09 Amazon Fine Food Reviews Analysis_GDBT

June 21, 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

# importing Cross validation libs
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import model_selection


# Python script for confusion matrix creation.
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

```python
from sklearn.metrics import classification_report
# ROC , AUC curve
# roc curve and auc
from sklearn.datasets import make_classification
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
from sklearn.metrics import roc_curve, auc
# kFold
from sklearn.model_selection import KFold

from sklearn.model_selection import GridSearchCV

import seaborn as sns

from sklearn.model_selection import TimeSeriesSplit
from sklearn.model_selection import RandomizedSearchCV

from joblib import dump, load
from sklearn_pandas import DataFrameMapper
from sklearn.metrics import f1_score,recall_score,precision_score
import xgboost as xgb
from xgboost.sklearn import XGBClassifier

from wordcloud import WordCloud
```

In [2]:
```python
# using SQLite Table to read data.
con = sqlite3.connect(r'/home/pranay/ML datasource/amazon-fine-food-reviews/database.s

# 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 850

# 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 (85000, 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  B005ZBZLT4                 Breyton  1331510400      2
        1  #oc-R11D9D7SHXIJB9  B005HG9ESG  Louis E. Emory "hoppy"  1342396800      5
        2  #oc-R11DNU2NBKQ23Z  B005ZBZLT4        Kim Cieszykowski  1348531200      1
        3  #oc-R11O5J5ZVQE25C  B005HG9ESG            Penguin Chick  1346889600      5
        4  #oc-R12KPBODL2B5ZD  B007OSBEV0   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  B001ATMQK2  undertheshrine "undertheshrine"  1296691200

               Score                                               Text  COUNT(*)
        80638      5  I bought this 6 pack because for the price tha...         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]: (75844, 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]: 89.22823529411764
```

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

In [13]:
```python
#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()
```

```
(75842, 10)
```

Out[13]:
```
1    63459
0    12383
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
==================================================
These were just adorable when used on a bee hive cake!  In the right setting they add that spe
==================================================
This is the BEST!<br />Years ago, I used it & loved it! Moved & could not find it again. I hav
==================================================
This product arrived in a timely manner, in good condition, and it was a hit with the family w
==================================================
```

```
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
==================================================
These were just adorable when used on a bee hive cake!  In the right setting they add that spe
==================================================
This is the BEST!Years ago, I used it & loved it! Moved & could not find it again. I have purc
==================================================
This product arrived in a timely manner, in good condition, and it was a hit with the family w
```

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

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

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

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

```
This is the BEST!<br />Years ago, I used it & loved it! Moved & could not find it again. I have
==================================================
```

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)
```

```
This is the BEST br Years ago I used it loved it Moved could not find it again I have purchase
```

```
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', '
                     'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'a
                     'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'throug
                     'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', '
                     'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'a
                     'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'to
                     's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", '
                     've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't
                     "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
                     "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
                     'won', "won't", 'wouldn', "wouldn't"])
```

```
In [22]: #filtered out whole reviews

         from bs4 import BeautifulSoup
         # Combining all the above stundents
         from tqdm import tqdm
         # tqdm is for printing the status bar
         word_counter = []
         def filterised_text(text):
             preprocessed_text = []
             for sentance in tqdm(text):
                 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 s
```

```
            count = len(sentance.split())
            word_counter.append(count)
            preprocessed_text.append(sentance.strip())
        return preprocessed_text
```

```
In [23]: preprocessed_reviews = filterised_text(final['Text'].values)
         final['preprocessed_reviews']  = preprocessed_reviews
         preprocessed_reviews[1822]
```

```
100%|| 75842/75842 [00:25<00:00, 2930.99it/s]
```

```
Out[23]: 'fell love product england boyfriend sandwich branston never want eat sandwich withou
```

```
In [24]: final['numbers_of_words']  = word_counter
         word_counter[1822]
```

```
Out[24]: 34
```

4.1.1 [3.2] Preprocessing Review Summary

```
In [25]: preprocessed_summary = filterised_text(final['Summary'].values)
         final['preprocessed_summary']  = preprocessed_summary
         preprocessed_summary[1822]
```

```
100%|| 75842/75842 [00:14<00:00, 5368.92it/s]
```

```
Out[25]: 'branstolicious'
```

```
In [26]: avg_w2v_trained_model_100000 = '/home/pranay/ML trained models/W2V/avg_w2v_trained_mo
         avg_w2v_test_model_100000 = '/home/pranay/ML trained models/W2V/avg_w2v_test_model_10

         w2v_tf_idf_trained_model_100000 = '/home/pranay/ML trained models/W2V_TFIDF/w2v_tf_id
         w2v_tf_idf_test_model_100000 = '/home/pranay/ML trained models/W2V_TFIDF/w2v_tf_idf_t
```

```
In [37]: depth_ = [2,5,7,10,25]
         depth_ = np.asarray(depth_)

         estimators = [25,50,100,500,1000]
         estimators_list = np.asarray(estimators)


         def finding_best_hyperparam(X_tr,y_tr):
             # instantiate a GBDT model
             xgb = XGBClassifier(class_weight='balanced', random_state=1)

             param_grid=dict(n_estimators=estimators_list,max_depth=depth_)
```

```python
    #For time based splitting
    tscv = TimeSeriesSplit(n_splits=10)

    # instantiate the training grid search model
    train_grid = GridSearchCV(xgb, param_grid, cv=tscv, scoring='roc_auc',n_jobs =-1,v

    # fit the training data to train model
    train_grid.fit(X_tr, y_tr)

    return train_grid

# https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-class

# plot AUC curve
def plotAUC_ROC(model,X_train, y_train, X_test, y_test):
    # predict probabilities
    test_probs = model.predict_proba(X_test)
    train_probs = model.predict_proba(X_train)

    # keep probabilities for the positive outcome only
    test_probs = test_probs[:, 1]
    train_probs = train_probs[:, 1]

    # calculate AUC
    test_auc = roc_auc_score(y_test, test_probs)
    train_auc = roc_auc_score(y_train, train_probs)

    # calculate roc curve
    train_fpr, train_tpr, thresholds = roc_curve(y_train, train_probs)
    test_fpr, test_tpr, thresholds2 = roc_curve(y_test, test_probs)

    # plot no skill
    pyplot.plot([0, 1], [0, 1], linestyle='--')

    # plot the roc curve for the model
    pyplot.plot(train_fpr, train_tpr, 'r',marker='.', label="train AUC ="+str(train_au
    pyplot.plot(test_fpr, test_tpr, 'b',marker='.',label="test AUC ="+str(test_auc))

    pyplot.legend()
    pyplot.xlabel("K: hyperparameter")
    pyplot.ylabel("AUC")
    pyplot.title("ERROR PLOTS")
    # show the plot
    pyplot.show()

    return train_auc, test_auc
```

```
# https://www.geeksforgeeks.org/confusion-matrix-machine-learning/
def plotConfusionMatrix(y_test,pred):
    # calculate confusion matrix
    cm =  confusion_matrix(y_test,pred)
    class_label = ['negative', 'positive']
    df_conf_matrix = pd.DataFrame(cm, index=class_label, columns=class_label)
    # heatmap --> Plot rectangular data as a color-encoded matrix.
    sns.heatmap(df_conf_matrix, annot=True, fmt='d')
    # give title to graph
    plt.title("Confusion Matrix")
    # mention axis label
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    # show the plot
    plt.show()

class color:
   PURPLE = '\033[95m'
   CYAN = '\033[96m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
   GREEN = '\033[92m'
   YELLOW = '\033[93m'
   RED = '\033[91m'
   BOLD = '\033[1m'
   UNDERLINE = '\033[4m'
   END = '\033[0m'

# https://qiita.com/bmj0114/items/8009f282c99b77780563
def plotHeatMap(trained_model, param):

    if param == 'trained':
        scores = trained_model.cv_results_['mean_train_score'].reshape(len(estimators_
    else:
        scores = trained_model.cv_results_['mean_test_score'].reshape(len(estimators_l

    plt.figure(figsize=(16, 12))
    sns.heatmap(scores, annot=True, cmap=plt.cm.hot, fmt=".3f", xticklabels=estimators
    plt.xlabel('n_estimators')
    plt.ylabel('max_depth')
    plt.xticks(np.arange(len(estimators_list)), estimators_list)
    plt.yticks(np.arange(len(depth_)), depth_)
    plt.title('Grid Search AUC Score')
    plt.show()
```

5 [4] Featurization

5.0.1 Splitting data

We have considered 85 k points

```
In [78]: X = final['preprocessed_reviews']
         y = final['Score']

         # split the data set into train and test
         X_train, x_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0

         print(X_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(53089,) (22753,) (53089,) (22753,)
```

5.1 [4.1] BAG OF WORDS

```
In [79]: ##BoW

         count_vect = CountVectorizer(ngram_range=(1,2), min_df=10) #in scikit-learn

         # train data
         X_train_bow = count_vect.fit_transform(X_train)

         # test data
         x_test_bow = count_vect.transform(x_test)


         print('X_train_bow',X_train_bow.shape)
         print('==='*10)
         print('x_test_bow',x_test_bow.shape)
```

```
X_train_bow (53089, 31683)
==============================
x_test_bow (22753, 31683)
```

5.1.1 Hyper param Tuning using GridSearch

finding 'max depth' & 'esimate models' which have maximum AUC Score

```
In [31]: bow_train_path = '/home/pranay/ML Hyperparam Tune/GBDT/bow_train_hyperparam_tuned'
         exists = os.path.isfile(bow_train_path)

         if exists:
             print("yes exists")
             bow_train = load(bow_train_path)
         else:
```

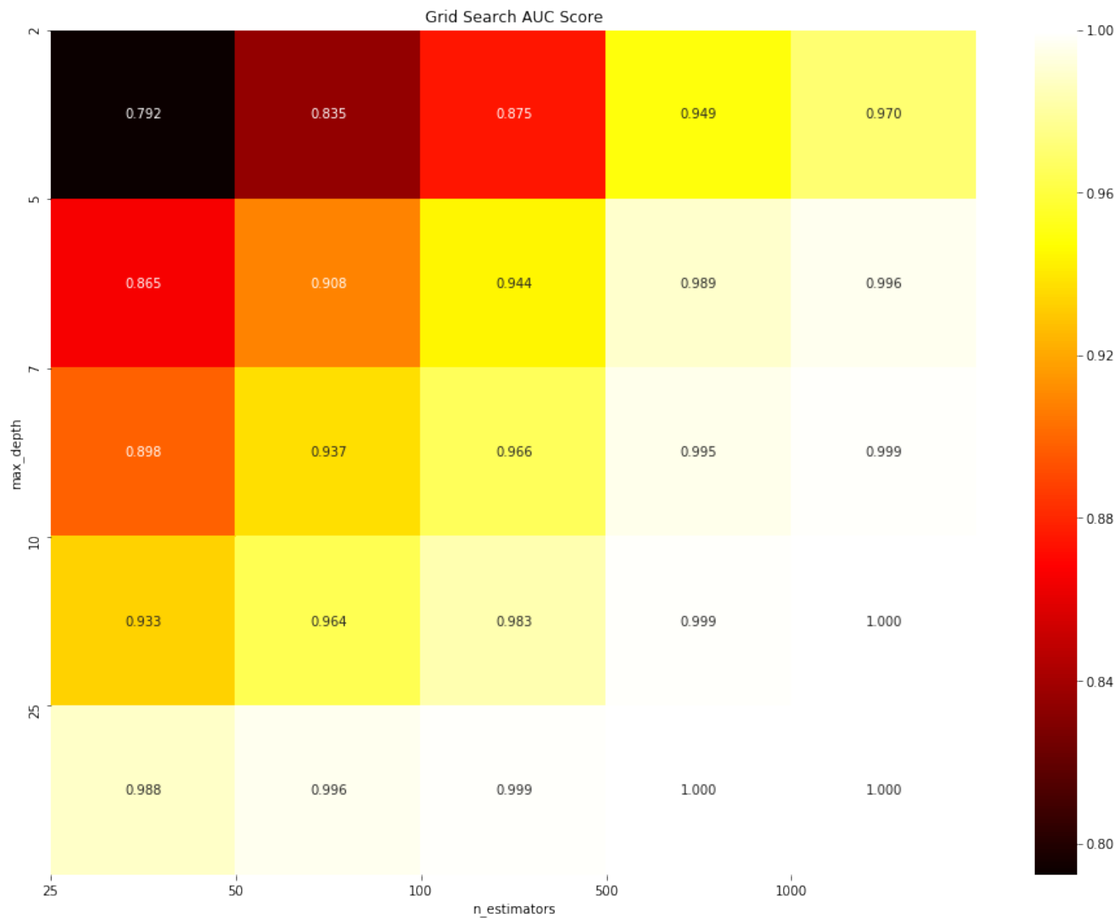
```
            print("not exists")
            bow_train = finding_best_hyperparam(X_train_bow,y_train)
            dump(bow_train,bow_train_path )


        # view the complete results (list of named tuples)
        print("=======Training========")
        print (bow_train.best_score_)
        print (bow_train.best_params_)
        print (bow_train.best_estimator_)

        # plotAccuracyGraph(bow_train,'n_estimators')
        best_estimators = bow_train.best_params_.get("n_estimators","")
        best_depth_size = bow_train.best_params_.get("max_depth", "")

        best_estimators, best_depth_size
```

```
yes exists
=======Training========
0.9377287074476324
{'max_depth': 5, 'n_estimators': 1000}
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=5, min_child_weight=1, missing=nan,
       n_estimators=1000, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=1, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)
```

```
Out[31]: (1000, 5)
```

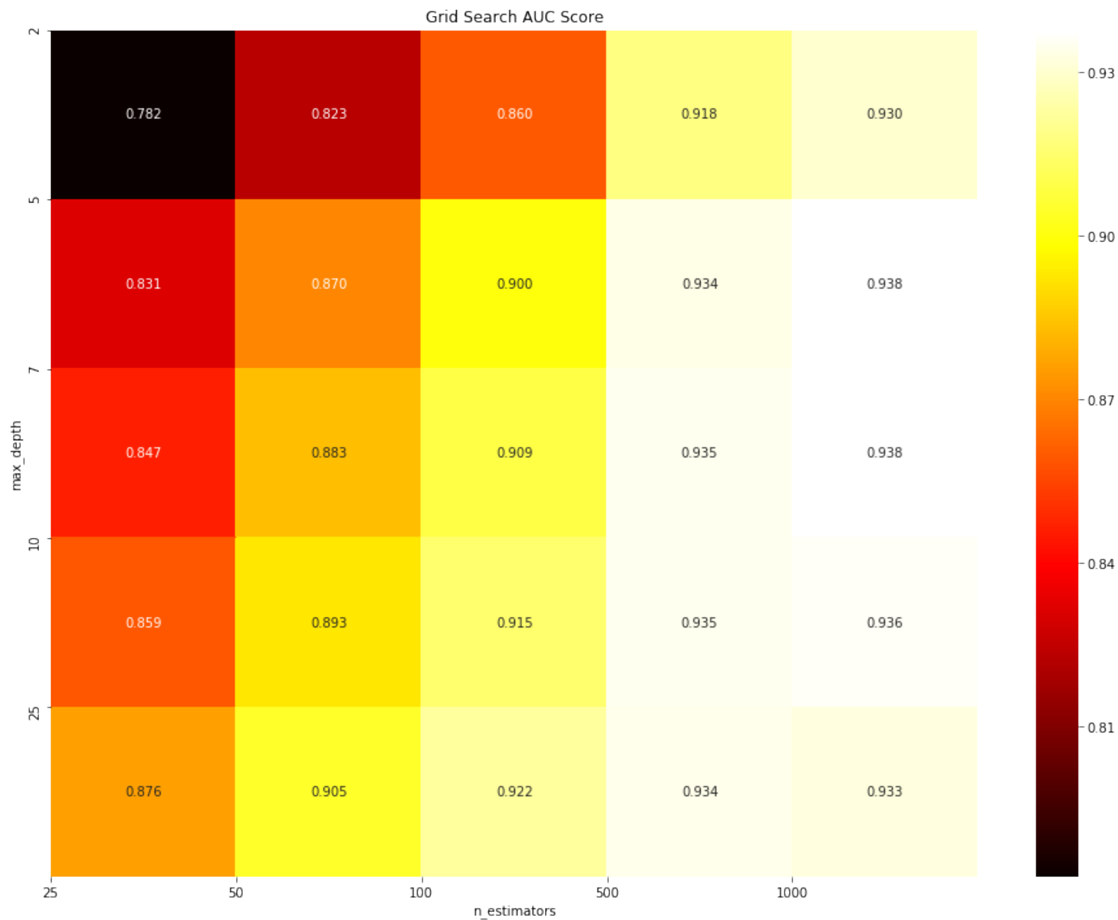
```
In [32]: print('\n'+color.BOLD +'AUC Train data'+color.END)
         plotHeatMap(bow_train,'trained')
```

AUC Train data




```
In [33]: print('\n'+color.BOLD +'AUC Validation data'+color.END)
         plotHeatMap(bow_train,'test')
```

AUC Validation data

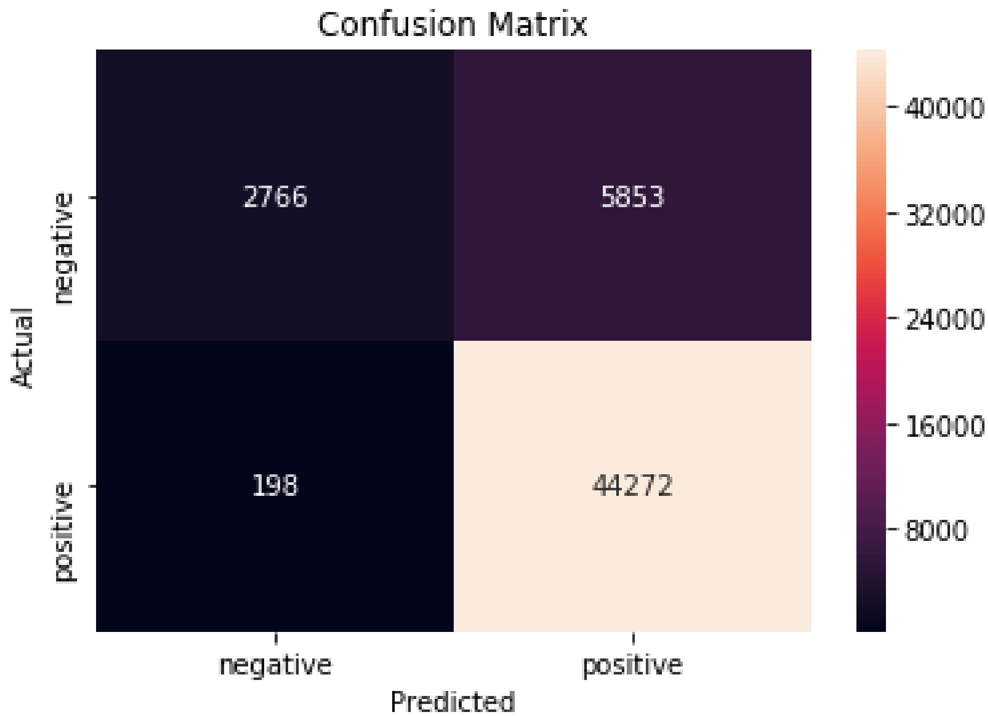


5.1.2 Applying GBDT on BOW

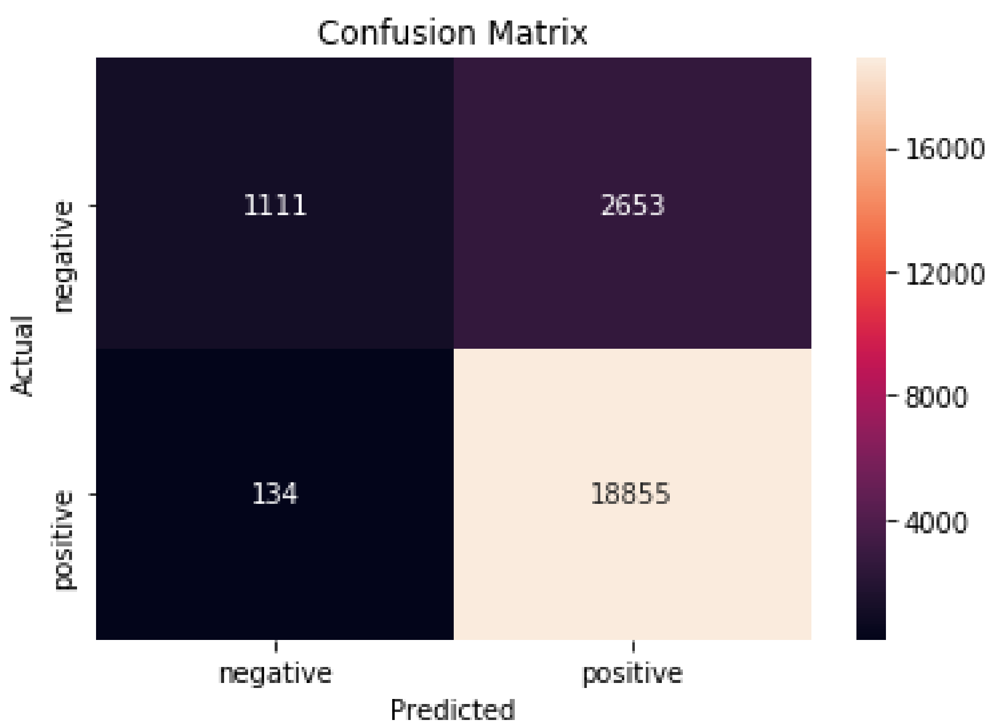
In [81]:
```
optimal_model = XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=4, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=1, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)


# fitting the model
optimal_model.fit(X_train_bow, y_train)

# predict the response
test_pred = optimal_model.predict(x_test_bow)
train_pred = optimal_model.predict(X_train_bow)
```

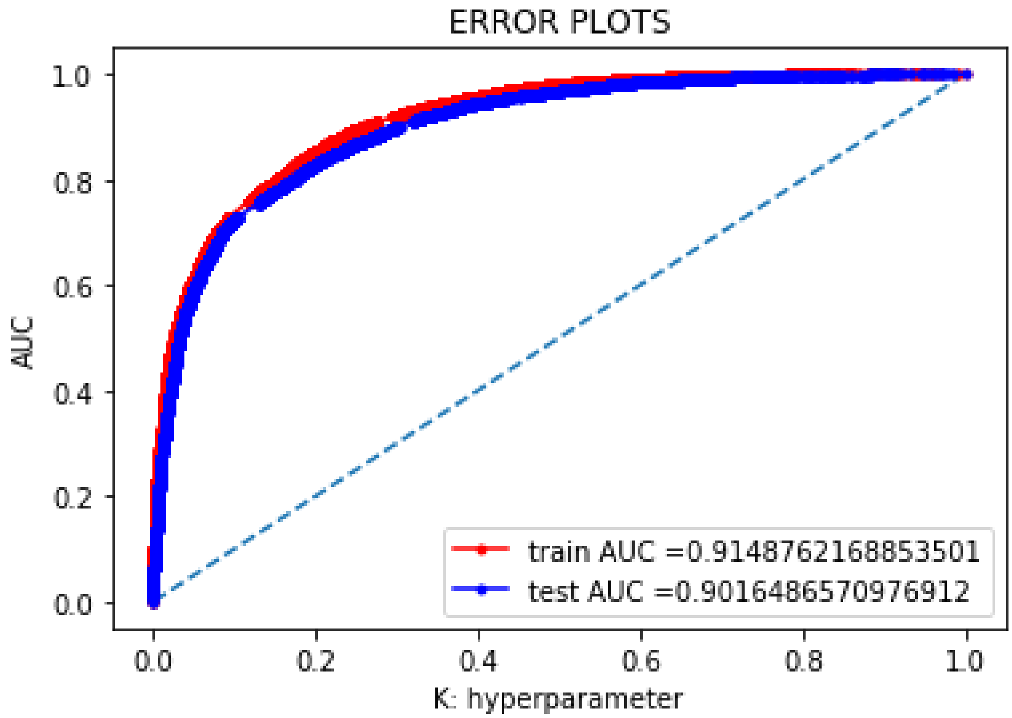
```
print('\n'+color.RED+'Max Depth : '+color.END+color.BOLD+str(4)+color.END)
print('\n'+color.RED+'Best Estimator : '+color.END+color.BOLD+str(100)+color.END)


# plot confusion matrix

print('\n'+color.BOLD +'Confusion Matrix for Train data'+color.END)
plotConfusionMatrix(y_train,train_pred)

print('\n'+color.BOLD +'Confusion Matrix for Test data'+color.END)
plotConfusionMatrix(y_test,test_pred)

# plot AUC
train_auc,test_auc = plotAUC_ROC(optimal_model,X_train_bow, y_train,x_test_bow, y_tes
print('\n'+color.RED+'AUC (Train): '+color.END+color.BOLD+str(train_auc)+color.END)
print('\n'+color.RED+'AUC (Test): '+color.END+color.BOLD+str(test_auc)+color.END)

# f1 score
score = f1_score(y_test,test_pred)
print('\n'+color.RED+'F1 SCORE (Train) : '+color.END+color.BOLD+str(f1_score(y_train,
print('\n'+color.RED+'F1 SCORE (Test) : '+color.END+color.BOLD+str(score)+color.END)

# recall
recall = metrics.recall_score(y_test, test_pred)
print('\n'+color.RED+'RECALL (Train): '+color.END+color.BOLD+str(metrics.recall_score
print('\n'+color.RED+'RECALL (Test): '+color.END+color.BOLD+str(recall)+color.END)

# precision
precision = metrics.precision_score(y_test, test_pred)
print('\n'+color.RED+'PRECISION  (Train) : '+color.END+color.BOLD+str(metrics.precisio
print('\n'+color.RED+'PRECISION  (Test) : '+color.END+color.BOLD+str(precision)+color
```

```
Max Depth : 4

Best Estimator : 100

Confusion Matrix for Train data
```



Confusion Matrix for Test data






```
AUC (Train): 0.9148762168853501

AUC (Test): 0.9016486570976912

F1 SCORE (Train) : 0.9360325598604579

F1 SCORE (Test) : 0.9311800874138826

RECALL (Train): 0.9955475601529121

RECALL (Test): 0.9929432829532887

PRECISION  (Train) : 0.8832319201995013

PRECISION  (Test) : 0.8766505486330668
```

5.1.3 Top 20 features

```
In [82]: topn_class = sorted(zip(optimal_model.feature_importances_, count_vect.get_feature_nam
         top_words_ = ''
```

```
        for feature, value in topn_class:
            print(feature, value)
            top_words_ +=' ' +value
```

```
0.017997468 waste money
0.017685018 return
0.014468065 delicious
0.014412325 terrible
0.014196279 great
0.013957699 worst
0.013775373 awful
0.013386117 not disappointed
0.013293953 money
0.01315962 refund
0.013138805 disappointed
0.012694619 not
0.012359969 threw
0.012193472 not good
0.012154945 horrible
0.011763318 favorite
0.011446781 not buy
0.0113734165 nice
0.011363316 bad
0.011279832 not recommend
```

```
In [83]: #  https://www.geeksforgeeks.org/generating-word-cloud-python/

        wordcloud = WordCloud(width = 800, height = 800,
                        background_color ='white',
                        min_font_size = 10).generate(top_words_)

        # plot the WordCloud image
        plt.figure(figsize = (8, 8), facecolor = None)
        plt.imshow(wordcloud)
        plt.axis("off")
        plt.tight_layout(pad = 0)

        plt.show()
```

5.2 Feature Engineering

Till now we only consider Text review as feature, we are adding some extra feature like **review summary** and **number of words** in review and test our model improves efficiency or not.

We have considered on 30000 points due to memory issue.

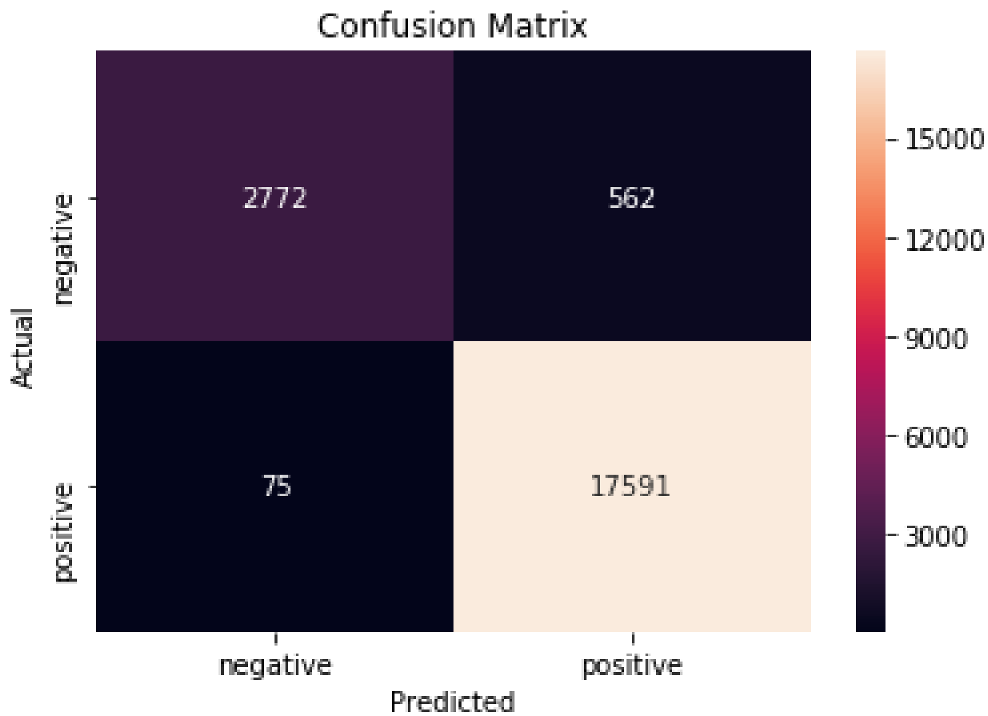
```
In [32]: # https://sondosatwi.wordpress.com/2017/08/01/using-text-data-and-dataframemapper-in-p

         X = final[:30000]
         y = final['Score'][:30000]

         # split the data set into train and test
         X_train, x_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0
```

```
print(X_train.shape, x_test.shape, y_train.shape, y_test.shape)

mapper = DataFrameMapper([
     ('preprocessed_reviews', CountVectorizer(ngram_range=(1,3), min_df=10)),
     ('preprocessed_summary', CountVectorizer(ngram_range=(1,3), min_df=10)),
     ('numbers_of_words', None),
 ])

train_features = mapper.fit_transform(X_train)
test_features = mapper.transform(x_test)

optimal_model = XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=4, min_child_weight=1,
       n_estimators=100, n_jobs=-1, nthread=None,
       objective='binary:logistic', random_state=1, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)

# fitting the model
optimal_model.fit(train_features,y_train)

# predict the response
test_pred = optimal_model.predict(test_features)
train_pred = optimal_model.predict(train_features)

# plot confusion matrix

print('\n'+color.BOLD +'Confusion Matrix for Train data'+color.END)
plotConfusionMatrix(y_train,train_pred)

print('\n'+color.BOLD +'Confusion Matrix for Test data'+color.END)
plotConfusionMatrix(y_test,test_pred)

# plot AUC
train_auc,test_auc = plotAUC_ROC(optimal_model,train_features, y_train,test_features,
print('\n'+color.RED+'AUC (Train): '+color.END+color.BOLD+str(train_auc)+color.END)
print('\n'+color.RED+'AUC (Test): '+color.END+color.BOLD+str(test_auc)+color.END)

# f1 score
score = f1_score(y_test,test_pred)
print('\n'+color.RED+'F1 SCORE (Train) : '+color.END+color.BOLD+str(f1_score(y_train,
print('\n'+color.RED+'F1 SCORE (Test) : '+color.END+color.BOLD+str(score)+color.END)

# recall
recall = metrics.recall_score(y_test, test_pred)
print('\n'+color.RED+'RECALL (Train): '+color.END+color.BOLD+str(metrics.recall_score
print('\n'+color.RED+'RECALL (Test): '+color.END+color.BOLD+str(recall)+color.END)
```

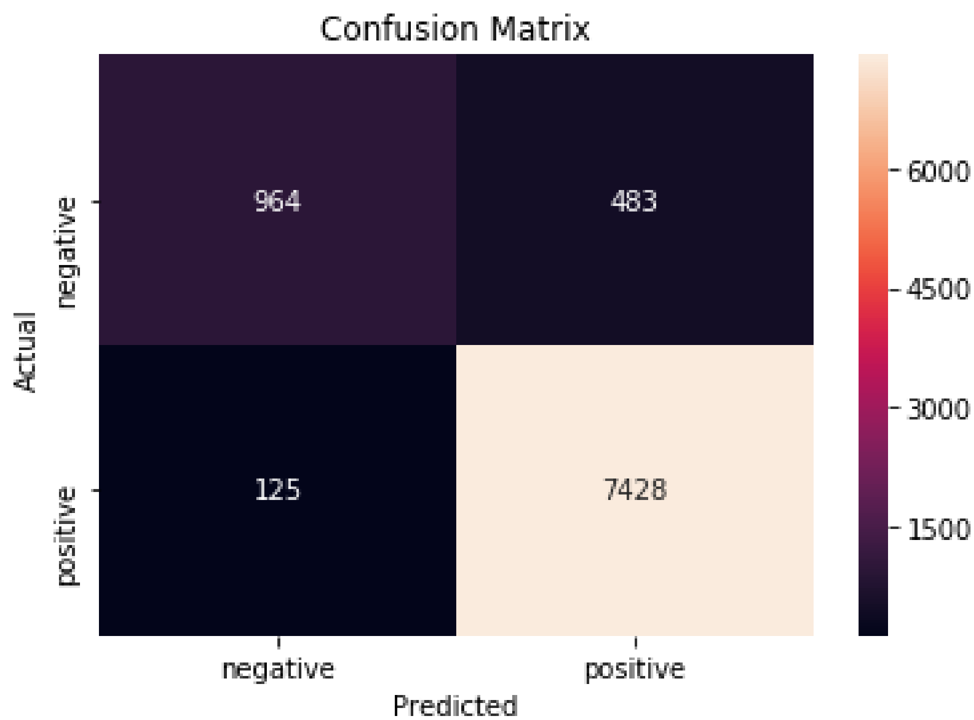
```python
# precision
precision = metrics.precision_score(y_test, test_pred)
print('\n'+color.RED+'PRECISION  (Train) : '+color.END+color.BOLD+str(metrics.precisio
print('\n'+color.RED+'PRECISION  (Test) : '+color.END+color.BOLD+str(precision)+color
```

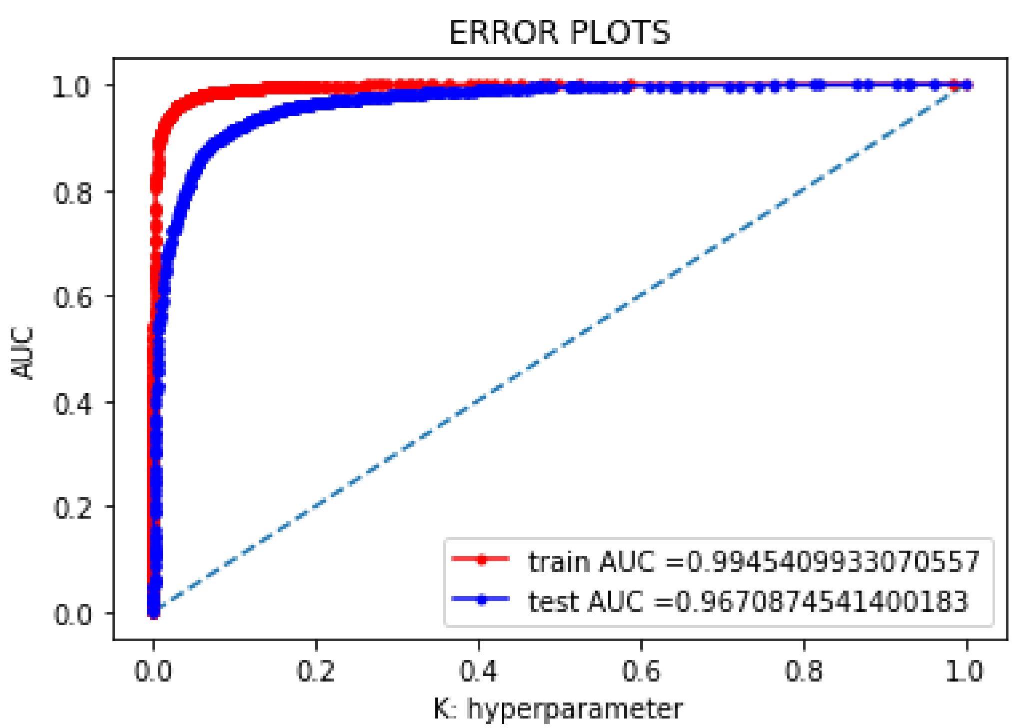
```
(21000, 13) (9000, 13) (21000,) (9000,)
```

Confusion Matrix for Train data



Confusion Matrix for Test data


Confusion Matrix
964
483
125
7428
negative
positive
Actual
negative
positive
Predicted
6000
4500
3000
1500


ERROR PLOTS
train AUC =0.9945409933070557
test AUC =0.9670874541400183
AUC
K: hyperparameter
0.0
0.2
0.4
0.6
0.8
1.0

```
AUC (Train): 0.9945409933070557

AUC (Test): 0.9670874541400183

F1 SCORE (Train) : 0.9822161422708618

F1 SCORE (Test) : 0.9606828763579928

RECALL (Train): 0.9957545567757274

RECALL (Test): 0.983450284655104

PRECISION  (Train) : 0.96904092987385

PRECISION  (Test) : 0.9389457717102768
```

5.3 [4.3] TF-IDF

```
In [41]: X = final['preprocessed_reviews']
         y = final['Score']

         # split the data set into train and test
         X_train, x_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0

         print(X_train.shape, x_test.shape, y_train.shape, y_test.shape)

         tf_idf_vect = TfidfVectorizer(ngram_range=(1,3), min_df=10) #in scikit-learn

         # train data
         X_train_tfidf = tf_idf_vect.fit_transform(X_train)

         # test data
         x_test_tfidf = tf_idf_vect.transform(x_test)

         print('X_train_tfidf', X_train_tfidf.shape)
         print('==='*10)
         print('x_test_tfidf', x_test_tfidf.shape)
```

```
(53089,) (22753,) (53089,) (22753,)
X_train_tfidf (53089, 34425)
==============================
x_test_tfidf (22753, 34425)
```

5.3.1 Hyper param Tuning using GridSearch

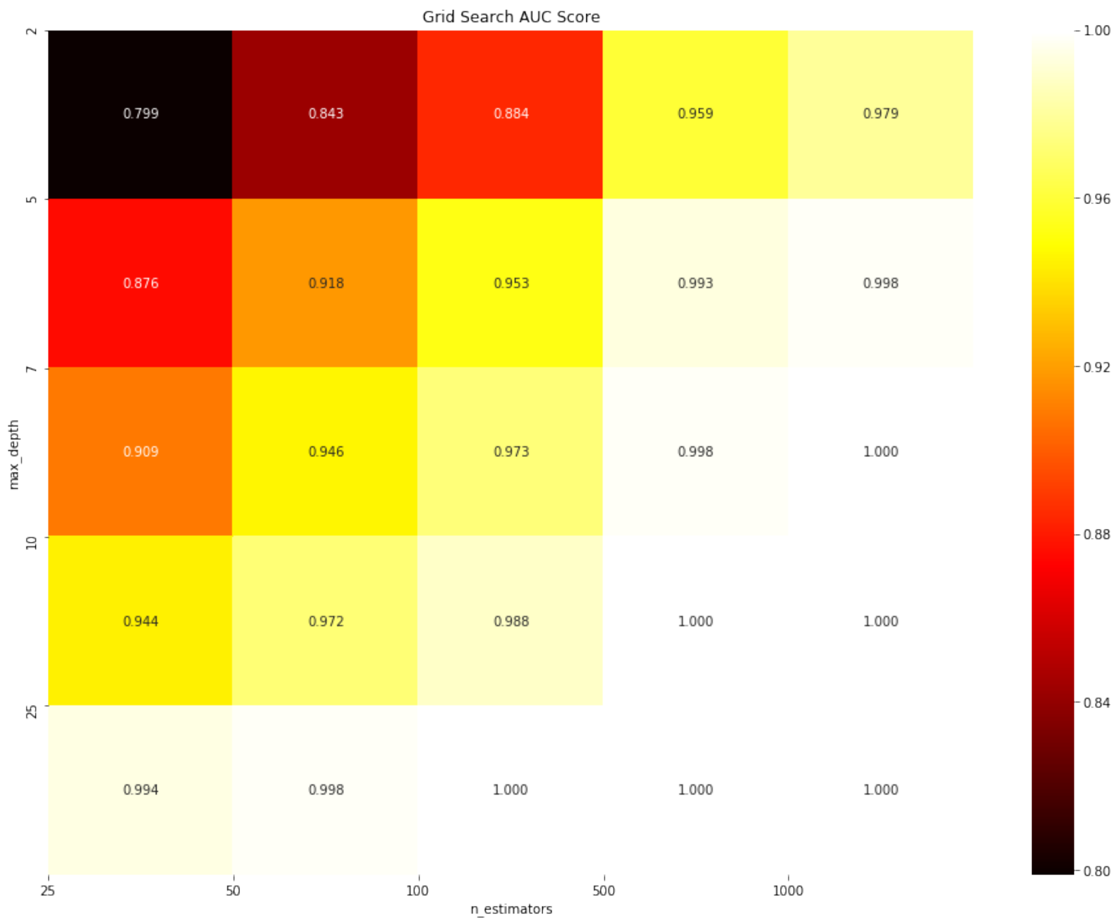
finding 'max depth' & 'esimate models' which have maximum AUC Score

```
In [42]: tfidf_train_path = '/home/pranay/ML Hyperparam Tune/GBDT/tfidf_train_hyperparam_tuned
         exists = os.path.isfile(tfidf_train_path)

         if exists:
             print("yes exists")
             tfidf_train = load(tfidf_train_path)
         else:
             print("not exists")
             tfidf_train = finding_best_hyperparam(X_train_tfidf,y_train)
             dump(tfidf_train,tfidf_train_path )


         # view the complete results (list of named tuples)
         print("=======Training========")
         print (tfidf_train.best_score_)
         print (tfidf_train.best_params_)
         print (tfidf_train.best_estimator_)

         best_depth_size = tfidf_train.best_params_.get("max_depth", "")
         best_estimators = tfidf_train.best_params_.get("n_estimators", "")
```

```
yes exists
=======Training========
0.934506456797819
{'max_depth': 5, 'n_estimators': 1000}
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=5, min_child_weight=1, missing=nan,
       n_estimators=1000, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=1, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)
```

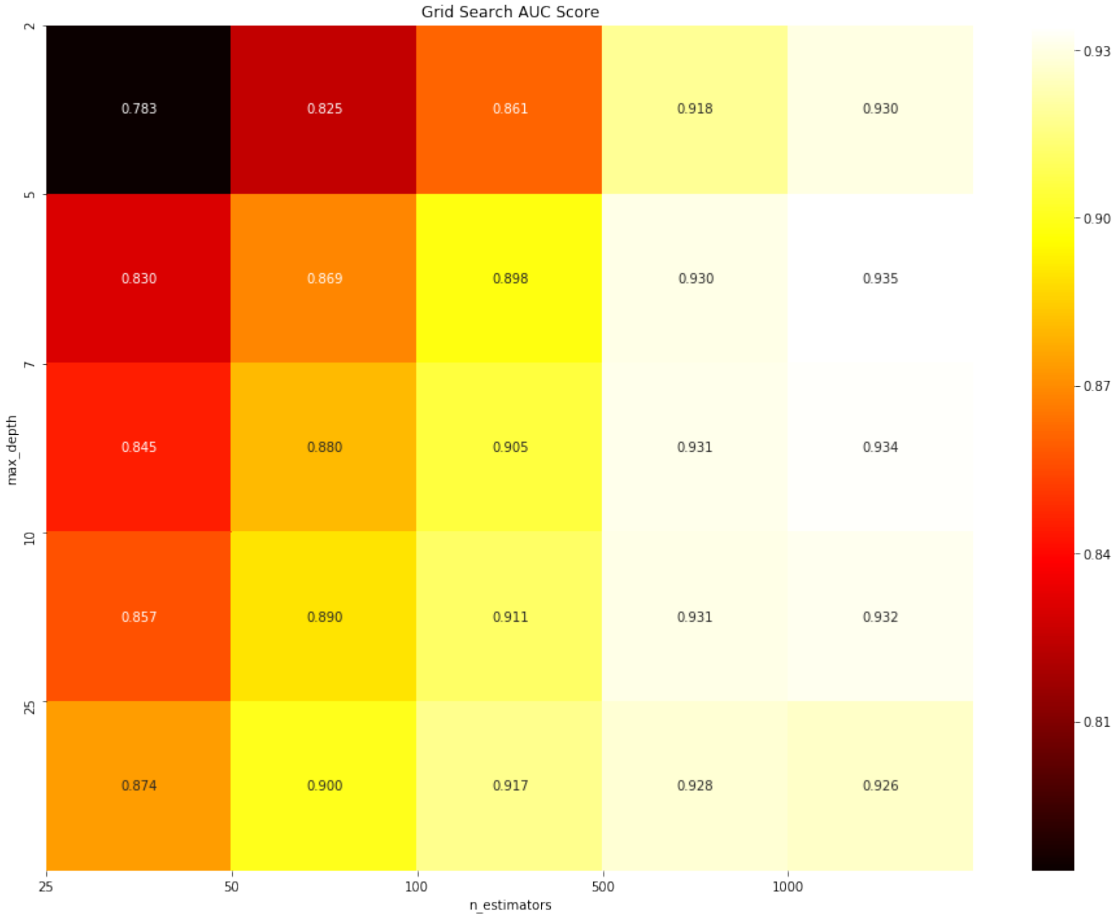
```
In [43]: print('\n'+color.BOLD +'AUC Train data'+color.END)
         plotHeatMap(tfidf_train,'trained')
```

AUC Train data




```
In [44]: print('\n'+color.BOLD +'AUC Validation data'+color.END)
         plotHeatMap(tfidf_train,'test')
```

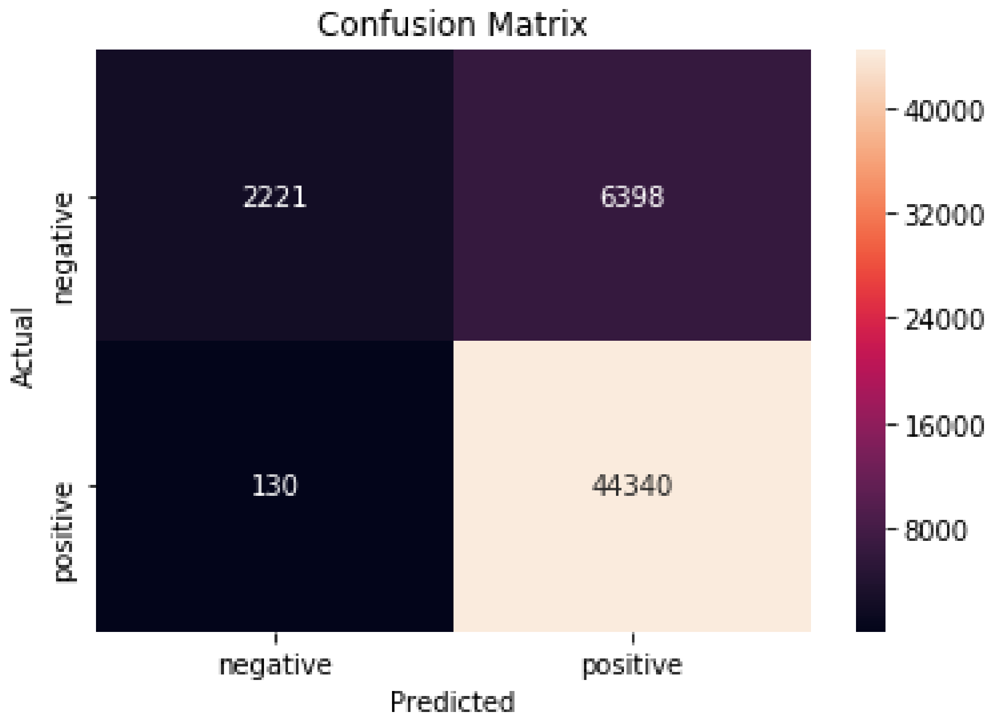
AUC Validation data



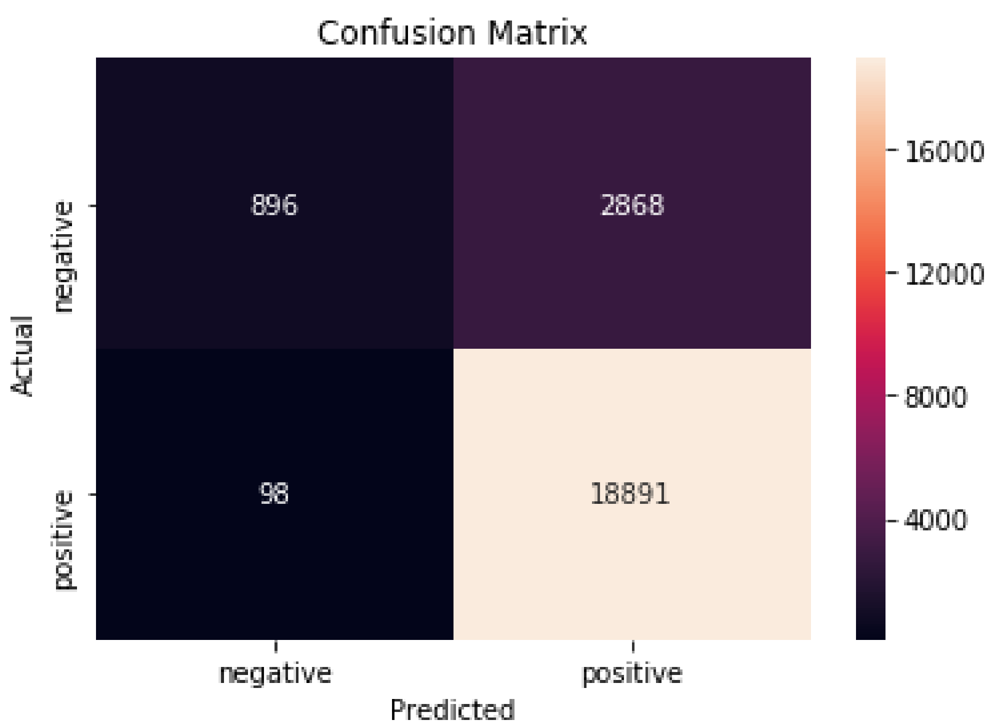
In [45]:
```
optimal_model = XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=-1, nthread=None,
       objective='binary:logistic', random_state=1, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)

# fitting the model
optimal_model.fit(X_train_tfidf, y_train)

# predict the response
test_pred = optimal_model.predict(x_test_tfidf)
train_pred = optimal_model.predict(X_train_tfidf)

print('\n'+color.RED+'Max Depth : '+color.END+color.BOLD+str(3)+color.END)
print('\n'+color.RED+'Best Estimator : '+color.END+color.BOLD+str(100)+color.END)
```

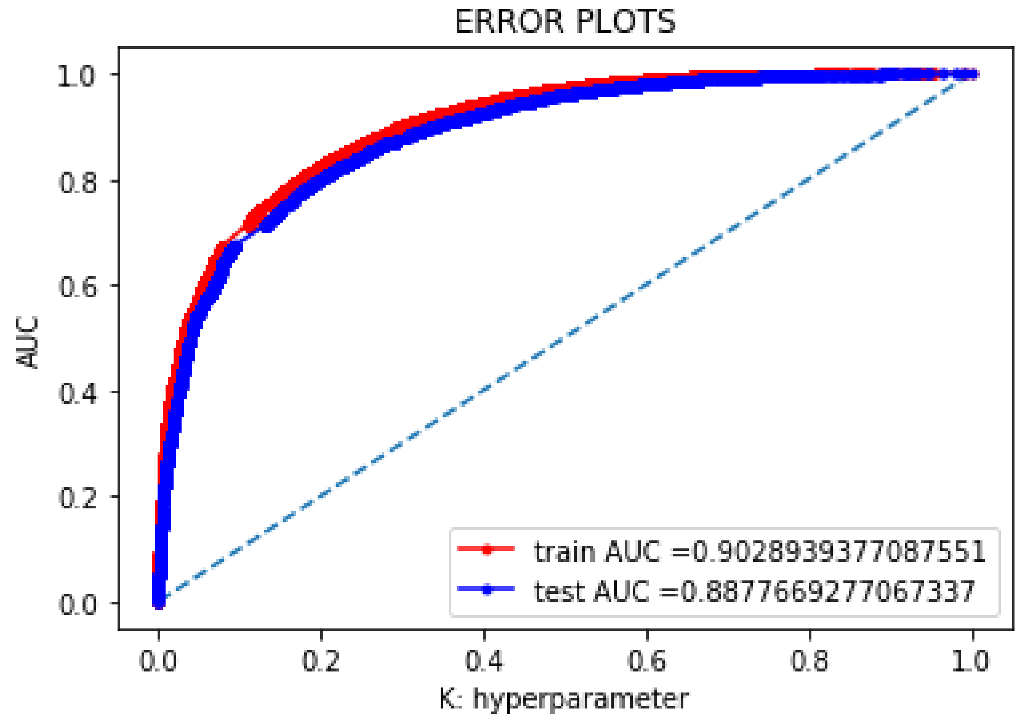
```
# plot confusion matrix

print('\n'+color.BOLD +'Confusion Matrix for Train data'+color.END)
plotConfusionMatrix(y_train,train_pred)

print('\n'+color.BOLD +'Confusion Matrix for Test data'+color.END)
plotConfusionMatrix(y_test,test_pred)

# plot AUC
train_auc,test_auc = plotAUC_ROC(optimal_model,X_train_tfidf, y_train,x_test_tfidf, y_
print('\n'+color.RED+'AUC (Train): '+color.END+color.BOLD+str(train_auc)+color.END)
print('\n'+color.RED+'AUC (Test): '+color.END+color.BOLD+str(test_auc)+color.END)

# f1 score
score = f1_score(y_test,test_pred)
print('\n'+color.RED+'F1 SCORE (Train) : '+color.END+color.BOLD+str(f1_score(y_train,
print('\n'+color.RED+'F1 SCORE (Test) : '+color.END+color.BOLD+str(score)+color.END)

# recall
recall = metrics.recall_score(y_test, test_pred)
print('\n'+color.RED+'RECALL (Train): '+color.END+color.BOLD+str(metrics.recall_score
print('\n'+color.RED+'RECALL (Test): '+color.END+color.BOLD+str(recall)+color.END)

# precision
precision = metrics.precision_score(y_test, test_pred)
print('\n'+color.RED+'PRECISION  (Train) : '+color.END+color.BOLD+str(metrics.precisi
print('\n'+color.RED+'PRECISION  (Test) : '+color.END+color.BOLD+str(precision)+color
```

```
Max Depth : 3

Best Estimator : 100

Confusion Matrix for Train data
```



Confusion Matrix for Test data






```
AUC (Train): 0.9028939377087551

AUC (Test): 0.8877669277067337

F1 SCORE (Train) : 0.9314343332493068

F1 SCORE (Test) : 0.9272111514675567

RECALL (Train): 0.9970766809084777

RECALL (Test): 0.9948391173837485

PRECISION  (Train) : 0.8739012180219954

PRECISION  (Test) : 0.8681924720805184
```

5.4 Top 20 Features

```
In [47]: topn_class = sorted(zip(optimal_model.feature_importances_, tf_idf_vect.get_feature_na
         top_words_tfidf = ''
```

```
        for feature, value in topn_class:
            print(feature, value)
            top_words_tfidf +=' ' +value
```

```
0.0234979 not
0.019388214 return
0.017660951 awful
0.017047124 horrible
0.01694386 threw
0.016529012 worst
0.015682874 money
0.015432428 delicious
0.015123611 great
0.014597757 disappointed
0.014460342 terrible
0.014275661 bad
0.01400822 not buy
0.013310471 received
0.013188662 love
0.013080963 waste money
0.012949058 best
0.012549151 perfect
0.01253835 refund
0.012405823 nice
```

```
In [48]: #  https://www.geeksforgeeks.org/generating-word-cloud-python/

         wordcloud = WordCloud(width = 800, height = 800,
                         background_color ='white',
                         min_font_size = 10).generate(top_words_tfidf)

         # plot the WordCloud image
         plt.figure(figsize = (8, 8), facecolor = None)
         plt.imshow(wordcloud)
         plt.axis("off")
         plt.tight_layout(pad = 0)

         plt.show()
```

5.4.1 Feature Engineering

Till now we only consider Text review as feature, we are adding some extra feature like **review summary** and **number of words** in review and test our model improves efficiency or not.

We have considered on 30000 points due to memory issue.

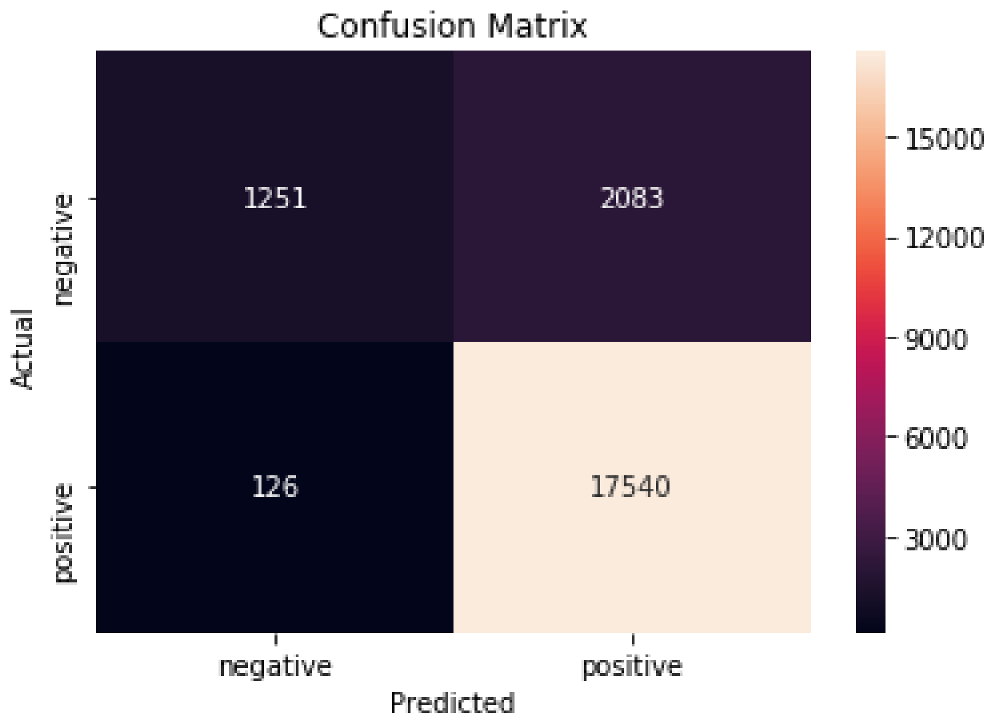
```
In [49]: # https://sondosatwi.wordpress.com/2017/08/01/using-text-data-and-dataframemapper-in-p

         X = final[:30000]
         y = final['Score'][:30000]

         # split the data set into train and test
         X_train, x_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0
```

```
print(X_train.shape, x_test.shape, y_train.shape, y_test.shape)

mapper = DataFrameMapper([
     ('preprocessed_reviews', TfidfVectorizer(ngram_range=(1,3), min_df=10)),
     ('preprocessed_summary', TfidfVectorizer(ngram_range=(1,3), min_df=10)),
     ('numbers_of_words', None),
 ])

train_features = mapper.fit_transform(X_train)
test_features = mapper.transform(x_test)

optimal_model = XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=-1, nthread=None,
       objective='binary:logistic', random_state=1, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)

# fitting the model
optimal_model.fit(train_features,y_train)

# predict the response
test_pred = optimal_model.predict(test_features)
train_pred = optimal_model.predict(train_features)

# plot confusion matrix

print('\n'+color.BOLD +'Confusion Matrix for Train data'+color.END)
plotConfusionMatrix(y_train,train_pred)

print('\n'+color.BOLD +'Confusion Matrix for Test data'+color.END)
plotConfusionMatrix(y_test,test_pred)

# plot AUC
train_auc,test_auc = plotAUC_ROC(optimal_model,train_features, y_train,test_features,
print('\n'+color.RED+'AUC (Train): '+color.END+color.BOLD+str(train_auc)+color.END)
print('\n'+color.RED+'AUC (Test): '+color.END+color.BOLD+str(test_auc)+color.END)

# f1 score
score = f1_score(y_test,test_pred)
print('\n'+color.RED+'F1 SCORE (Train) : '+color.END+color.BOLD+str(f1_score(y_train,
print('\n'+color.RED+'F1 SCORE (Test) : '+color.END+color.BOLD+str(score)+color.END)

# recall
recall = metrics.recall_score(y_test, test_pred)
print('\n'+color.RED+'RECALL (Train): '+color.END+color.BOLD+str(metrics.recall_score
print('\n'+color.RED+'RECALL (Test): '+color.END+color.BOLD+str(recall)+color.END)
```

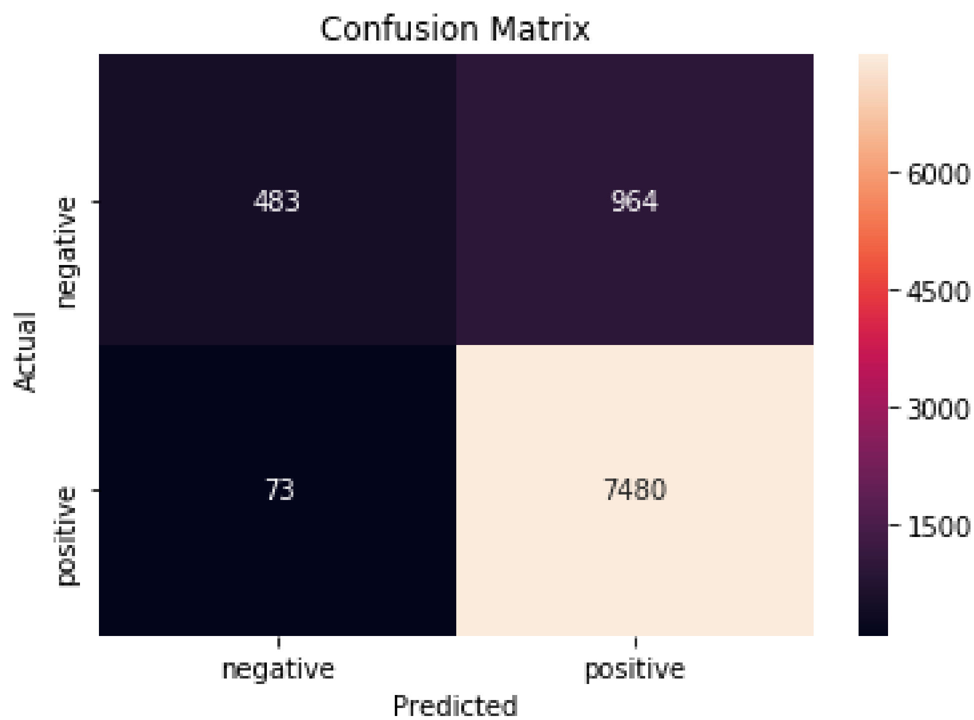
```python
# precision
precision = metrics.precision_score(y_test, test_pred)
print('\n'+color.RED+'PRECISION  (Train) : '+color.END+color.BOLD+str(metrics.precisio
print('\n'+color.RED+'PRECISION  (Test) : '+color.END+color.BOLD+str(precision)+color
```

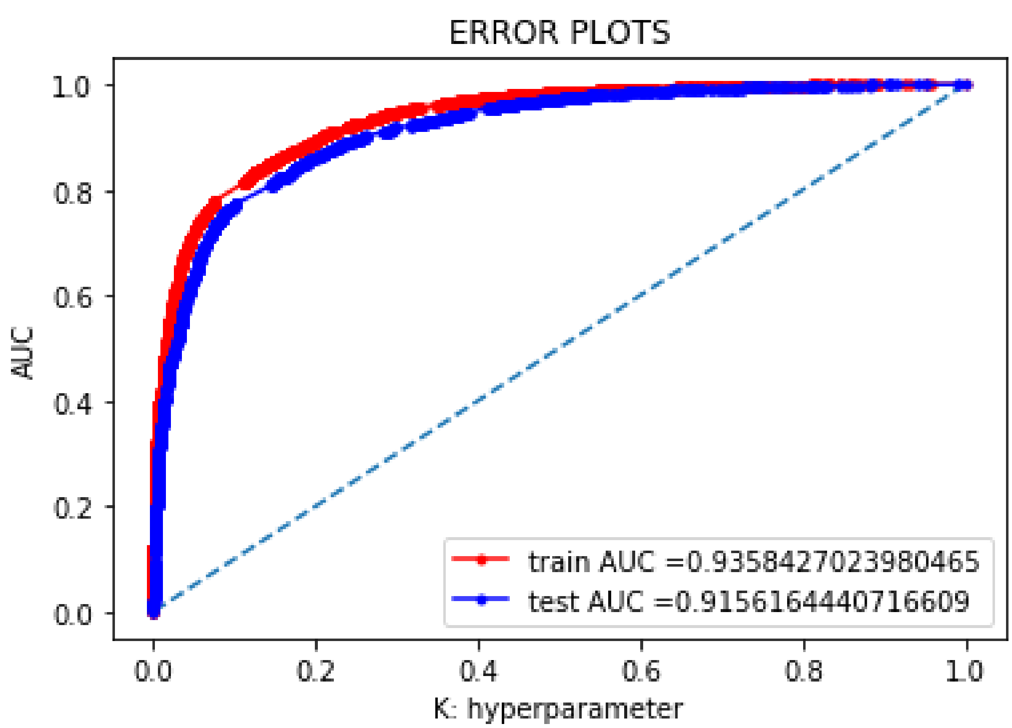
```
(21000, 13) (9000, 13) (21000,) (9000,)
```

Confusion Matrix for Train data



Confusion Matrix for Test data


Confusion Matrix
483
964
73
7480
6000
4500
3000
1500
negative
positive
Actual
negative
positive
Predicted


ERROR PLOTS
1.0
0.8
0.6
0.4
0.2
0.0
AUC
train AUC =0.9358427023980465
test AUC =0.9156164440716609
0.0
0.2
0.4
0.6
0.8
1.0
K: hyperparameter

```
AUC (Train): 0.9358427023980465

AUC (Test): 0.9156164440716609

F1 SCORE (Train) : 0.9407600096543217

F1 SCORE (Test) : 0.9351753453772583

RECALL (Train): 0.992867655383222

RECALL (Test): 0.9903349662385807

PRECISION  (Train) : 0.8938490546807318

PRECISION  (Test) : 0.8858360966366651
```

5.5 [4.4] Word2Vec

```
In [50]: X = final['preprocessed_reviews']
         y = final['Score']

         # split the data set into train and test
         X_train, x_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0

         print(X_train.shape, x_test.shape, y_train.shape, y_test.shape)

(53089,) (22753,) (53089,) (22753,)
```

```
In [51]: # Train your own Word2Vec model using your own text corpus

         # Train data
         list_of_sentance=[]
         for sentance in X_train:
             list_of_sentance.append(sentance.split())

         # Test data
         list_of_test_sentence = []
         for sentance in x_test:
             list_of_test_sentence.append(sentance.split())
```

```
In [52]: # Using Google News Word2Vectors

         # in this project we are using a pretrained model by google
         # its 3.3G file, once you load this into your memory
         # it occupies ~9Gb, so please do this step only if you have >12G of ram
         # we will provide a pickle file wich contains a dict ,
```

```
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    # train data
    w2v_model_tr=Word2Vec(list_of_sentance,min_count=5,size=50, workers=4)
    # train model on test data
    w2v_model_test = Word2Vec(list_of_test_sentence,min_count=5,size=50, workers=4)
    print(w2v_model_tr.wv.most_similar('great'))
    print('='*50)
    print(w2v_model_tr.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bi
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, 
```

```
[('fantastic', 0.8369816541671753), ('awesome', 0.83486008644104), ('good', 0.8010945916175842
==================================================
[('greatest', 0.7605360746383667), ('best', 0.6946088075637817), ('tastiest', 0.69453716278076
```

In [53]:
```
# train data operation
w2v_train_words = list(w2v_model_tr.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_train_words))
print("sample words ", w2v_train_words[0:50])
```

```
number of words that occured minimum 5 times  13914
sample words  ['product', 'china', 'known', 'would', 'not', 'ordered', 'share', 'concerns', 'f
```

In [54]:
```
## test data operation
w2v_test_words = list(w2v_model_test.wv.vocab)
```

```
print("number of words that occured minimum 5 times ",len(w2v_test_words))
print("sample words ", w2v_test_words[0:50])
```

```
number of words that occured minimum 5 times  9272
sample words  ['recently', 'recieved', 'samples', 'energy', 'husband', 'son', 'used', 'next',
```

5.6 [4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

In [55]:
```
# average Word2Vec

# train data operation
exists = os.path.isfile(avg_w2v_trained_model_100000)
exists = False
if exists:
    print("yes exist")
    final_w2v_train = load(avg_w2v_trained_model_100000)
else:
    print("not exist")
    # compute average word2vec for each review.
    final_w2v_train = []; # the avg-w2v for each sentence/review is stored in this li
    for sent in tqdm(list_of_sentance): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might ne
        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_train_words:
                vec = w2v_model_tr.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        final_w2v_train.append(sent_vec)
    print(len(final_w2v_train))
    print(len(final_w2v_train[0]))
#     dump(final_w2v_train,avg_w2v_trained_model_100000)


# test data operation
exists = os.path.isfile(avg_w2v_test_model_100000)
exists = False
if exists:
    print("yes exist")
    final_w2v_test = load(avg_w2v_test_model_100000)
else:
    print("not exist")
    final_w2v_test = []; # the avg-w2v for each sentence/review is stored in this lis
    for sent in tqdm(list_of_test_sentence): # for each review/sentence
```

```
            sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might ne
            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_test_words:
                    vec = w2v_model_test.wv[word]
                    sent_vec += vec
                    cnt_words += 1
            if cnt_words != 0:
                sent_vec /= cnt_words
            final_w2v_test.append(sent_vec)
        print(len(final_w2v_test))
        print(len(final_w2v_test[0]))
    #     dump(final_w2v_test,avg_w2v_test_model_100000)
```

```
  0%|          | 0/53089 [00:00<?, ?it/s]

not exist


100%|| 53089/53089 [01:42<00:00, 517.62it/s]
  4%|         | 926/22753 [00:01<00:25, 870.92it/s]

53089
50
not exist


100%|| 22753/22753 [00:30<00:00, 754.78it/s]

22753
50
```

5.6.1 Hyper param Tuning using GridSearch

finding 'max depth' which have maximum AUC Score

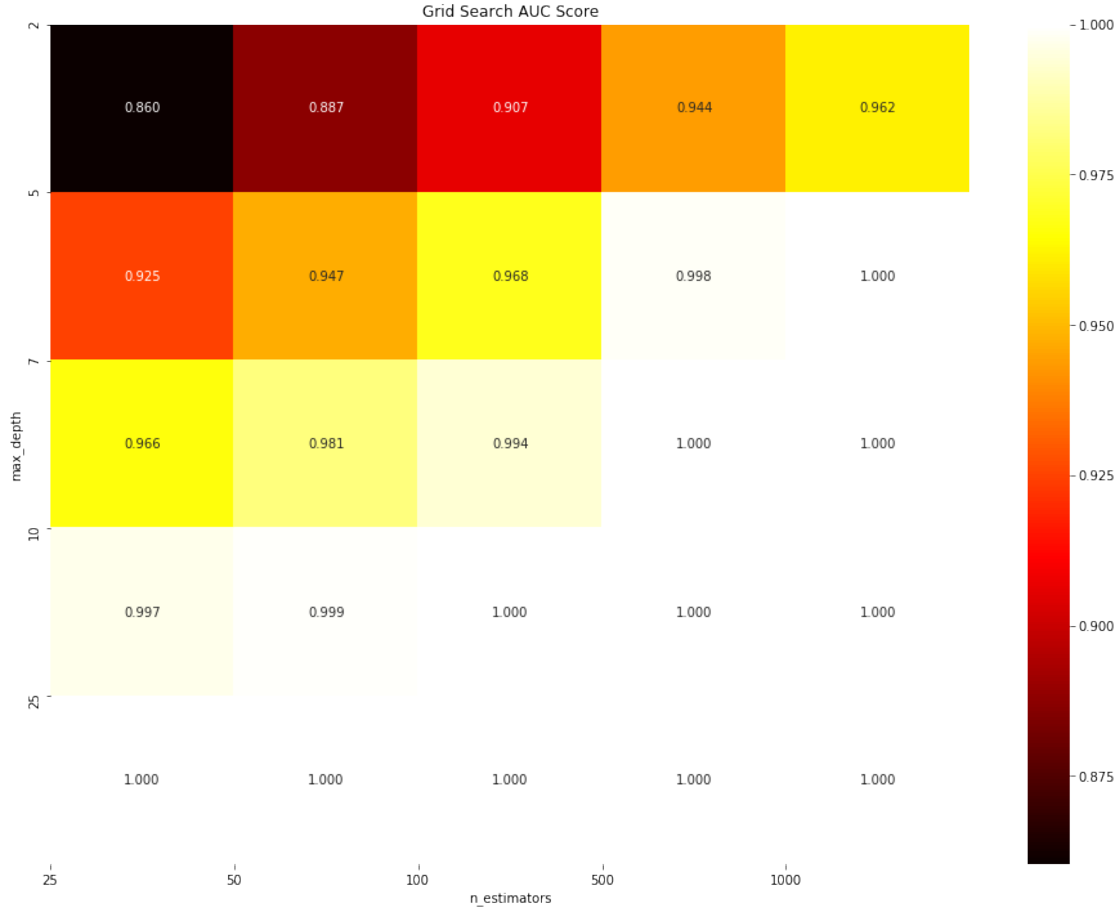
```
In [56]: w2v_train_path = '/home/pranay/ML Hyperparam Tune/GBDT/w2v_train_hyperparam_tuned'
         exists = os.path.isfile(w2v_train_path)

         if exists:
             print("yes exists")
             w2v_train = load(w2v_train_path)
         else:
             print("not exists")
             final_w2v_train = np.asarray(final_w2v_train)
```

```
        w2v_train = finding_best_hyperparam(final_w2v_train,y_train)
        dump(w2v_train,w2v_train_path )


    # view the complete results (list of named tuples)
    print("=======Training========")
    print (w2v_train.best_score_)
    print (w2v_train.best_params_)
    print (w2v_train.best_estimator_)

    best_depth_size = w2v_train.best_params_.get("max_depth", "")
    best_estimators = w2v_train.best_params_.get("n_estimators", "")
```

```
yes exists
=======Training========
0.9010064822913382
{'max_depth': 25, 'n_estimators': 1000}
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=25, min_child_weight=1, missing=nan,
       n_estimators=1000, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=1, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)
```

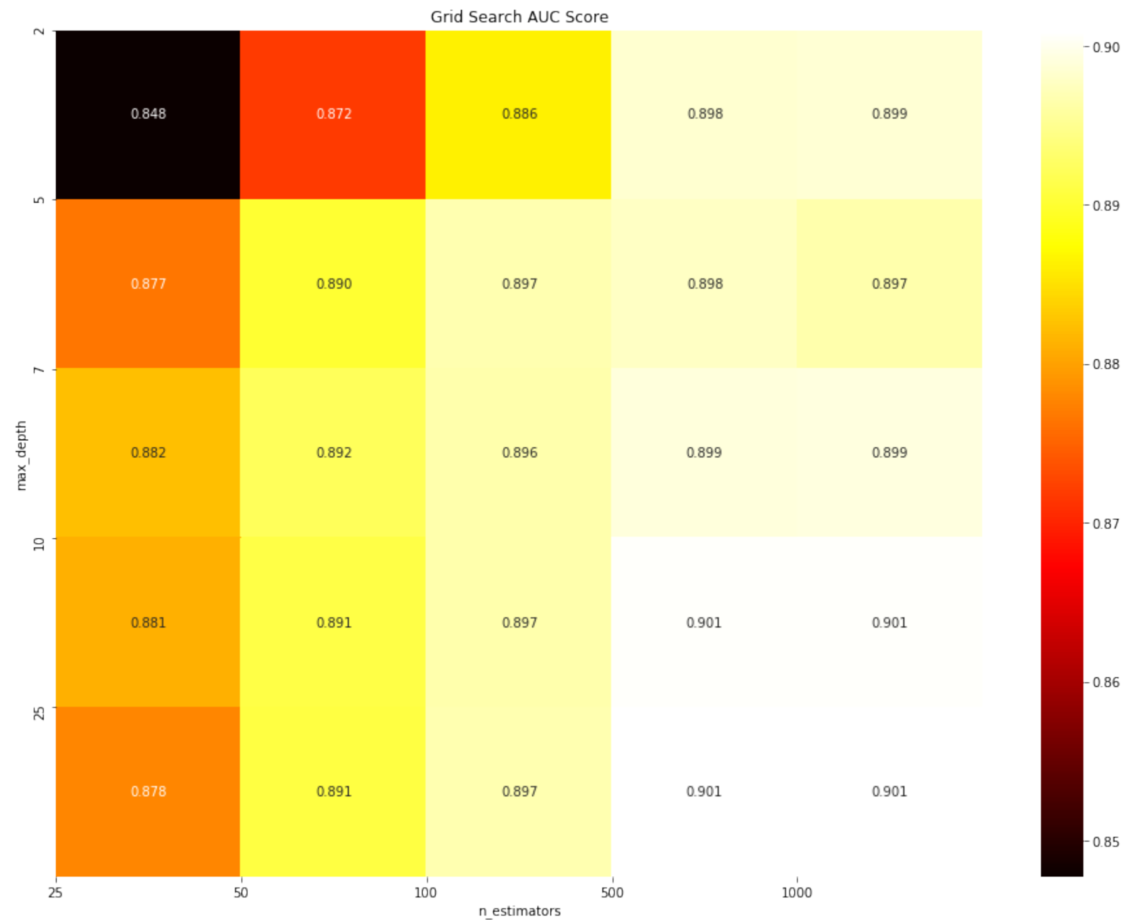
In [57]:
```
print('\n'+color.BOLD +'AUC Train data'+color.END)
plotHeatMap(w2v_train,'trained')
```

AUC Train data




```
In [58]: print('\n'+color.BOLD +'AUC Validation data'+color.END)
         plotHeatMap(w2v_train,'test')
```

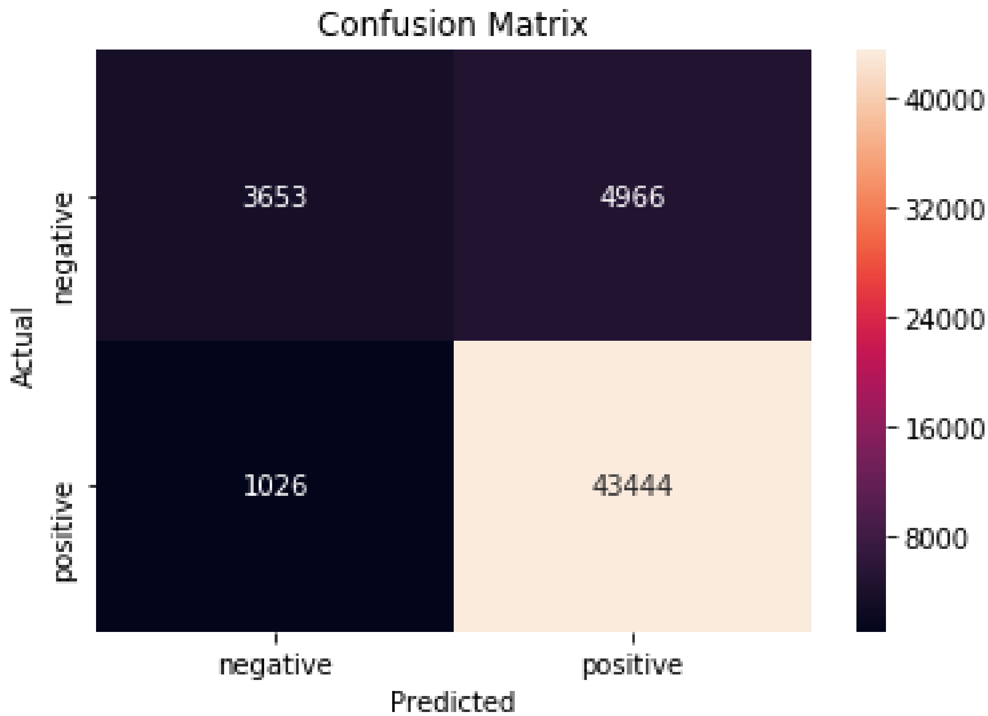
AUC Validation data



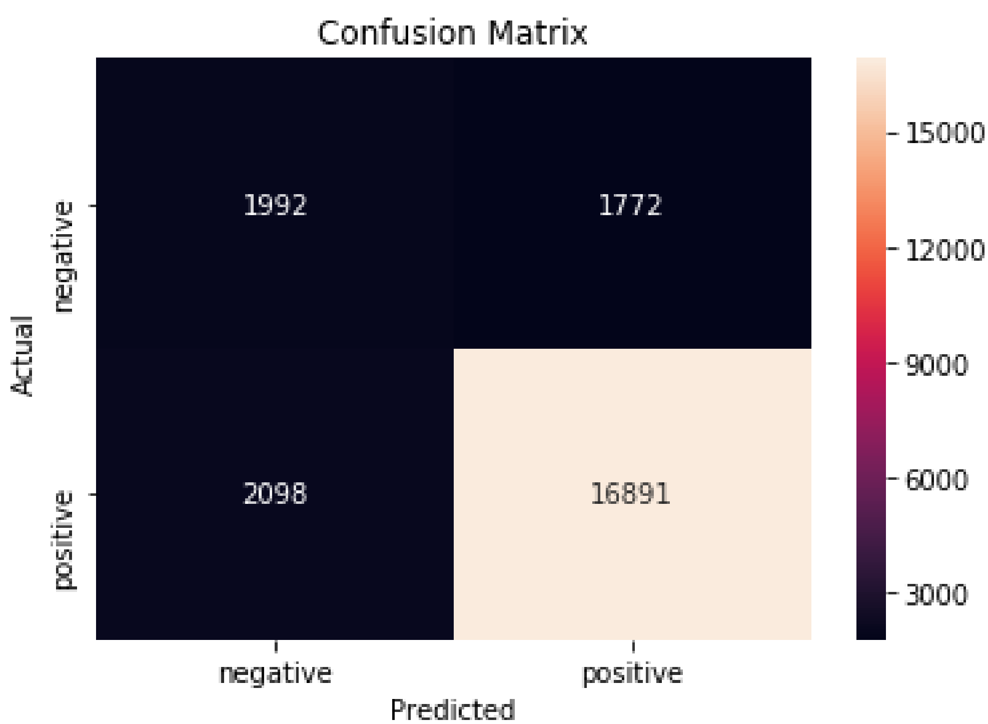
In [77]:
```
final_w2v_test = np.asarray(final_w2v_test)
final_w2v_train = np.asarray(final_w2v_train)

optimal_model = XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=-1, nthread=None,
       objective='binary:logistic', random_state=1, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)

# fitting the model
optimal_model.fit(final_w2v_train, y_train)

# predict the response
test_pred = optimal_model.predict(final_w2v_test)
train_pred = optimal_model.predict(final_w2v_train)
```

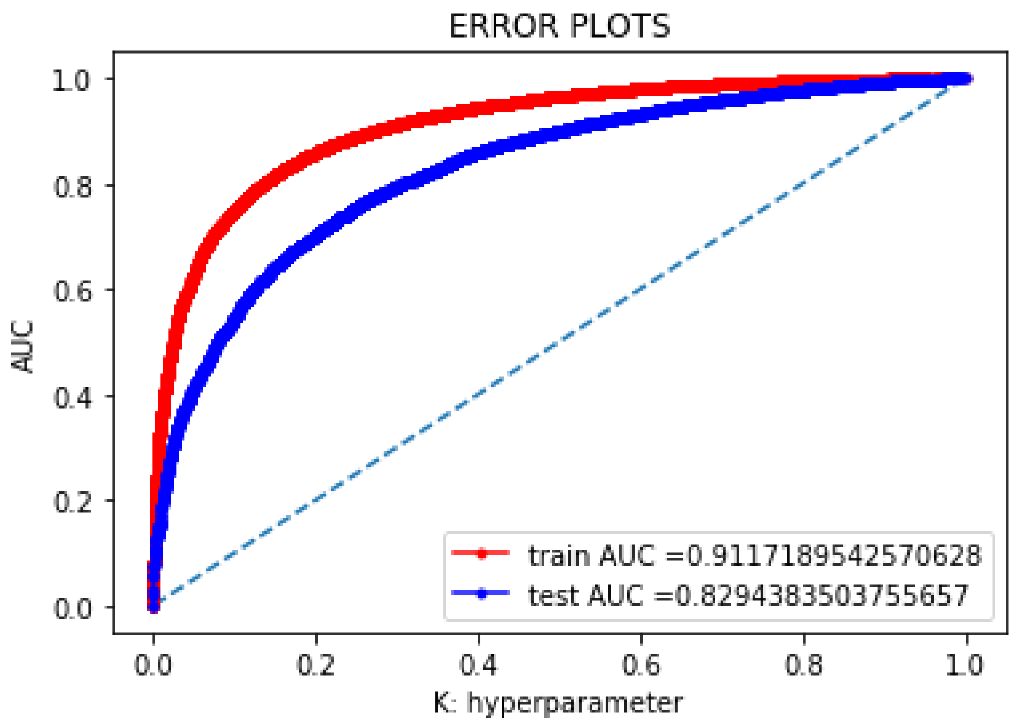
```
print('\n'+color.RED+'Max Depth : '+color.END+color.BOLD+str(3)+color.END)
print('\n'+color.RED+'Estimators : '+color.END+color.BOLD+str(100)+color.END)


# plot confusion matrix

print('\n'+color.BOLD +'Confusion Matrix for Train data'+color.END)
plotConfusionMatrix(y_train,train_pred)

print('\n'+color.BOLD +'Confusion Matrix for Test data'+color.END)
plotConfusionMatrix(y_test,test_pred)

# plot AUC
train_auc,test_auc = plotAUC_ROC(optimal_model,final_w2v_train, y_train,final_w2v_tes
print('\n'+color.RED+'AUC (Train): '+color.END+color.BOLD+str(train_auc)+color.END)
print('\n'+color.RED+'AUC (Test): '+color.END+color.BOLD+str(test_auc)+color.END)

# f1 score
score = f1_score(y_test,test_pred)
print('\n'+color.RED+'F1 SCORE (Train) : '+color.END+color.BOLD+str(f1_score(y_train,
print('\n'+color.RED+'F1 SCORE (Test) : '+color.END+color.BOLD+str(score)+color.END)

# recall
recall = metrics.recall_score(y_test, test_pred)
print('\n'+color.RED+'RECALL (Train): '+color.END+color.BOLD+str(metrics.recall_score
print('\n'+color.RED+'RECALL (Test): '+color.END+color.BOLD+str(recall)+color.END)

# precision
precision = metrics.precision_score(y_test, test_pred)
print('\n'+color.RED+'PRECISION  (Train) : '+color.END+color.BOLD+str(metrics.precisio
print('\n'+color.RED+'PRECISION  (Test) : '+color.END+color.BOLD+str(precision)+color
```

```
Max Depth : 3

Estimators : 100

Confusion Matrix for Train data
```



Confusion Matrix for Test data






```
AUC (Train): 0.9117189542570628

AUC (Test): 0.8294383503755657

F1 SCORE (Train) : 0.9354866494401377

F1 SCORE (Test) : 0.8972166153192394

RECALL (Train): 0.976928266246908

RECALL (Test): 0.8895149823582074

PRECISION  (Train) : 0.8974178888659368

PRECISION  (Test) : 0.9050527782242941
```

5.7 [4.4.1.2] TFIDF weighted W2v

```
In [61]: X = final['preprocessed_reviews']
         y = final['Score']
```

```
# split the data set into train and test
X_train, x_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0

print(X_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(53089,) (22753,) (53089,) (22753,)
```

In [62]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [63]:
```
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect.idf_)))

# TF-IDF weighted Word2Vec

# Train data operation

# store model to hard disk if exist then load model directly from memory
exists = os.path.isfile(w2v_tf_idf_trained_model_100000)
exists = False
if exists:
    print("yes exist")
    final_tfidf_w2v_tr = load(w2v_tf_idf_trained_model_100000)
else:
    print("not exist")
    tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
    # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = t

    final_tfidf_w2v_tr = []; # the tfidf-w2v for each sentence/review is stored in th
    row=0;
    for sent in tqdm(list_of_sentance): # for each review/sentence
        sent_vec = np.zeros(50) # as word vectors are of zero length
        weight_sum =0; # num of words with a valid vector in the sentence/review
        for word in sent: # for each word in a review/sentence
            if word in w2v_train_words and word in tfidf_feat:
                vec = w2v_model_tr.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
            final_tfidf_w2v_tr.append(sent_vec)
            row += 1
    #     dump(final_tfidf_w2v_tr,w2v_tf_idf_trained_model_100000)


    # Test data operation ============

    # store model to hard disk if exist then load model directly from memory
    exists = os.path.isfile(w2v_tf_idf_test_model_100000)
    exists = False
    if exists:
        print("yes exist")
        final_tfidf_w2v_test = load(w2v_tf_idf_test_model_100000)

    else:
        print("not exist")
        # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = t
        final_tfidf_w2v_test = []; # the tfidf-w2v for each sentence/review is stored in
        row=0;
        for sent in tqdm(list_of_test_sentence): # 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_test_words and word in tfidf_feat:
                    vec = w2v_model_test.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
            final_tfidf_w2v_test.append(sent_vec)
            row += 1
    #     dump(final_tfidf_w2v_test,w2v_tf_idf_test_model_100000)
```

```
  0%|          | 7/53089 [00:00<28:05, 31.50it/s]

not exist


100%|| 53089/53089 [43:17<00:00, 21.13it/s]
  0%|          | 32/22753 [00:01<19:51, 19.07it/s]
```

```
not exist
```

```
100%|| 22753/22753 [16:18<00:00, 23.25it/s]
```

5.7.1 Hyper param Tuning using GridSearch

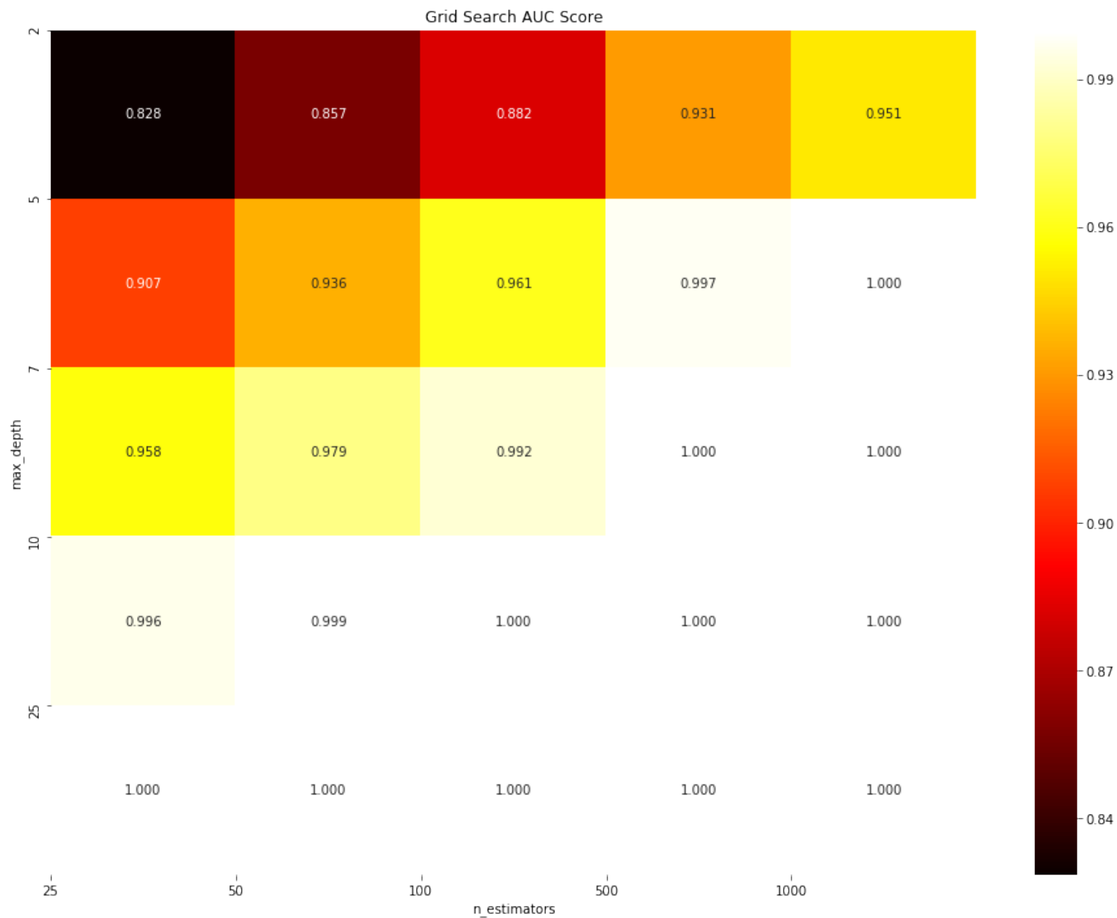
finding 'max depth' which have maximum AUC Score

```
In [64]: tfidf_w2v_train_path = '/home/pranay/ML Hyperparam Tune/GBDT/tfidf_w2v_train_hyperpara
         exists = os.path.isfile(tfidf_w2v_train_path)

         if exists:
             print("yes exists")
             w2v_tfidf_train = load(tfidf_w2v_train_path)
         else:
             print("not exists")
             final_tfidf_w2v_tr = np.asarray(final_tfidf_w2v_tr)
             w2v_tfidf_train = finding_best_hyperparam(final_tfidf_w2v_tr,y_train)
             dump(w2v_tfidf_train,tfidf_w2v_train_path )


         # view the complete results (list of named tuples)
         print("=======Training========")
         print (w2v_tfidf_train.best_score_)
         print (w2v_tfidf_train.best_params_)
         print (w2v_tfidf_train.best_estimator_)

         best_depth_size = w2v_tfidf_train.best_params_.get("max_depth", "")
         best_estimators = w2v_tfidf_train.best_params_.get("n_estimators", "")
```

```
yes exists
=======Training========
0.8782853837864152
{'max_depth': 10, 'n_estimators': 500}
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=10, min_child_weight=1, missing=nan,
       n_estimators=500, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=1, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)
```

```
In [65]: print('\n'+color.BOLD +'AUC Train data'+color.END)
         plotHeatMap(w2v_tfidf_train,'trained')
```

AUC Train data



In [66]:
```
print('\n'+color.BOLD +'AUC Validation data'+color.END)
plotHeatMap(w2v_tfidf_train,'validation')
```

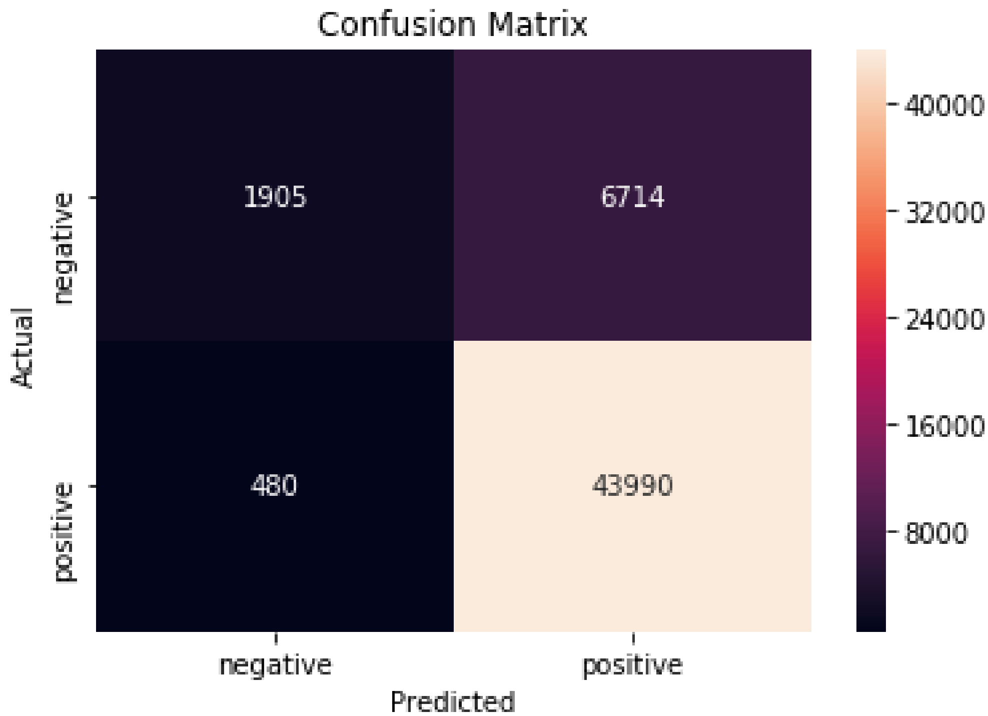
AUC Validation data

5.8 GBDT on TFIDF - W2V

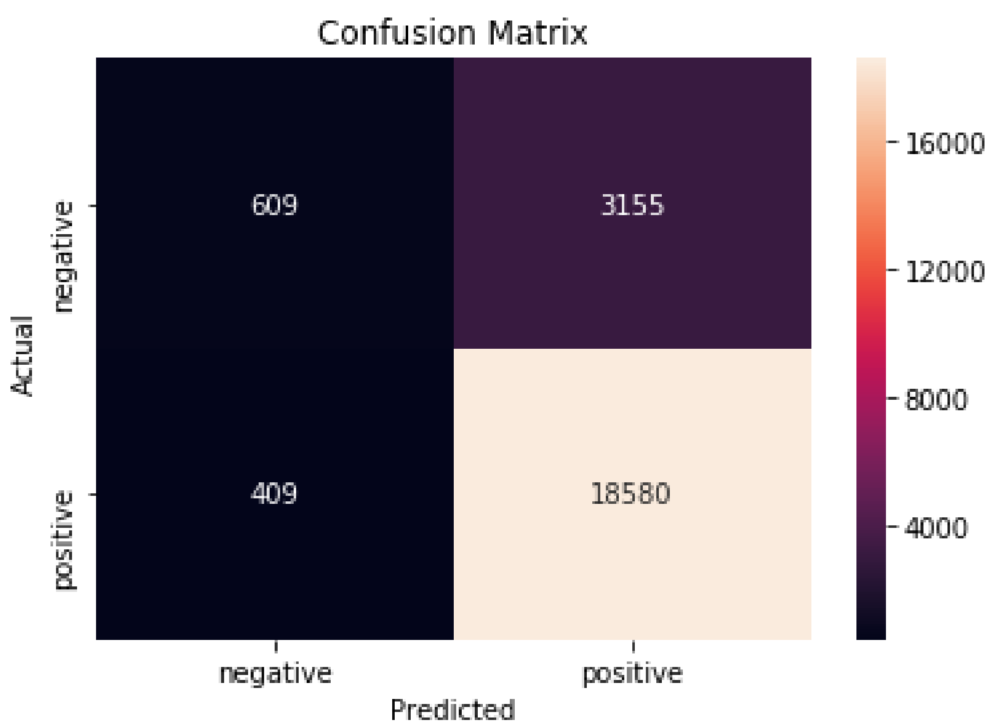
In [76]:
```
final_tfidf_w2v_test = np.asarray(final_tfidf_w2v_test)
final_tfidf_w2v_tr = np.asarray(final_tfidf_w2v_tr)


optimal_model = XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balance
       colsample_bylevel=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=2, min_child_weight=1, missing=None,
       n_estimators=80, n_jobs=-1, nthread=None,
       objective='binary:logistic', random_state=1, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=True,
       subsample=1)

# fitting the model
optimal_model.fit(final_tfidf_w2v_tr, y_train)
# predict the response
test_pred = optimal_model.predict(final_tfidf_w2v_test)
train_pred = optimal_model.predict(final_tfidf_w2v_tr)
```

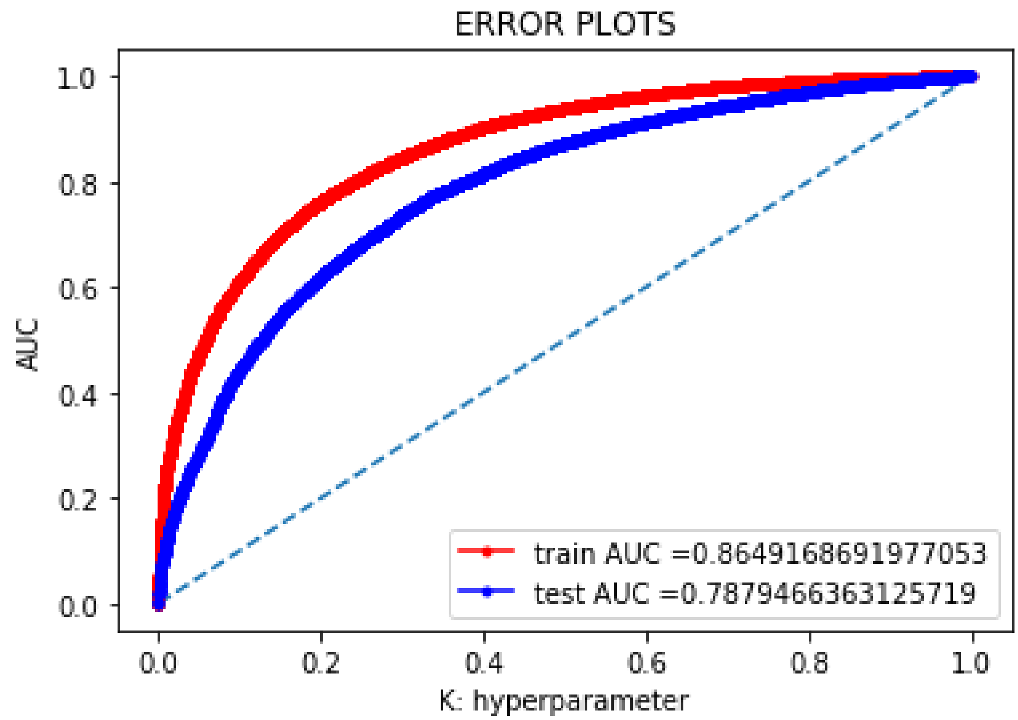
```
print('\n'+color.RED+'Max Depth : '+color.END+color.BOLD+str(2)+color.END)
print('\n'+color.RED+'Estimators: '+color.END+color.BOLD+str(80)+color.END)


# plot confusion matrix

print('\n'+color.BOLD +'Confusion Matrix for Train data'+color.END)
plotConfusionMatrix(y_train,train_pred)

print('\n'+color.BOLD +'Confusion Matrix for Test data'+color.END)
plotConfusionMatrix(y_test,test_pred)

# plot AUC
train_auc,test_auc = plotAUC_ROC(optimal_model,final_tfidf_w2v_tr, y_train,final_tfidf
print('\n'+color.RED+'AUC (Train): '+color.END+color.BOLD+str(train_auc)+color.END)
print('\n'+color.RED+'AUC (Test): '+color.END+color.BOLD+str(test_auc)+color.END)

# f1 score
score = f1_score(y_test,test_pred)
print('\n'+color.RED+'F1 SCORE (Train) : '+color.END+color.BOLD+str(f1_score(y_train,t
print('\n'+color.RED+'F1 SCORE (Test) : '+color.END+color.BOLD+str(score)+color.END)

# recall
recall = metrics.recall_score(y_test, test_pred)
print('\n'+color.RED+'RECALL (Train): '+color.END+color.BOLD+str(metrics.recall_score
print('\n'+color.RED+'RECALL (Test): '+color.END+color.BOLD+str(recall)+color.END)

# precision
precision = metrics.precision_score(y_test, test_pred)
print('\n'+color.RED+'PRECISION  (Train) : '+color.END+color.BOLD+str(metrics.precisio
print('\n'+color.RED+'PRECISION  (Test) : '+color.END+color.BOLD+str(precision)+color
```

```
Max Depth : 2

Estimators: 80

Confusion Matrix for Train data
```



Confusion Matrix for Test data






```
AUC (Train): 0.8649168691977053

AUC (Test): 0.7879466363125719

F1 SCORE (Train) : 0.9244121293630614

F1 SCORE (Test) : 0.9124840388959827

RECALL (Train): 0.989206206431302

RECALL (Test): 0.9784612143872768

PRECISION  (Train) : 0.8675844114862733

PRECISION  (Test) : 0.8548424200598114
```

6 [6] Conclusions

In [84]:
```
import pandas as pd
from prettytable import PrettyTable

print(color.BOLD+'\t\t\t\t GBDT  '+color.END)
print('\n')

print(color.BOLD+'For BOW and TFIDF, We have considered 85k points'+color.END)
print(color.BOLD+'For BOW- Additional Feature and TFIDF- Additional Feature, We have c


x = PrettyTable()
x.field_names =  ['Metric','BOW','BOW-Additional Feature', 'TFIDF', 'TFIDF- Additional

x.add_row(["Max Depth ", 4,4,3,3,3,2])
x.add_row(["no Estimator ", 100,100,100,100,100,80])


x.add_row(["AUC Train ", 0.91487,0.99454,0.90289,0.93584,0.91171,0.86491])
x.add_row(["AUC Test ", 0.90164,0.96708,0.88776,0.91561,0.82943,0.78794])

x.add_row(["F1 SCORE Train ", 0.93603,0.98221,0.93143,0.94076,0.93548,0.92441])
x.add_row(["F1 SCORE Test ", 0.93118,0.96068,0.92721,0.93517,0.89721,0.91248])

x.add_row(["RECALL Train ",0.99554,0.99575,0.99707,0.99286,0.97692,0.98920])
x.add_row(["RECALL Test ", 0.99294,0.98345,0.99483,0.99033,0.88951,0.97846])
93449
x.add_row(["PRECISION Train ", 0.88323,0.96904,0.87390,0.89384,0.89741,0.86751])
x.add_row(["PRECISION Test ",0.87665,0.93894,0.86819,0.88583,0.90502,0.85484])


print('\n')
print(x)
```

```
                                GBDT


For BOW and TFIDF, We have considered 85k points
For BOW- Additional Feature and TFIDF- Additional Feature, We have considered 30k points


+-------------------+---------+------------------------+---------+---------------------------+-
|       Metric      |   BOW   | BOW-Additional Feature |  TFIDF  | TFIDF- Additional Features |
+-------------------+---------+------------------------+---------+---------------------------+-
|     Max Depth     |    4    |           4            |    3    |             3             |
|   no Estimator    |   100   |          100           |   100   |            100            |
|     AUC Train     | 0.91487 |        0.99454         | 0.90289 |          0.93584          |
```

```
|    AUC Test      | 0.90164 |        0.96708         | 0.88776 |          0.91561           |
| F1 SCORE Train   | 0.93603 |        0.98221         | 0.93143 |          0.94076           |
|  F1 SCORE Test   | 0.93118 |        0.96068         | 0.92721 |          0.93517           |
|  RECALL Train    | 0.99554 |        0.99575         | 0.99707 |          0.99286           |
|   RECALL Test    | 0.99294 |        0.98345         | 0.99483 |          0.99033           |
| PRECISION Train  | 0.88323 |        0.96904         |  0.8739 |          0.89384           |
| PRECISION Test   | 0.87665 |        0.93894         | 0.86819 |          0.88583           |
+------------------+---------+------------------------+---------+----------------------------+-
```

In [ ]: