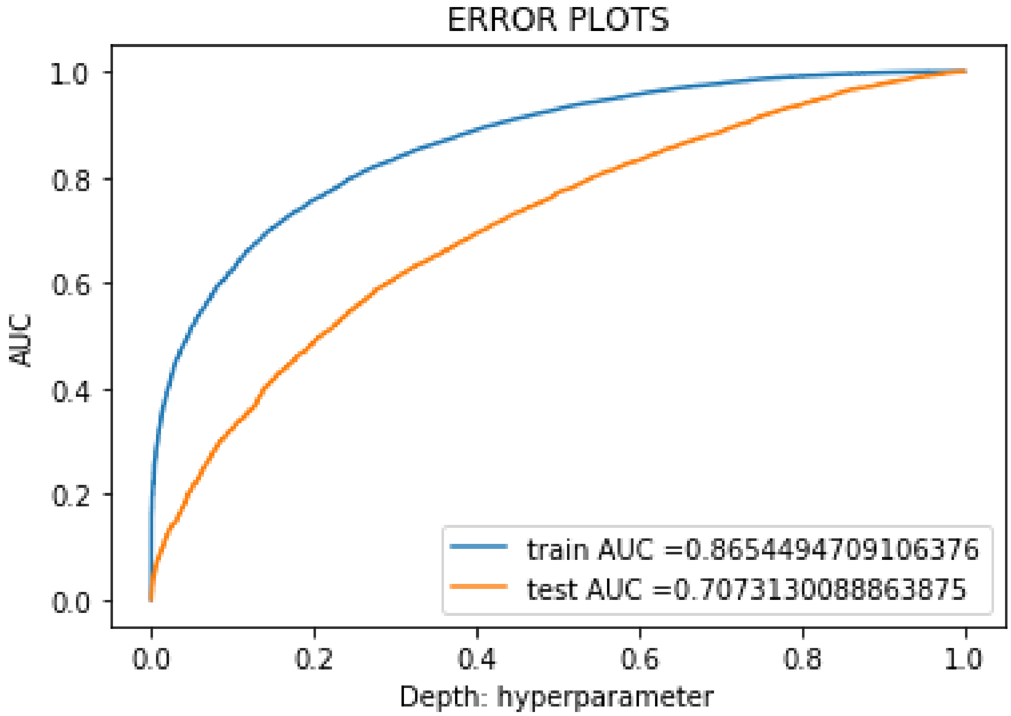

```
In [129]: #clf = XGBClassifier()
          # For Depth
          depth = [1,3,5,7,9,11,12,13]
          parameters = {'max_depth':[1,3,5,7,9,11,12,13]}
          grid = GridSearchCV(XGBClassifier(booster='gbtree',n_estimators = 70), parameters, c
          grid.fit(sent_vectors_train, Y_train)

          print("best depth = ", grid.best_params_)

          train_auc_aw2v = grid.cv_results_['mean_train_score']
          cv_auc_aw2v = grid.cv_results_['mean_test_score']

          plt.plot(depth, train_auc_aw2v, label='Train AUC')
          plt.scatter(depth, train_auc_aw2v, label='Train AUC')
          plt.plot(depth, cv_auc_aw2v, label='CV AUC')
          plt.scatter(depth, cv_auc_aw2v, label='CV AUC')

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

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

```
best depth =  {'max_depth': 7}
```



In [103]:
```
clf = XGBClassifier(max_depth = 6, class_weight ='balanced')
clf.fit(sent_vectors_train, Y_train)

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

train_fpr_aw2v, train_tpr_aw2v, thresholds_aw2v = roc_curve(Y_train, clf.predict_prob
test_fpr_aw2v, test_tpr_aw2v, thresholds_aw2v = roc_curve(Y_test, clf.predict_proba(s

plt.plot(train_fpr_aw2v, train_tpr_aw2v, label="train AUC ="+str(auc(train_fpr_aw2v,
plt.plot(test_fpr_aw2v, test_tpr_aw2v, label="test AUC ="+str(auc(test_fpr_aw2v, tes
plt.legend()
plt.xlabel("Depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```



In [59]:
```
# clf = XGBClassifier()
estimator = [5,10,25,50,75,100,125,150]
depth = [1,3,5,7,9,11,12,13]

parameters = {'n_estimators': estimator,'max_depth': depth}
grid = GridSearchCV(XGBClassifier(max_features='sqrt'), parameters, cv=3, scoring='roc
grid.fit(sent_vectors_train, Y_train)
```

Out[59]:
```
GridSearchCV(cv=3, error_score='raise',
       estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, max_features='sqrt',
       min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
       nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'n_estimators': [5, 10, 25, 50, 75, 100, 125, 150], 'max_depth': [
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

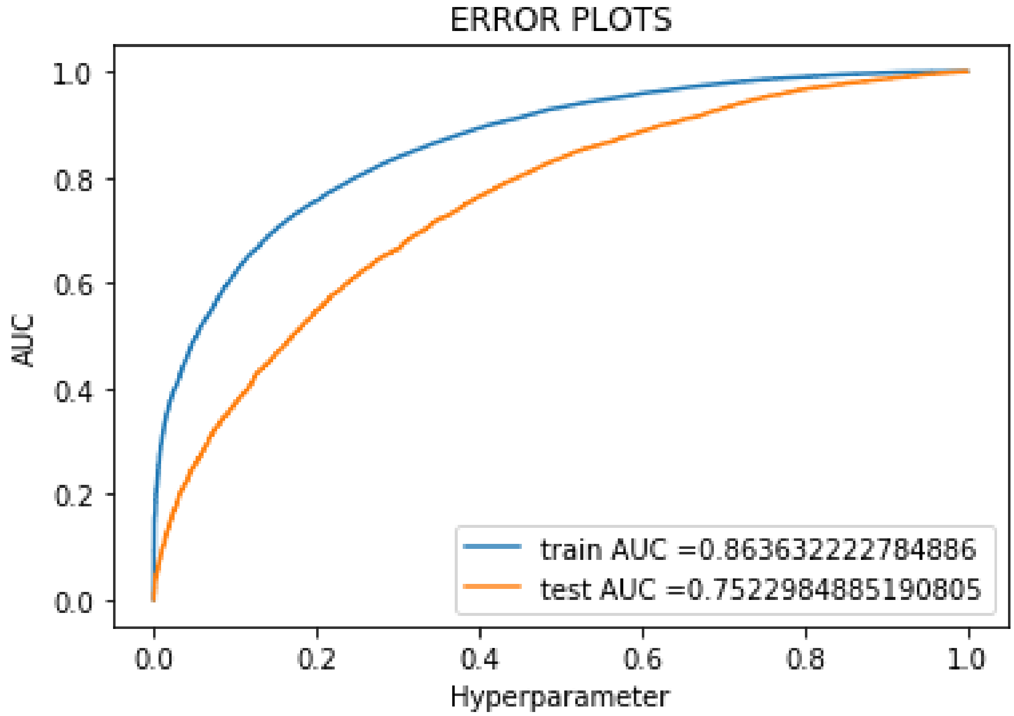
In [60]:
```
optimal_estimator = grid.best_estimator_.n_estimators
print("Value for optimal estimator : ",optimal_estimator)
```

```
    optimal_depth = grid.best_estimator_.max_depth
    print("Value for optimal depth : ",optimal_depth)
```

```
Value for optimal estimator :  150
Value for optimal depth :  5
```

6.2.16 Testing with Test data

```
In [61]: clf = XGBClassifier(max_depth = optimal_depth, n_estimators= optimal_estimator, class_
         clf.fit(sent_vectors_train, Y_train)

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

         train_fpr_aw2v, train_tpr_aw2v, thresholds_aw2v = roc_curve(Y_train, clf.predict_proba
         test_fpr_aw2v, test_tpr_aw2v, thresholds_aw2v = roc_curve(Y_test, clf.predict_proba(se

         plt.plot(train_fpr_aw2v, train_tpr_aw2v, label="train AUC ="+str(auc(train_fpr_aw2v, t
         plt.plot(test_fpr_aw2v, test_tpr_aw2v, label="test AUC ="+str(auc(test_fpr_aw2v, test_
         plt.legend()
         plt.xlabel("Hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```

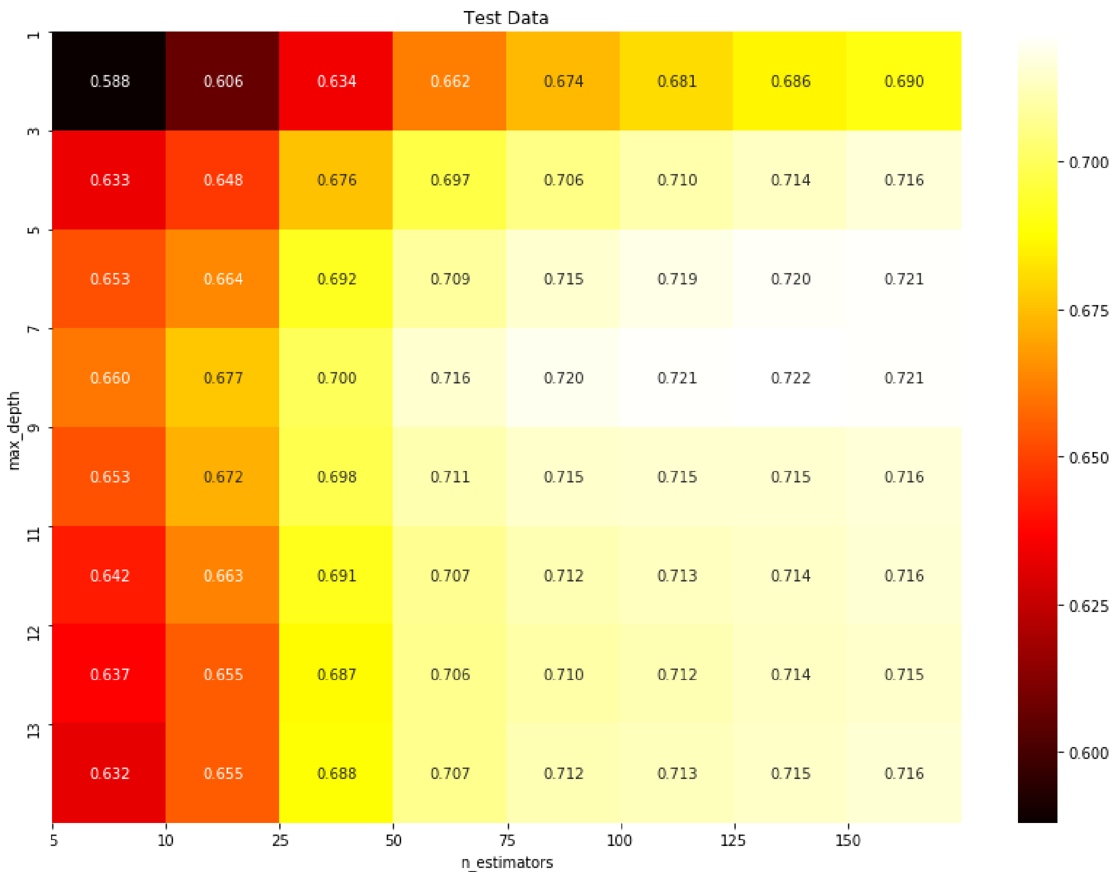


```
In [133]: clf = XGBClassifier(n_estimators = optimal_estimator, max_depth = optimal_depth)
          clf.fit(sent_vectors_train, Y_train)
          preda1 = clf.predict(sent_vectors_test)

          acca1 = accuracy_score(Y_test, preda1) * 100
          prea1 = precision_score(Y_test, preda1) * 100
          reca1 = recall_score(Y_test, preda1) * 100
          f1a1 = f1_score(Y_test, preda1) * 100

          print('\nAccuracy=%f%%' % (acca1))
          print('\nprecision=%f%%' % (prea1))
          print('\nrecall=%f%%' % (reca1))
          print('\nF1-Score=%f%%' % (f1a1))
```

```
Accuracy=83.928300%

precision=84.338698%

recall=99.210060%

F1-Score=91.171930%
```

6.2.17 Heatmap for Test Data

```
In [134]: scores = grid.cv_results_['mean_test_score'].reshape(len(estimator),len(depth))
          plt.figure(figsize=(14,10))
          sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator, y
          plt.xlabel('n_estimators')
          plt.ylabel('max_depth')
          plt.xticks(np.arange(len(estimator)), estimator)
          plt.yticks(np.arange(len(depth)), depth)
          plt.title('Test Data')
          plt.show()
```



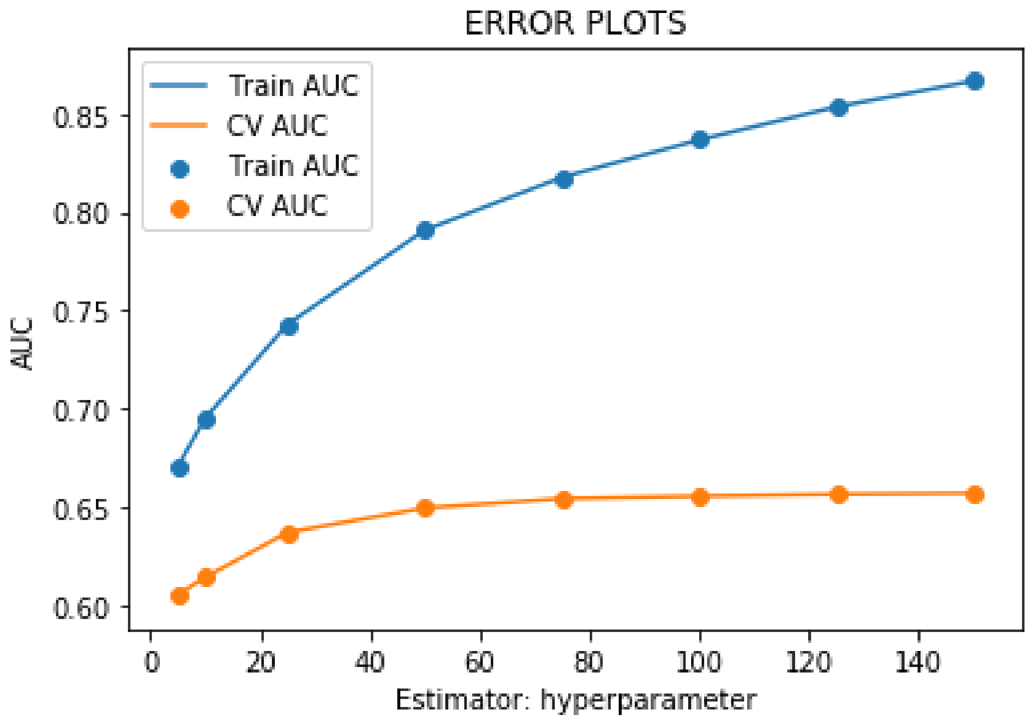
6.2.18 [5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

6.2.19 Hyperparamater tuning using GridSearch

```
In [157]: #clf = XGBClassifier()
          # For Estimator
          estimator = [5,10,25,50,75,100,125,150]
          parameters = {'n_estimators':[5,10,25,50,75,100,125,150]}
          grid = GridSearchCV(XGBClassifier(max_depth = 5), parameters, cv=3, scoring='roc_auc
          grid.fit(tfidf_sent_vectors_train1, Y_train)

          print("best estimator = ", grid.best_params_)

          train_auc_tfw2v = grid.cv_results_['mean_train_score']
          cv_auc_tfw2v = grid.cv_results_['mean_test_score']

          plt.plot(estimator, train_auc_tfw2v, label='Train AUC')
          plt.scatter(estimator, train_auc_tfw2v, label='Train AUC')
          plt.plot(estimator, cv_auc_tfw2v, label='CV AUC')
          plt.scatter(estimator, cv_auc_tfw2v, label='CV AUC')
```

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

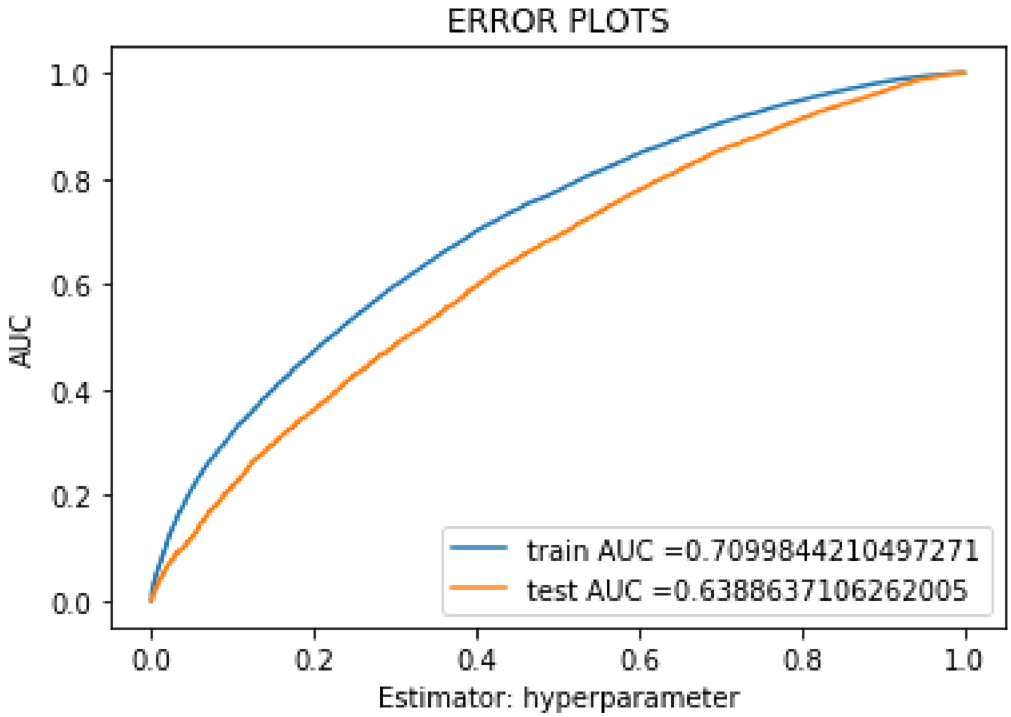
```
best estimator =  {'n_estimators': 150}
```



6.2.20 Testing with Test data

In [106]:
```
clf = XGBClassifier(n_estimators = 150, class_weight ='balanced')
clf.fit(tfidf_sent_vectors_train1, Y_train)

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

train_fpr_tfw2v, train_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_train, clf.predict_
test_fpr_tfw2v, test_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_test, clf.predict_pro

plt.plot(train_fpr_tfw2v, train_tpr_tfw2v, label="train AUC ="+str(auc(train_fpr_tfw
plt.plot(test_fpr_tfw2v, test_tpr_tfw2v, label="test AUC ="+str(auc(test_fpr_tfw2v, 
plt.legend()
```

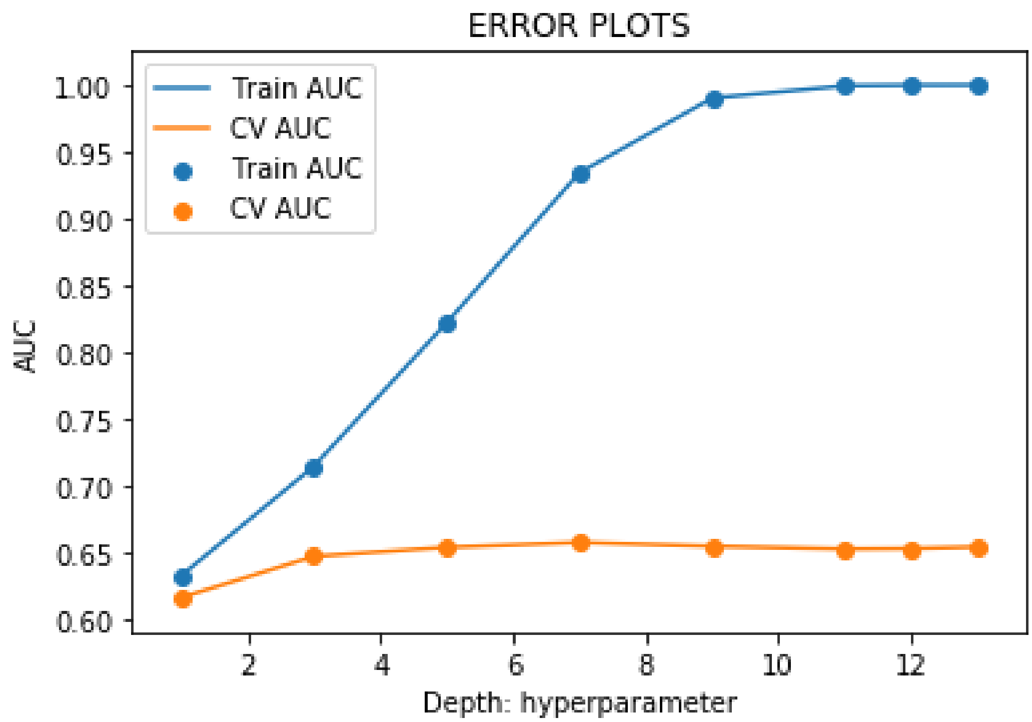
```
plt.xlabel("Estimator: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```




```
In [158]: #clf = XGBClassifier()
          # For Depth
          depth = [1,3,5,7,9,11,12,13]
          parameters = {'max_depth':[1,3,5,7,9,11,12,13]}
          grid = GridSearchCV(XGBClassifier(booster='gbtree',n_estimators = 80), parameters, c
          grid.fit(tfidf_sent_vectors_train1, Y_train)

          print("best depth = ", grid.best_params_)

          train_auc_tfw2v = grid.cv_results_['mean_train_score']
          cv_auc_tfw2v = grid.cv_results_['mean_test_score']

          plt.plot(depth, train_auc_tfw2v, label='Train AUC')
          plt.scatter(depth, train_auc_tfw2v, label='Train AUC')
          plt.plot(depth, cv_auc_tfw2v, label='CV AUC')
          plt.scatter(depth, cv_auc_tfw2v, label='CV AUC')

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

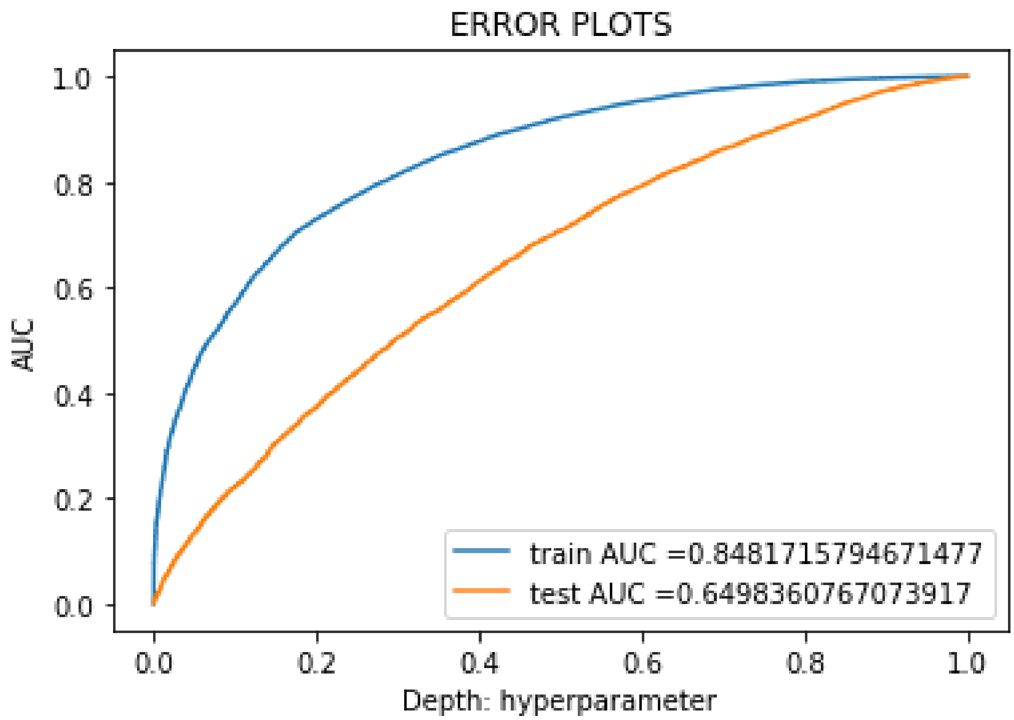
```
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.show()
```

```
best depth =  {'max_depth': 7}
```



6.2.21 Testing with Test Data

```
In [108]: clf = XGBClassifier(max_depth = 6, class_weight ='balanced')
          clf.fit(tfidf_sent_vectors_train1, Y_train)

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

          train_fpr_tfw2v, train_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_train, clf.predict_
          test_fpr_tfw2v, test_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_test, clf.predict_prol

          plt.plot(train_fpr_tfw2v, train_tpr_tfw2v, label="train AUC ="+str(auc(train_fpr_tfw2
          plt.plot(test_fpr_tfw2v, test_tpr_tfw2v, label="test AUC ="+str(auc(test_fpr_tfw2v, t
          plt.legend()
          plt.xlabel("Depth: hyperparameter")
          plt.ylabel("AUC")
          plt.title("ERROR PLOTS")
          plt.show()
```



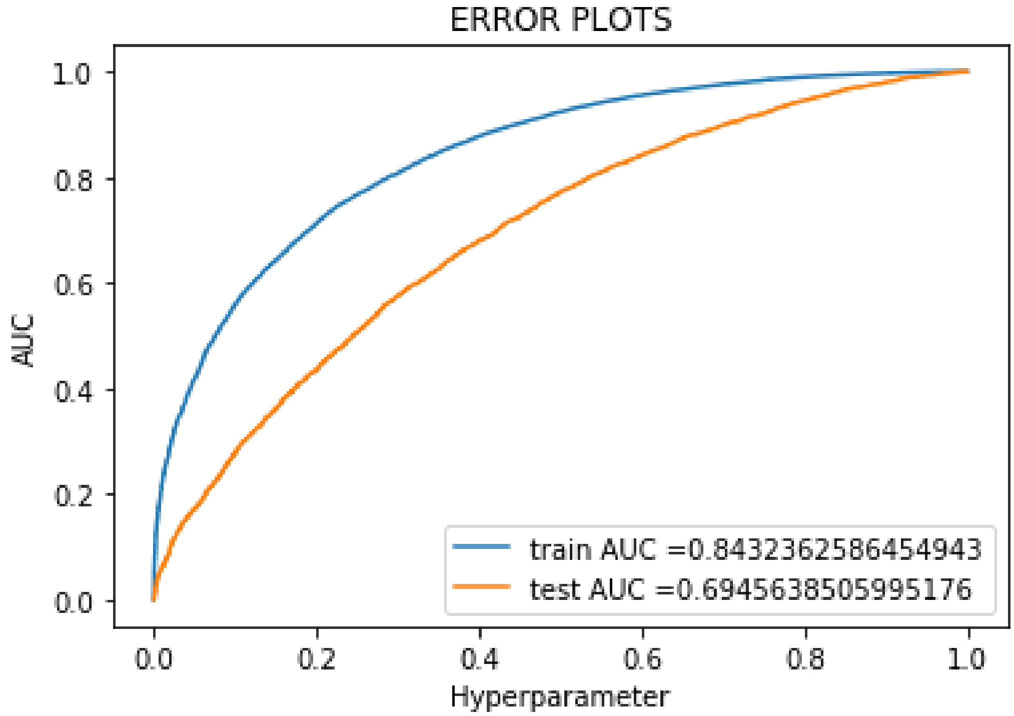

```
In [62]: # clf = XGBClassifier()
         estimator = [5,10,25,50,75,100,125,150]
         depth = [1,3,5,7,9,11,12,13]

         parameters = {'n_estimators': estimator,'max_depth': depth}
         grid = GridSearchCV(XGBClassifier(max_features='sqrt'), parameters, cv=3, scoring='ro
         grid.fit(tfidf_sent_vectors_train1, Y_train)

Out[62]: GridSearchCV(cv=3, error_score='raise',
                estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
                max_delta_step=0, max_depth=3, max_features='sqrt',
                min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
                nthread=None, objective='binary:logistic', random_state=0,
                reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                silent=None, subsample=1, verbosity=1),
                fit_params=None, iid=True, n_jobs=-1,
                param_grid={'n_estimators': [5, 10, 25, 50, 75, 100, 125, 150], 'max_depth': [
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring='roc_auc', verbose=0)

In [63]: optimal_estimator = grid.best_estimator_.n_estimators
         print("Value for optimal estimator : ",optimal_estimator)
```

```
optimal_depth = grid.best_estimator_.max_depth
print("Value for optimal depth : ",optimal_depth)
```

```
Value for optimal estimator :  150
Value for optimal depth :  5
```

6.2.22 Testing with Test data

```
In [64]: clf = XGBClassifier(max_depth = optimal_depth, n_estimators= optimal_estimator, class_
         clf.fit(tfidf_sent_vectors_train1, Y_train)

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

         train_fpr_tfw2v, train_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_train, clf.predict_pr
         test_fpr_tfw2v, test_tpr_tfw2v, thresholds_tfw2v = roc_curve(Y_test, clf.predict_proba

         plt.plot(train_fpr_tfw2v, train_tpr_tfw2v, label="train AUC ="+str(auc(train_fpr_tfw2v
         plt.plot(test_fpr_tfw2v, test_tpr_tfw2v, label="test AUC ="+str(auc(test_fpr_tfw2v, te
         plt.legend()
         plt.xlabel("Hyperparameter")
         plt.ylabel("AUC")
         plt.title("ERROR PLOTS")
         plt.show()
```

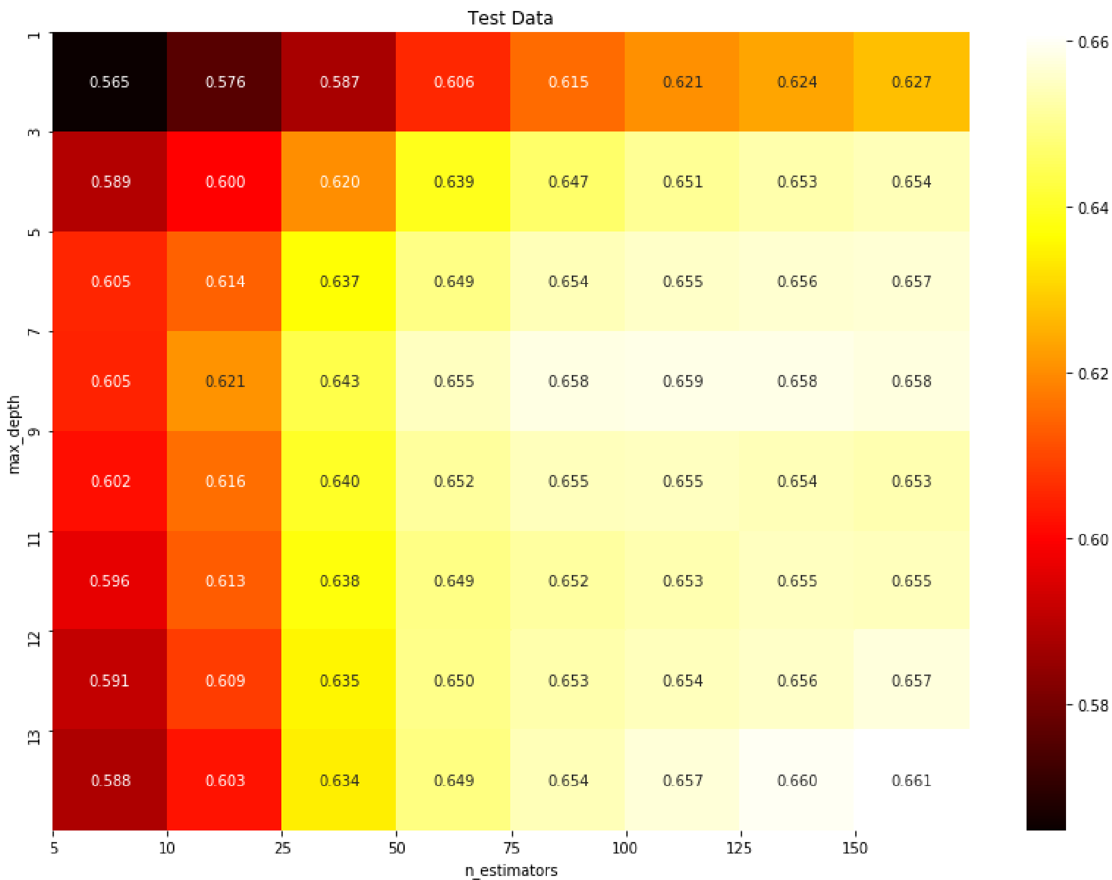


```
In [163]: clf = XGBClassifier(n_estimators = optimal_estimator, max_depth = optimal_depth)
          clf.fit(tfidf_sent_vectors_train1, Y_train)
          predw1 = clf.predict(tfidf_sent_vectors_test1)

          accw1 = accuracy_score(Y_test, predw1) * 100
          prew1 = precision_score(Y_test, predw1) * 100
          recw1 = recall_score(Y_test, predw1) * 100
          f1w1 = f1_score(Y_test, predw1) * 100

          print('\nAccuracy=%f%%' % (accw1))
          print('\nprecision=%f%%' % (prew1))
          print('\nrecall=%f%%' % (recw1))
          print('\nF1-Score=%f%%' % (f1w1))
```

```
Accuracy=83.776394%

precision=83.928913%

recall=99.695828%

F1-Score=91.135458%
```

6.2.23 Heatmap for Test Data

```
In [164]: scores = grid.cv_results_['mean_test_score'].reshape(len(estimator),len(depth))
          plt.figure(figsize=(14,10))
          sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimator, y
          plt.xlabel('n_estimators')
          plt.ylabel('max_depth')
          plt.xticks(np.arange(len(estimator)), estimator)
          plt.yticks(np.arange(len(depth)), depth)
          plt.title('Test Data')
          plt.show()
```



7 [6] Conclusions

In [165]:
```
number= [1,2,3,4,5,6,7,8]
name= ["Bow", "Tfidf", "Avg W2v", "Tfidf W2v", "Bow" ,"Tfidf","Avg W2v", "Tfidf W2v"]
svm= ["Random Forest","Random Forest","Random Forest","Random Forest", "XGBoost","XGI
acc= [accb1,acct,acca,accw,accb,acct1,acca1,accw1]
pre= [preb1,pret,prea,prew,preb,pret1,prea1,prew1]
rec= [recb1,rect,reca,recw,recb,rect1,reca1,recw1]
f1= [f1b1,f1t,f1a,f1w,f1b,f1t1,f1a1,f1w1]


#Initialize Prettytable
ptable = PrettyTable()
ptable.add_column("Index", number)
ptable.add_column("Model", name)
ptable.add_column("Ensemble Model", svm)
ptable.add_column("Accuracy%", acc)
ptable.add_column("Precision%", pre)
ptable.add_column("Recall%", rec)
```

```python
ptable.add_column("F1%", f1)

print(ptable)
```

Index	Model	Ensemble Model	Accuracy%	Precision%	Recall%
1	Bow	Random Forest	87.11833510557496	86.86157376270918	99.677668316157
2	Tfidf	Random Forest	87.65000759532128	87.38103770955283	99.623189721705
3	Avg W2v	Random Forest	83.65866626158287	83.65742499050512	100.0
4	Tfidf W2v	Random Forest	83.74221479568585	83.76331811263317	99.93644163980
5	Bow	XGBoost	90.3615372930275	91.04848561438982	98.125028374267
6	Tfidf	XGBoost	90.236214491873	90.90832632464256	98.143187905752
7	Avg W2v	XGBoost	83.92830016709706	84.33869785033383	99.210060380442
8	Tfidf W2v	XGBoost	83.77639374145527	83.92891266959678	99.695827847641

1. Both Bow and Tfidf have more accuracy than Avg W2v and Tfidf W2v in RF and XGBoost
2. Bow and Tfidf models of XGBoost have high accuracy.
3. Both parameters n_estimators and max_depth are used for hyperparameter tuning at same time.
4. Heatmap for Train and Test data are plotted for each model using n-estimator and max_depth.
5. For XGBoost we have taken smaller value of max_depth parameter.
6. On applying Random Forest on Avg-W2v and Tfidf-W2v, on plotting ROC curve we do not get good results.