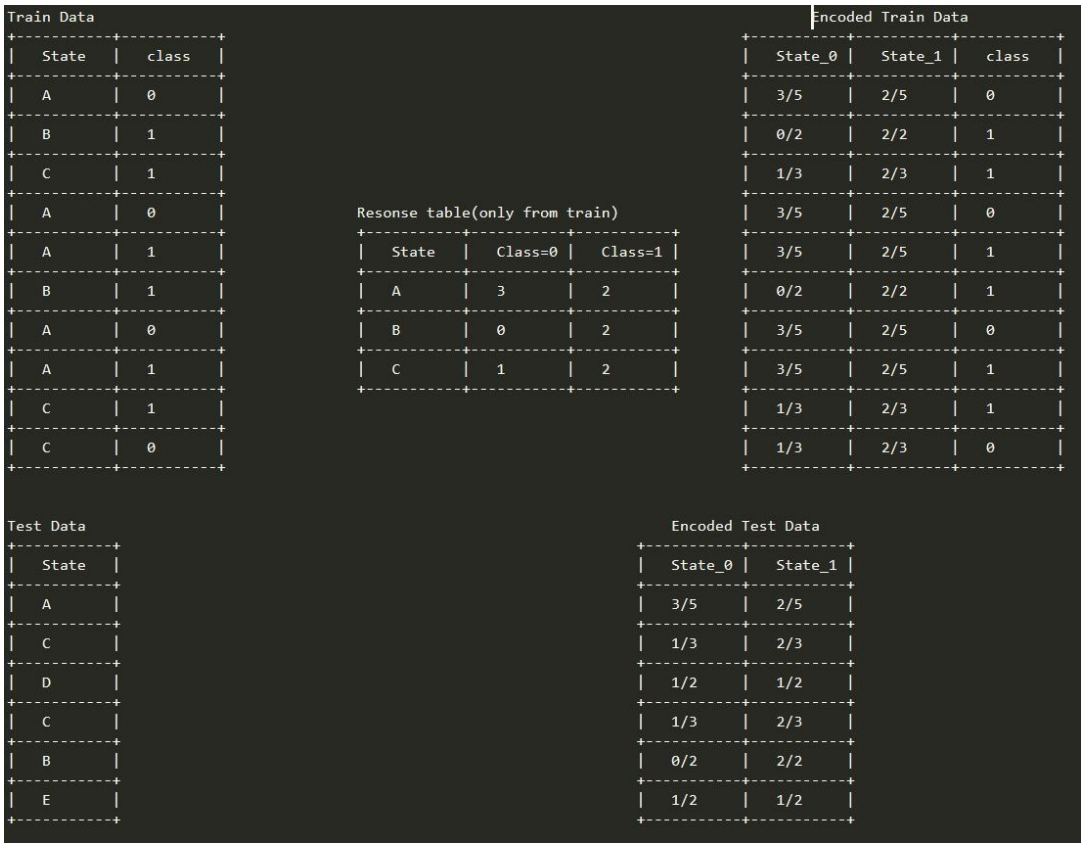
Assignment 9: GBDT

Response Coding: Example



Train Data

State	class
A	0
B	1
C	1
A	0
A	1
B	1
A	0
A	1
C	1
C	0

Resonse table(only from train)

State	Class=0	Class=1
A	3	2
B	0	2
C	1	2

Encoded Train Data

State_0	State_1	class
3/5	2/5	0
0/2	2/2	1
1/3	2/3	1
3/5	2/5	0
3/5	2/5	1
0/2	2/2	1
3/5	2/5	0
3/5	2/5	1
1/3	2/3	1
1/3	2/3	0

Test Data

State
A
C
D
C
B
E

Encoded Test Data

State_0	State_1
3/5	2/5
1/3	2/3
1/2	1/2
1/3	2/3
0/2	2/2
1/2	1/2

> The response tabel is built only on train dataset. For a category which is not there in train data and present in test data, we will encode them with default values Ex: in our test data if have State: D then we encode it as [0.5, 0.05]

1. **Apply GBDT on these feature sets**
   - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check the bellow example, include all 4 values as 4 features)
   - Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (Consider any two hyper parameters)**
   - Find the best hyper parameter which will give the maximum AUC value
   - find the best hyper paramter using k-fold cross validation/simple cross validation data
   - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task
3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the

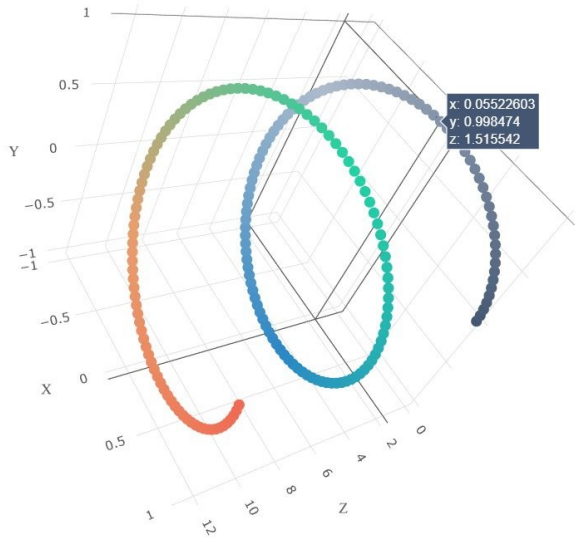


figure with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

or

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

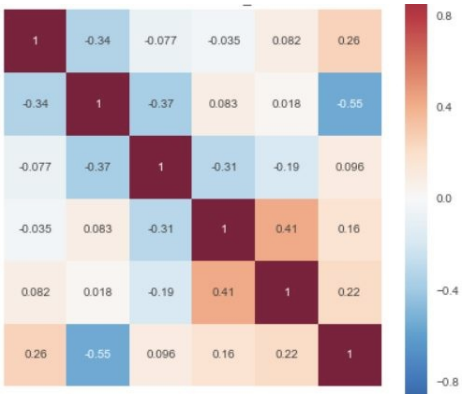
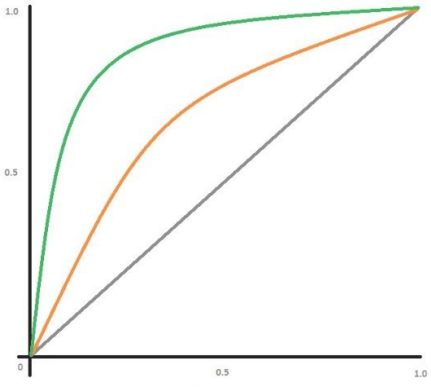


figure seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**

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



curve on both train and test.

- Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

4. You need to summarize the results at the end of the notebook, summarize it in the table format

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
from nltk.corpus import stopwords
import pickle

from tqdm import tqdm
import os
```

In [2]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students with the biggest
for learning my students learn in many different ways using all of our senses and multiple intelligences i use a
of techniques to help all my students succeed students in my class come from a variety of different backgrounds w
for wonderful sharing of experiences and cultures including native americans our school is a caring community of
learners which can be seen through collaborative student project based learning in and out of the classroom kinde
in my class love to work with hands on materials and have many different opportunities to practice a skill before
mastered having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curr
montana is the perfect place to learn about agriculture and nutrition my students love to role play in our preter
in the early childhood classroom i have had several kids ask me can we try cooking with real food i will take the
and create common core cooking lessons where we learn important math and writing concepts while cooking delicious
food for snack time my students will have a grounded appreciation for the work that went into making the food and
of where the ingredients came from as well as how it is healthy for their bodies this project would expand our le
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce make our
and mix up healthy plants from our classroom garden in the spring we will also create our own cookbooks to be pri
shared with families students will gain math and literature skills as well as a life long enjoyment for healthy c
nannan'
ss = sid.polarity_scores(for_sentiment)

for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

In [3]:

```
%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 plotly import plotly
# import plotly.offline as offline
# import plotly.graph_objs as go
#offline.init_notebook_mode()
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import matplotlib.pyplot as plt
from collections import Counter
import lightgbm as lgb
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
```

1. GBDT (xgboost/lightgbm)

1.1 Loading Data

In [4]:
```
#Load the main data
data = pd.read_csv('preprocessed_data_cleaned.csv',nrows = 50000)
```

In [5]:
```
#import the whole training data because of Project title,Project title is not present in preprocessed
#i have Preprocessed manually to after imported the file
preprocessed_title = pd.read_csv('preprocessed_project_title.csv',nrows = 50000)
```

In [6]:
```
preprocessed_title.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 1 columns):
 #   Column                      Non-Null Count  Dtype 
---  ------                      --------------  ----- 
 0   project_title_preprocessed  50000 non-null  object
dtypes: object(1)
memory usage: 390.8+ KB
```

In [7]:
```
print("Shape of Preprocessed_data:{0} and preprocessed Title data:{1}".format(data.shape,preprocessed_title.shape
```

```
Shape of Preprocessed_data:(50000, 9) and preprocessed Title data:(50000, 1)
```

In [8]:
```
print("Column of Preprocessed Data :{0}".format(data.columns))
```

```
Column of Preprocessed Data :Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

Merge the Original Data and Project_title

In [9]:
```
donor_choose_data = pd.concat([data,preprocessed_title],axis = 1)
print("Shape of the Dataset : {0}".format(donor_choose_data.shape))
```

```
Shape of the Dataset : (50000, 10)
```

In [10]:
```
print("Columns Of the Dataset : {0}".format(donor_choose_data.columns))
```

```
Columns Of the Dataset : Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price',
       'project_title_preprocessed'],
      dtype='object')
```

In [11]:
```
X = donor_choose_data.drop(columns = ['project_is_approved'], axis = True)
Y = donor_choose_data["project_is_approved"]
print("Shape of Independent Feature : {0} and Dependent Feature : {1}".format(X.shape , Y.shape))
```

```
Shape of Independent Feature : (50000, 9) and Dependent Feature : (50000,)
```

In [12]:
```
Y.head()
```

Out[12]:
```
0    1
1    1
2    1
3    1
4    1
Name: project_is_approved, dtype: int64
```

In [13]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,random_state = 10, test_size = 0.2)
X_train_final,X_cv,Y_train_final,Y_cv = train_test_split(X_train,Y_train,random_state = 12,test_size = 0.2)
print("Shape Of the Training Data : {0}".format(X_train_final.shape))
print("Shape of the Cross Validation Data : {0}".format(X_cv.shape))
print("Shape of the Test Data: {0}".format(X_test.shape))
```

```
Shape Of the Training Data : (32000, 9)
Shape of the Cross Validation Data : (8000, 9)
Shape of the Test Data: (10000, 9)
```

Responce Encoding

1.3 Make Data Model Ready: encoding eassay, and project_title

In [14]:
```
vectorizer = TfidfVectorizer(max_features = 5000,ngram_range = (1,2))
vectorizer.fit(X_train_final.essay.values)
x_train_essay_tfidf = vectorizer.transform(X_train_final.essay.values)
x_test_essay_tfidf = vectorizer.transform(X_test.essay.values)
x_cv_essay_tfidf = vectorizer.transform(X_cv.essay.values)
```

In [15]:
```
print("After Transforming the Features to Vector Form To check the shape of the features")
print("Vectorization -- shape of the training Data : {0}".format(x_train_essay_tfidf.shape))
print("Vectorization -- shape of the test Data : {0}".format(x_test_essay_tfidf.shape))
print("Vectorization -- shape of the validation Data : {0}".format(x_cv_essay_tfidf.shape))
```

```
After Transforming the Features to Vector Form To check the shape of the features
Vectorization -- shape of the training Data : (32000, 5000)
Vectorization -- shape of the test Data : (10000, 5000)
Vectorization -- shape of the validation Data : (8000, 5000)
```

Transform Tfidf Vector for Transform Features

In [16]:
```
vectorizer_project_title = TfidfVectorizer(max_features = 5000,ngram_range = (1,2))
vectorizer_project_title.fit(X_train_final.project_title_preprocessed.values)
x_train_project_title_tfidf = vectorizer.transform(X_train_final.project_title_preprocessed.values)
```

```
x_test_project_title_tfidf = vectorizer.transform(X_test.project_title_preprocessed.values)
x_cv_project_title_tfidf = vectorizer.transform(X_cv.project_title_preprocessed.values)
```

In [17]:

```
print("After Transforming the Features to Vector Form To check the shape of the features")
print("Vectorization -- shape of the training Data : {0}".format(x_train_project_title_tfidf.shape))
print("Vectorization -- shape of the test Data : {0}".format(x_test_project_title_tfidf.shape))
print("Vectorization -- shape of the validation Data : {0}".format(x_cv_project_title_tfidf.shape))
```

```
After Transforming the Features to Vector Form To check the shape of the features
Vectorization -- shape of the training Data : (32000, 5000)
Vectorization -- shape of the test Data : (10000, 5000)
Vectorization -- shape of the validation Data : (8000, 5000)
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [18]:

```
class ResponseEncoding:

    def __init__(self):

        self.__positive_category = None

        self.__negative_category = None

        self.__unique_category = None

    def fit(self,data,column_name,target_feature):

        feature_value = data[column_name].values
        #positive  category _ value Counts
        self.__positive_category = data[data[target_feature] == 1][column_name].value_counts().to_dict()
        #negative Category Value Counts
        self.__negative_category = data[data[target_feature] == 0][column_name].value_counts().to_dict()
        #total unique category
        self.__unique_category =  data[column_name].value_counts().to_dict()

    def get_values(self):
        return (self.__unique_category)

    def transform(self,data,column_name, weights = [0,0]):

        prob_class_zero = []

        prob_class_one = []

        feature_value = data[column_name].values

        for i in range(len(feature_value)):

            if feature_value[i] in self.__unique_category:

                if feature_value[i] in self.__positive_category:
                    prob_class_one.append(self.__positive_category[feature_value[i]] / self.__unique_category[fea
                else:
                    prob_class_one.append(0)
                if feature_value[i] in self.__negative_category:
                    prob_class_zero.append(self.__negative_category[feature_value[i]] / self.__unique_category[fe
                else:
                    prob_class_zero.append(0)
            else:
                prob_class_one.append(weights[0])
                prob_class_zero.append(weights[1])

        return np.hstack((np.array(prob_class_zero).reshape(-1,1),np.array(prob_class_one).reshape(-1,1)))
```

In [21]:

```
X_train_final_merge = pd.concat([X_train_final,Y_train_final],axis = 1)
print("Shape: {0}".format(X_train_final_merge.shape))
```

```
Shape: (32000, 10)
```

Categorical Encoding For Training Data

In [19]:

```
categorical_features  = ['school_state','teacher_prefix','project_grade_category','clean_categories','clean_subca
```

In [22]:

```
#Encoding for training Data
X_train_categorical_encoding_dict = {}
X_test_categorical_encoding_dict = {}
X_cv_categorical_encoding_dict = {}
for feature_name in categorical_features:
    responseEncoding = ResponseEncoding()
    responseEncoding.fit(X_train_final_merge,feature_name,'project_is_approved')
    X_train_categorical_encoding_dict[feature_name]  = responseEncoding.transform( data = X_train_final,\
                                                                                   column_name = feature_na
    X_test_categorical_encoding_dict[feature_name]  = responseEncoding.transform( data = X_test,\
                                                                                   column_name = feature_na
    X_cv_categorical_encoding_dict[feature_name]  = responseEncoding.transform( data = X_cv,\
                                                                                   column_name = feature_na
# X_train_final_school_state_zero,X_train_final_school_state_one = responseEncoding.transform( data = X_train_fi
#                                                                                    column_name = 'school_
```

In [23]:

```
#Encoding teacher_prefix category
categorical_features  = ['school_state','teacher_prefix','project_grade_category','clean_categories','clean_subca
print("Categorical Encoding for teacher_prefix Feature")
X_train_teacher_prefix_category,x_test_teacher_prefix_category,x_cv_test_prefix_category =X_train_categorical_enc
                          X_cv_categorical_encoding_dict['teacher_prefix']
print("After Transforming the Data Shape")

print("Shape of the Training Feature Category : {0}".format(X_train_teacher_prefix_category.shape))

print("Shape of the Test Feature Category : {0}".format(x_test_teacher_prefix_category.shape))

print("Shape of the Validation Feature Category : {0}".format(x_cv_test_prefix_category.shape))

print("=="* 100)
print("Categorical Encoding for Project_grade_category")
X_train_project_grade_category,X_test_project_grade_category,X_cv_project_grade_category = X_train_categorical_er
                          X_cv_categorical_encoding_dict['project_grade_category']
print("After Transforming the Data Shape")

print("Shape of the Training Feature Category : {0}".format(X_train_project_grade_category.shape))

print("Shape of the Test Feature Category : {0}".format(X_test_project_grade_category.shape))

print("Shape of the Validation Feature Category : {0}".format(X_cv_project_grade_category.shape))
print("=="* 60)
print("Categorical Encoding for teacher_school_State category")
X_train_student_state_category,x_test_student_state_prefix_category,x_cv_student_state_prefix_category = X_train_
                          X_cv_categorical_encoding_dict['school_state']
print("After Transforming the Data Shape")

print("Shape of the Training Feature Category : {0}".format(X_train_student_state_category.shape))

print("Shape of the Test Feature Category : {0}".format(x_test_student_state_prefix_category.shape))

print("Shape of the Validation Feature Category : {0}".format(x_cv_student_state_prefix_category.shape))

print("=="* 100)
print("Categorical Encoding for clean Project Subject Category")
X_train_project_subject_category,x_test_project_subject_category,x_cv_project_subject_category= X_train_categoric
                          X_cv_categorical_encoding_dict['clean_categories']
print("After Transforming the Data Shape")

print("Shape of the Training Feature Category : {0}".format(X_train_project_subject_category.shape))

print("Shape of the Test Feature Category : {0}".format(x_test_project_subject_category.shape))

print("Shape of the Validation Feature Category : {0}".format(x_cv_project_subject_category.shape))
print("=="* 100)
print("Categorical Encoding for Project Subject Sub Category")
X_train_project_subject_sub_category,x_test_project_subject_sub_category,x_cv_project_subject_sub_category = X_tr
                          X_cv_categorical_encoding_dict['clean_subcategories']
print("After Transforming the Data Shape")

print("Shape of the Training Feature Category : {0}".format(X_train_project_subject_sub_category.shape))

print("Shape of the Test Feature Category : {0}".format(x_test_project_subject_sub_category.shape))

print("Shape of the Validation Feature Category : {0}".format(x_cv_project_subject_sub_category.shape))
```

```
Categorical Encoding for teacher_prefix Feature
After Transforming the Data Shape
Shape of the Training Feature Category : (32000, 2)
Shape of the Test Feature Category : (10000, 2)
Shape of the Validation Feature Category : (8000, 2)
========================================================================================================================
================================================================================
Categorical Encoding for Project_grade_category
```

```
After Transforming the Data Shape
Shape of the Training Feature Category : (32000, 2)
Shape of the Test Feature Category : (10000, 2)
Shape of the Validation Feature Category : (8000, 2)
================================================================================================================================
=======
Categorical Encoding for teacher_school_State category
After Transforming the Data Shape
Shape of the Training Feature Category : (32000, 2)
Shape of the Test Feature Category : (10000, 2)
Shape of the Validation Feature Category : (8000, 2)
================================================================================================================================
================================================================================
Categorical Encoding for clean Project Subject Category
After Transforming the Data Shape
Shape of the Training Feature Category : (32000, 2)
Shape of the Test Feature Category : (10000, 2)
Shape of the Validation Feature Category : (8000, 2)
================================================================================================================================
================================================================================
Categorical Encoding for Project Subject Sub Category
After Transforming the Data Shape
Shape of the Training Feature Category : (32000, 2)
Shape of the Test Feature Category : (10000, 2)
Shape of the Validation Feature Category : (8000, 2)
```

Numerical Transformation

In [24]:

```
from sklearn.preprocessing import Normalizer

def NumericalScaling(x_train,x_cv,X_test,column_name):

    train_norm = Normalizer()
    train_norm.fit(x_train[column_name].values.reshape(-1,1))
    x_train_norm_transform = train_norm.fit_transform(x_train[column_name].values.reshape(-1,1))
    x_cv_norm = train_norm.transform(x_cv[column_name].values.reshape(-1,1))
    x_test_norm = train_norm.transform(X_test[column_name].values.reshape(-1,1))
    print("After Transforming the Data Shape")
    print("Shape of the Training Feature Category : {0}".format(x_train_norm_transform.shape))
    print("Shape of the Test Feature Category : {0}".format(x_cv_norm.shape))
    print("Shape of the Validation Feature Category : {0}".format(x_test_norm.shape))

    return (x_train_norm_transform,x_cv_norm,x_test_norm)
```

In [25]:

```
#Numerical Feature Normalization
#X_train.price.values.reshape(-1,1)
#this Line Conver the One dimentional array to multiple dimentional Array

#Numerical Encoding for the feature Price
print("Numerical Transformation for the Feature price")
x_train_price_norm,x_cv_price_norm,x_test_price_norm =NumericalScaling(X_train_final,X_cv,X_test,'price')


print("="*70)
#Numerical Feature Transformation for the feature Previous Teacher Project
print("Numerical Transformation for the Feature teacher_number_of_previously_posted_projects")
x_train_previous_project_norm,x_cv_previous_project_norm,x_test_previous_project_norm =\
NumericalScaling (X_train_final,X_cv,X_test,'teacher_number_of_previously_posted_projects')
```

```
Numerical Transformation for the Feature price
After Transforming the Data Shape
Shape of the Training Feature Category : (32000, 1)
Shape of the Test Feature Category : (8000, 1)
Shape of the Validation Feature Category : (10000, 1)
======================================================================
Numerical Transformation for the Feature teacher_number_of_previously_posted_projects
After Transforming the Data Shape
Shape of the Training Feature Category : (32000, 1)
Shape of the Test Feature Category : (8000, 1)
Shape of the Validation Feature Category : (10000, 1)
```

In [26]:

```
X_train_final.essay.values[0]
```

Out[26]:

```
'this second year school offer chinese class still brand new subject students six grade none ever china know litt
le china nothing chinese language this would really hard starters since totally different language considered dif
```

```
ficult language however students great passion learning chinese eager know china with passion move overcome diffi
culty meet in order help earlier quicker i use lot online resources make class vivid interesting meaningful my st
udents need chrome books make learning experience resourceful meaningful fun my students need chrome books especi
ally students start learn chinese first time life use chrome books get help online resource would help learn bett
er beginning get interested chinese learning make better foundation chinese learning last year used school chrome
books quizlet students loved helped memorize better chinese characters and chrome books students able enjoy chine
se culture online resources but since use often school school not enough every class i would like keep classroom
students share whenever want get information online nannan'
```

In [27]:

```
#https://towardsdatascience.com/design-your-own-sentiment-score-e524308cf787
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sentiment = SentimentIntensityAnalyzer()
def compute_sentiment_score(document):
    negative_sentiment_score = []
    positive_sentiment_score = []
    neutral_sentiment_score = []
    total_prob_score = []
    #iterate the Each Document
    for sentence in tqdm(document):
        #computing the sentiment score
        sentiment_score = sentiment.polarity_scores(sentence)

        # print(type(sentiment_score)) - here sentimet Score is a dictionary

        # Example Score
        #{{'neg': 0.038, 'neu': 0.726, 'pos': 0.236, 'compound': 0.9784}}

        negative_sentiment_score.append(sentiment_score['neg'])

        positive_sentiment_score.append(sentiment_score['pos'])

        neutral_sentiment_score.append(sentiment_score['neu'])

        total_prob_score.append(sentiment_score['compound'])

    return (np.array(positive_sentiment_score).reshape(-1,1) , np.array(negative_sentiment_score).reshape(-1,1)\
            ,np.array(neutral_sentiment_score).reshape(-1,1),\
           np.array(total_prob_score).reshape(-1,1))
```

In [28]:

```
#Sentiment Score for Training Data
X_train_positive_sentiment_score,X_train_negative_sentiment_score,X_train_neutral_sentiment_score,\
x_train_total_prob_score = compute_sentiment_score(X_train_final.essay.values)
print("Shape Of the Feature : {0}".format(X_train_positive_sentiment_score.shape))


#Sentiment Score for Testing Data
X_test_positive_sentiment_score,X_test_negative_sentiment_score,X_test_neutral_sentiment_score,\
x_test_total_prob_score = compute_sentiment_score(X_test.essay.values)

#Sentinment Score for Validation Data
X_cv_positive_sentiment_score,X_cv_negative_sentiment_score,X_cv_neutral_sentiment_score,\
x_cv_total_prob_score = compute_sentiment_score(X_cv.essay.values)
```

```
100%|████████████████████████████████████████████████████████████████| 32000/32000 [01:36<00:00, 330.2
0it/s]
  1%|▌                                                                | 52/10000 [00:00<00:19, 519.9
6it/s]
Shape Of the Feature : (32000, 1)
```

```
100%|████████████████████████████████████████████████████████████████| 10000/10000 [00:25<00:00, 388.9
0it/s]
100%|████████████████████████████████████████████████████████████████| 8000/8000 [00:22<00:00, 356.5
5it/s]
```

Merge all the Transformed Features

In [29]:

```
#Combine All the Features
from scipy.sparse import hstack

#Training Preprocessing Dataset
```

```
X_train_final_data = hstack((x_train_essay_tfidf,x_train_project_title_tfidf, X_train_teacher_prefix_category, X_
                     X_train_student_state_category, X_train_project_subject_category,X_train_project_subject_sub
                    x_train_price_norm,x_train_previous_project_norm,X_train_positive_sentiment_score,\
                    X_train_negative_sentiment_score,X_train_neutral_sentiment_score,\
                    x_train_total_prob_score,)).tocsr()


#Test Prprocessing final Dataset
X_test_final_data = hstack((x_test_essay_tfidf,x_test_project_title_tfidf, x_test_teacher_prefix_category, X_test
                    x_test_student_state_prefix_category, x_test_project_subject_category,x_test_project_subject_
                     x_test_price_norm,x_test_previous_project_norm,X_test_positive_sentiment_score,
                    X_test_negative_sentiment_score,X_test_neutral_sentiment_score,\
                        x_test_total_prob_score )).tocsr()

#Cross Validation Preprocessing Dataset
X_cv_final_data = hstack((x_cv_essay_tfidf,x_cv_project_title_tfidf, x_cv_test_prefix_category, X_cv_project_grad
                x_cv_student_state_prefix_category, x_cv_project_subject_category,x_cv_project_subject_sub_categor
                  x_cv_price_norm,x_cv_previous_project_norm,X_cv_positive_sentiment_score,
                           X_cv_negative_sentiment_score,X_cv_neutral_sentiment_score,\
                    x_cv_total_prob_score )).tocsr()

print("="*100)
print("Ater Encoding and Transforming Numerical and Categorical Features")
print(X_train_final_data.shape, Y_train_final.shape)
print(X_test_final_data.shape, Y_test.shape)
print(X_cv_final_data.shape, Y_cv.shape)
```

```
====================================================================================================
Ater Encoding and Transforming Numerical and Categorical Features
(32000, 10016) (32000,)
(10000, 10016) (10000,)
(8000, 10016) (8000,)
```

1.5 Appling Models on different kind of featurization as mentioned in the instructions

Apply GBDT on different kind of featurization as mentioned in the instructions

For Every model that you work on make sure you do the step 2 and step 3 of instrucations

In [30]:

```
def batchPredict(model,data):
    y_predict_list = []

    dataLoop = data.shape[0] - (data.shape[0] %1000)

    for i in range(0, dataLoop, 1000):
        y_predict_list.extend(model.predict_proba(data[i:i+1000])[:,1])
    #predict remainng data
    if data.shape[0] % 1000 != 0:
        y_predict_list.extend(model.predict_proba(data[dataLoop:])[:,1])
    return y_predict_list
```

In [31]:

```
#https://machinelearningmastery.com/configure-gradient-boosting-algorithm/


train_score = []
crossValidation_Score = []
train_auc,cv_auc = [],[]
#hyper param tuning
n_estimators=[10,20,30,50,60,70,80,90,100,150,200]
max_depth = [2,3,4,5,7,10]
for estimator in tqdm(n_estimators):
    for depth in max_depth:
        treetfidf = lgb.LGBMClassifier(learning_rate=0.1,n_estimators = estimator,max_depth = depth)
        treetfidf.fit(X_train_final_data, Y_train_final)

        y_train_pred = batchPredict(treetfidf, X_train_final_data)
        y_cv_pred = batchPredict(treetfidf, X_cv_final_data)

        train_auc.append(roc_auc_score(Y_train_final,y_train_pred))
        cv_auc.append(roc_auc_score(Y_cv, y_cv_pred))
```

```
100%|██████████| 11/11 [20:34<00:00, 112.26s/it]
```

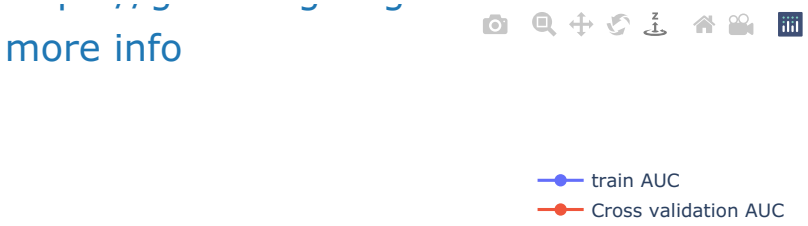
In [32]:

```
n_estimator_x = []
```

```
max_depth_y = max_depth * len(n_estimators)
for est in n_estimators:
    for depth in max_depth:
        n_estimator_x.append(est)
```

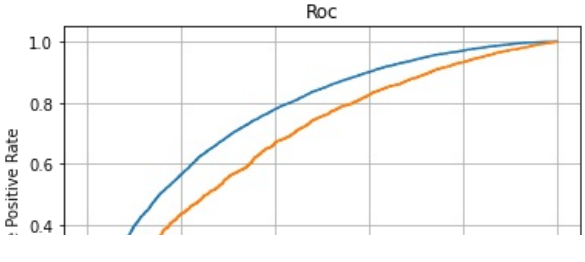
In [33]:

```
trainPlot = go.Scatter3d(x=n_estimator_x,y=max_depth_y,z=train_auc, name = 'train AUC')
validationPlot = go.Scatter3d(x=n_estimator_x,y=max_depth_y,z=cv_auc, name = 'Cross validation AUC')
data = [trainPlot, validationPlot]

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

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

plt.show()
```



In [34]:

```
max_depth = 2
best_estimator = 200
tree_gbdt = lgb.LGBMClassifier(max_depth = max_depth,n_estimators = best_estimator )
tree_gbdt.fit(X_train_final_data,Y_train_final)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batchPredict(tree_gbdt, X_train_final_data)
y_test_pred = batchPredict(tree_gbdt, X_test_final_data)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train_final, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Roc")
plt.grid()
plt.show()
```



In [35]:

```
def find_best_threshold(threshold, fpr, tpr):

    #https://numpy.org/doc/stable/reference/generated/numpy.argmax.html
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("The maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

def predict_with_best_threshold(proba, threshold):
    predictions = [1 if i >= threshold else 0 for i in proba]
    return predictions
```

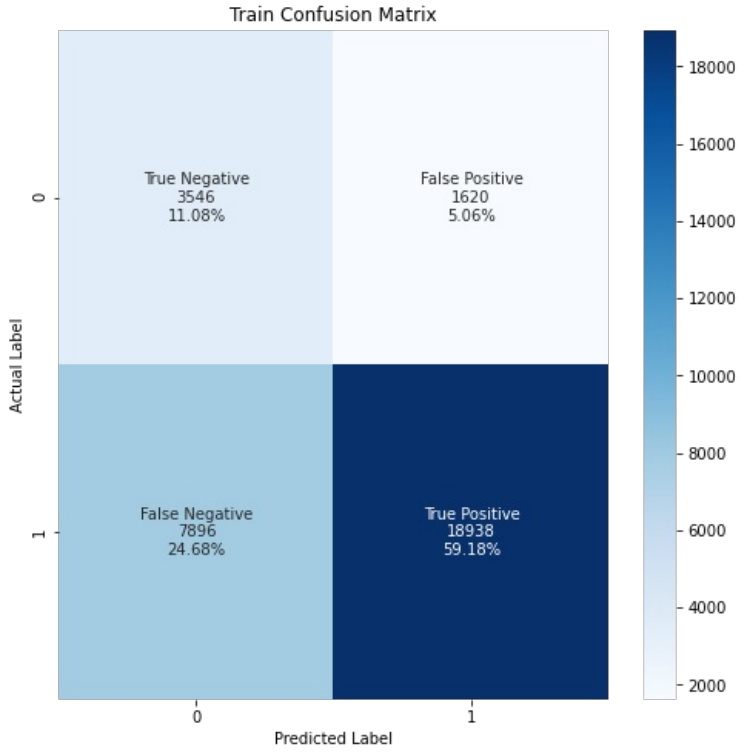
In [36]:

```
#SRC : https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
def confusionMatrixPlot(confusionMatrix,title):

    plt.figure(figsize = (8,8))
    group_names = ['True Negative','False Positive','False Negative','True Positive']
    group_counts = ["{0:0.0f}".format(value) for value in
                    confusionMatrix.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                         confusionMatrix.flatten()/np.sum(confusionMatrix)]
    labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
              zip(group_names,group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    sns.heatmap(confusionMatrix, annot=labels,fmt='', cmap='Blues')
    plt.xlabel('Predicted Label')
    plt.ylabel('Actual Label')
    plt.title(title)
```

In [37]:

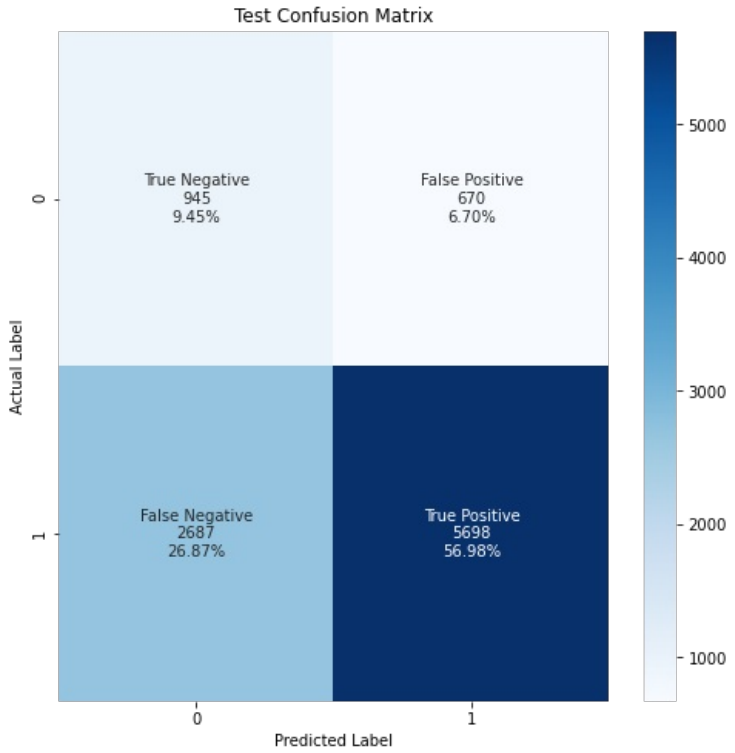
```
from sklearn.metrics import confusion_matrix

#training threshold,train_false_positive_rate,train_true_positive_rate
best_threshold = find_best_threshold(tr_thresholds, train_fpr, train_tpr)


y_train_prediction = predict_with_best_threshold(y_train_pred,best_threshold)

train_confusion_matrix = confusion_matrix(Y_train_final,y_train_prediction)
confusionMatrixPlot(train_confusion_matrix,"Train Confusion Matrix")
```

```
The maximum value of tpr*(1-fpr) 0.48443222610869413 for threshold 0.826
```



In [38]:
```
y_test_prediction = predict_with_best_threshold(y_test_pred, best_threshold)
test_confusion_matrix = confusion_matrix(Y_test, y_test_prediction)
print("Confusion Matrix For Test Data")
print("="* 100)
confusionMatrixPlot(test_confusion_matrix,"Test Confusion Matrix")
```

```
Confusion Matrix For Test Data
====================================================================================================
```



Build A Model AvgW2vec Tfidf

In [39]:
```
#word Inverted frequency

def Tfidf_word_2_vec(corpus,glovewords,idf_word_dict):
    average_word_to_vec_list = []
    for sentence in tqdm(corpus):
        vector = np.zeros(300)
        tfidf_weight = 0
        for word in sentence.split():
            if word in idf_word_dict and word in glovewords:
                #Compute the tfidfvalue
                tfidf = idf_word_dict[word] * sentence.split().count(word)/len(sentence.split())
                vector += glovewords[word] * tfidf
                tfidf_weight += tfidf
        if tfidf_weight != 0:
            #Caculatet the average of the vectorizer
            vector = vector / tfidf_weight
        average_word_to_vec_list.append(vector)

    return average_word_to_vec_list
```

In [40]:
```
vectorizer = TfidfVectorizer(max_features = 5000,ngram_range = (1,2))
```

```
vectorizer.fit(X_train_final.essay.values)
x_train_essay_tfidf = vectorizer.transform(X_train_final.essay.values)
x_test_essay_tfidf = vectorizer.transform(X_test.essay.values)
x_cv_essay_tfidf = vectorizer.transform(X_cv.essay.values)
```

In [41]:

```
vectorizer_project_title = TfidfVectorizer(max_features = 5000,ngram_range = (1,2))
vectorizer_project_title.fit(X_train_final.project_title_preprocessed.values)
x_train_project_title_tfidf = vectorizer.transform(X_train_final.project_title_preprocessed.values)
x_test_project_title_tfidf = vectorizer.transform(X_test.project_title_preprocessed.values)
x_cv_project_title_tfidf = vectorizer.transform(X_cv.project_title_preprocessed.values)
```

In [42]:

```
#inverse Document Frquescy of Essay Feature

idf_word_essay = dict(zip(vectorizer.get_feature_names(),vectorizer.idf_))
##inverse Document Frquescy of Essay Feature
idf_project_title_feature = dict(zip(vectorizer_project_title.get_feature_names(),vectorizer_project_title.idf_))
```

In [43]:

```
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [44]:

```
x_train_essay_tfidf_w2vec = Tfidf_word_2_vec(X_train_final.essay.values,model,idf_word_essay)
x_test_essay_tfidf_w2vec = Tfidf_word_2_vec(X_test.essay.values,model,idf_word_essay)
x_cv_essay_tfidf_w2vec = Tfidf_word_2_vec(X_cv.essay.values,model,idf_word_essay)

print("After Word to vec Transformation")
print("Shape of train data : ({0},{1})".format(len(x_train_essay_tfidf_w2vec),len(x_train_essay_tfidf_w2vec[0])))
print("Shape of test data : ({0},{1})".format(len(x_test_essay_tfidf_w2vec),len(x_test_essay_tfidf_w2vec[0])))
print("Shape of validation data : ({0},{1})".format(len(x_cv_essay_tfidf_w2vec),len(x_cv_essay_tfidf_w2vec[0])))
```

```
100%|████████████████████████████████████████████████████████████| 32000/32000 [03:58<00:00, 134.29it/s]
100%|████████████████████████████████████████████████████████████| 10000/10000 [01:11<00:00, 139.99it/s]
100%|████████████████████████████████████████████████████████████| 8000/8000 [00:52<00:00, 152.14it/s]

After Word to vec Transformation
Shape of train data : (32000,300)
Shape of test data : (10000,300)
Shape of validation data : (8000,300)
```

In [45]:

```
x_train_project_tfidf_w2vec = Tfidf_word_2_vec(X_train_final.project_title_preprocessed.values,model,idf_project_
x_test_project_tfidf_w2vec = Tfidf_word_2_vec(X_test.project_title_preprocessed.values,model,idf_project_title_fe
x_cv_project_tfidf_w2vec = Tfidf_word_2_vec(X_cv.project_title_preprocessed.values,model,idf_project_title_featur

print("After Word to vec Transformation")
print("Shape of train data : ({0},{1})".format(len(x_train_project_tfidf_w2vec),len(x_train_project_tfidf_w2vec[0
print("Shape of test data : ({0},{1})".format(len(x_test_project_tfidf_w2vec),len(x_test_project_tfidf_w2vec[0])))
print("Shape of validation data : ({0},{1})".format(len(x_cv_project_tfidf_w2vec),len(x_cv_project_tfidf_w2vec[0]
```

```
100%|████████████████████████████████████████████████████████████| 32000/32000 [00:01<00:00, 16966.18it/s]
100%|████████████████████████████████████████████████████████████| 10000/10000 [00:00<00:00, 23607.54it/s]
100%|████████████████████████████████████████████████████████████| 8000/8000 [00:00<00:00, 22279.45it/s]
After Word to vec Transformation
Shape of train data : (32000,300)
Shape of test data : (10000,300)
Shape of validation data : (8000,300)
```

In [46]:

```
#Combine All the Features
from scipy.sparse import hstack

#Training Preprocessing Dataset
X_train_final_data = np.hstack((x_train_essay_tfidf_w2vec,x_train_project_tfidf_w2vec, X_train_teacher_prefix_cat
                     X_train_student_state_category, X_train_project_subject_category,X_train_project_subject_sub
                    x_train_price_norm,x_train_previous_project_norm,X_train_positive_sentiment_score,\
                    X_train_negative_sentiment_score,X_train_neutral_sentiment_score,\
                    x_train_total_prob_score,))
```

```
#Test Prprocessing final Dataset
X_test_final_data = np.hstack((x_test_essay_tfidf_w2vec,x_test_project_tfidf_w2vec, x_test_teacher_prefix_categor
                    x_test_student_state_prefix_category, x_test_project_subject_category,x_test_project_subject_
                     x_test_price_norm,x_test_previous_project_norm,X_test_positive_sentiment_score,
                    X_test_negative_sentiment_score,X_test_neutral_sentiment_score,\
                        x_test_total_prob_score ))

#Cross Validation Preprocessing Dataset
X_cv_final_data = np.hstack((x_cv_essay_tfidf_w2vec,x_cv_project_tfidf_w2vec, x_cv_test_prefix_category, X_cv_pro
                x_cv_student_state_prefix_category, x_cv_project_subject_category,x_cv_project_subject_sub_categor
                  x_cv_price_norm,x_cv_previous_project_norm,X_cv_positive_sentiment_score,
                         X_cv_negative_sentiment_score,X_cv_neutral_sentiment_score,\
                    x_cv_total_prob_score ))

print("="*100)
print("Ater Encoding and Transforming Numerical and Categorical Features")
print(X_train_final_data.shape, Y_train_final.shape)
print(X_test_final_data.shape, Y_test.shape)
print(X_cv_final_data.shape, Y_cv.shape)
```

```
====================================================================================================
Ater Encoding and Transforming Numerical and Categorical Features
(32000, 616) (32000,)
(10000, 616) (10000,)
(8000, 616) (8000,)
```

In [47]:

```
import lightgbm as lgb
train_score = []
crossValidation_Score = []
train_auc,cv_auc = [],[]
#hyper param tuning
n_estimators=[10,20,30,50,60,70,80,90,100,150,200]
max_depth = [2,3,4,5,7,10]
#iterate the estimator and iterate the learning rate
for estimator in tqdm(n_estimators):
    for depth in max_depth:
        #get the LGBM Classifier
        treetfidf = lgb.LGBMClassifier(learning_rate=0.1,n_estimators = estimator,max_depth = depth)
        #fit the model
        treetfidf.fit(X_train_final_data, Y_train_final)

        y_train_pred = batchPredict(treetfidf, X_train_final_data)
        y_cv_pred = batchPredict(treetfidf, X_cv_final_data)

        train_auc.append(roc_auc_score(Y_train_final,y_train_pred))
        cv_auc.append(roc_auc_score(Y_cv, y_cv_pred))
n_estimator_x = []
#this will be used for plotig the graph,because we don't use grid search and random search CV
max_depth_y = max_depth * len(n_estimators)
for est in n_estimators:
    for depth in max_depth:
        n_estimator_x.append(est)
```

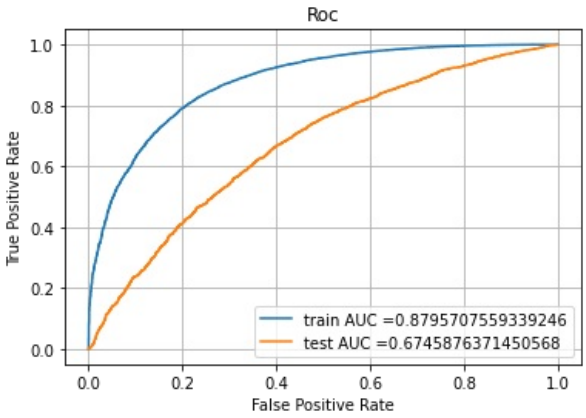
```
100%|██████████████████████████████████████████████████████████████████████████████| 11/11 [09:57<00:00, 54.3
6s/it]
```

In [48]:

```
trainPlot = go.Scatter3d(x=n_estimator_x,y=max_depth_y,z=train_auc, name = 'train AUC')
validationPlot = go.Scatter3d(x=n_estimator_x,y=max_depth_y,z=cv_auc, name = 'Cross validation AUC')
data = [trainPlot, validationPlot]

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

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

plt.show()
```

In [52]:

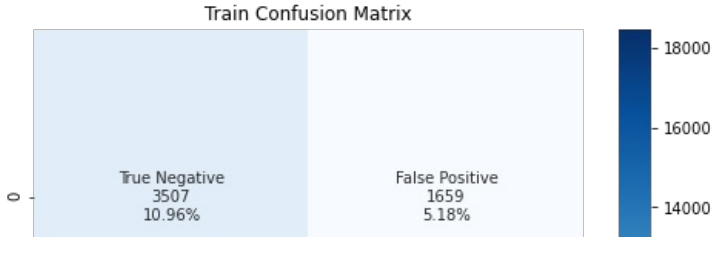
```
max_depth = 4
best_estimator = 150
tree_gbdt = lgb.LGBMClassifier(max_depth = max_depth,n_estimators = best_estimator )
tree_gbdt.fit(X_train_final_data,Y_train_final)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

y_train_pred = batchPredict(tree_gbdt, X_train_final_data)
y_test_pred = batchPredict(tree_gbdt, X_test_final_data)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train_final, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Roc")
plt.grid()
plt.show()
```



In [50]:

```
best_threshold = find_best_threshold(tr_thresholds, train_fpr, train_tpr)


y_train_prediction = predict_with_best_threshold(y_train_pred,best_threshold)

train_confusion_matrix = confusion_matrix(Y_train_final,y_train_prediction)
confusionMatrixPlot(train_confusion_matrix,"Train Confusion Matrix")
```

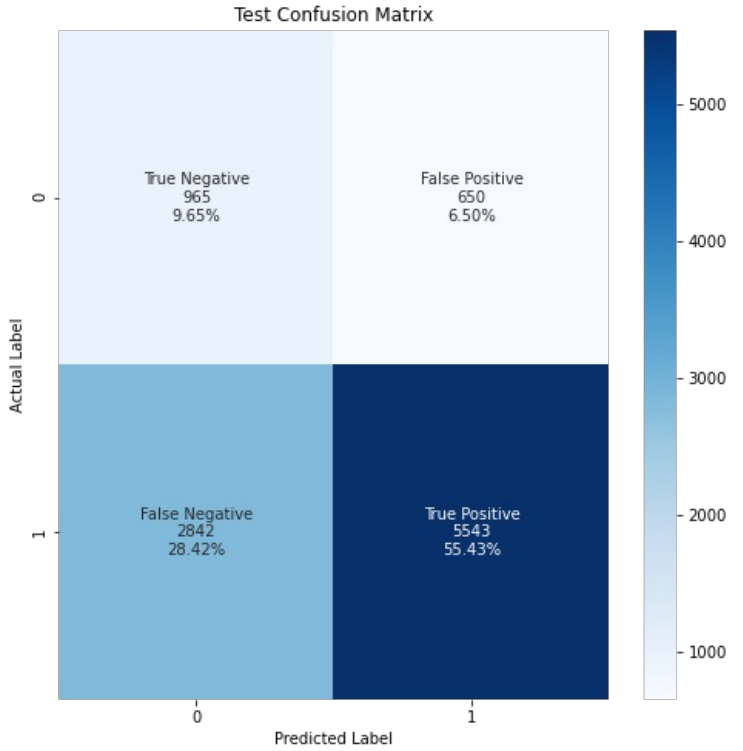
```
The maximum value of tpr*(1-fpr) 0.46691038126003226 for threshold 0.826
```



In [51]:

```
y_test_prediction = predict_with_best_threshold(y_test_pred, best_threshold)
test_confusion_matrix = confusion_matrix(Y_test, y_test_prediction)
print("Confusion Matrix For Test Data")
print("="* 100)
confusionMatrixPlot(test_confusion_matrix,"Test Confusion Matrix")
```

```
Confusion Matrix For Test Data
====================================================================================================
```



3. Conclusion

as mentioned in the step 4 of instructions

In [55]:

```
#https://www.geeksforgeeks.org/creating-tables-with-prettytable-library-python/
from prettytable import PrettyTable

# Specify the Column Names while initializing the Table
table = PrettyTable(["Vectorizer", "Model", "Hyper Parameter","Test AUC"])

# Add rows
table.add_row(["Response Encoding + TFIDF", "GBDT Classifier", "max_depth =2 ,n_estimator = 200", "0.68"])
table.add_row(["Response Encoding + TFIDF W2vec", "GBDT  Classifier", "max_depth = 4,n_estimator = 150 ", "67.5"]
```

```
print(table)
```

```
+---------------------------------+------------------+---------------------------------+----------+
|            Vectorizer           |      Model       |         Hyper Parameter         | Test AUC |
+---------------------------------+------------------+---------------------------------+----------+
|    Response Encoding + TFIDF    | GBDT Classifier  | max_depth =2 ,n_estimator = 200 |   0.68   |
| Response Encoding + TFIDF W2vec | GBDT  Classifier | max_depth = 4,n_estimator = 150 |   67.5   |
+---------------------------------+------------------+---------------------------------+----------+
```