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

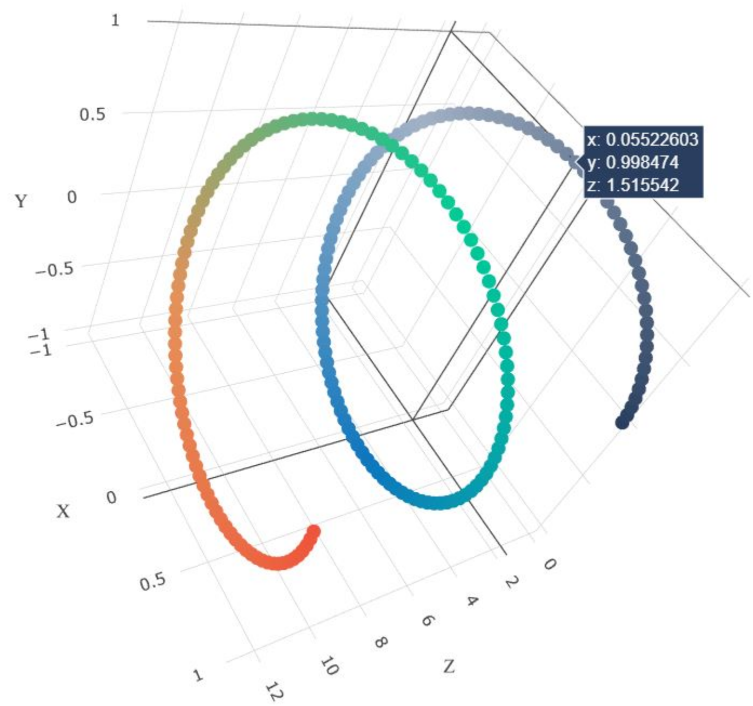
State_0	State_1
3/5	2/5
1/3	2/3
1/2	1/2
1/3	2/3
0/2	2/2
1/2	1/2

> 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

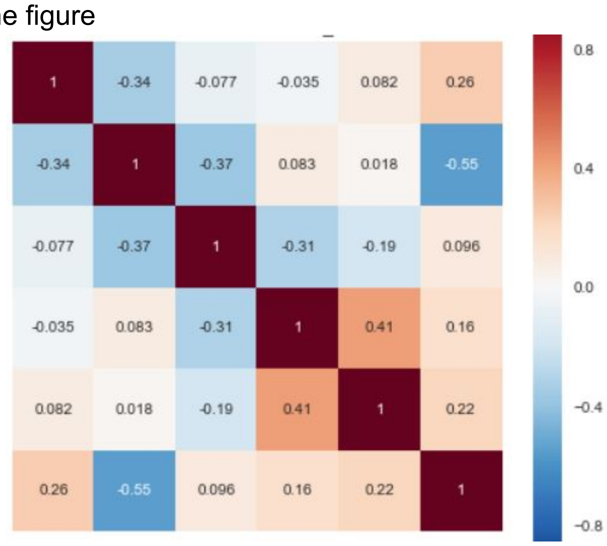
2. **The hyper paramter tuning (Consider any two hyper parameters)**
    - Find the best hyper parameter which will give the maximum [AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
    - find the best hyper paramter using k-fold cross validation/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



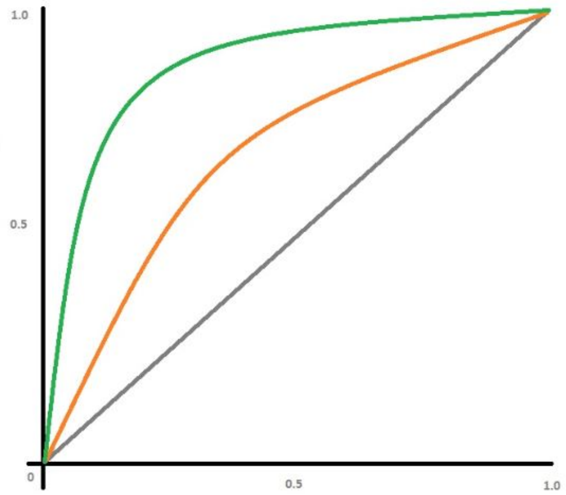
with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

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 (https://seaborn.pydata.org/generated/seaborn.heatmap.html)](https://seaborn.pydata.org/generated/seaborn.heatmap.html) 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

	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]:

```
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 stu
for learning my students learn in many different ways using all of our senses and multiple
of techniques to help all my students succeed students in my class come from a variety of d
for wonderful sharing of experiences and cultures including native americans our school is
learners which can be seen through collaborative student project based learning in and out
in my class love to work with hands on materials and have many different opportunities to p
mastered having the social skills to work cooperatively with friends is a crucial aspect of
montana is the perfect place to learn about agriculture and nutrition my students love to r
in the early childhood classroom i have had several kids ask me can we try cooking with rea
and create common core cooking lessons where we learn important math and writing concepts w
food for snack time my students will have a grounded appreciation for the work that went in
of where the ingredients came from as well as how it is healthy for their bodies this proje
nutrition and agricultural cooking recipes by having us peel our own apples to make homemad
and mix up healthy plants from our classroom garden in the spring we will also create our o
shared with families students will gain math and literature skills as well as a life long e
nannan'
ss = sid.polarity_scores(for_sentiment)

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


for k in ss:
    print('{0}, '.format(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, 0.01, 0.745, 0.245, 0.9
975,

[nltk_data] Downloading package vader_lexicon to C:\Users\My
[nltk_data]     Computer\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

1. GBDT (xgboost/lightgbm)

In [2]:

```
#Importing libraries

import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
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
import prettytable
from prettytable import PrettyTable
from tqdm import tqdm_notebook as tqdm
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/

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()
from collections import Counter
```

1.1 Loading Data

In [3]:

```
data  = pd.read_csv('preprocessed_data.csv',nrows=60000)
data.head(5)
```

Out[3]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_pr
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	
2	ca	mrs	grades_prek_2	
3	ga	mrs	grades_prek_2	
4	wa	mrs	grades_3_5	

In [4]:

```
data.columns
```

Out[4]:

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       'clean_categories', 'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

In [5]:

```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[5]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_pr
0	ca	mrs	grades_prek_2	

1.2 Splitting data into Train and cross validation(or test): Stratified Sampling

In [6]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.30, stratify=
```

1.3 Make Data Model Ready: encoding eassay

1.3.1 TFIDF featurization of eassy feature

In [7]:

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)
text_tfidf = vectorizer.fit(X_train['essay'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf= vectorizer.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
essay_features_tfidf = vectorizer.get_feature_names()
```

```
(29400, 8) (29400,)
(12600, 8) (12600,)
(18000, 8) (18000,)
============================================================================
========================
After vectorizations
(29400, 5000) (29400,)
(12600, 5000) (12600,)
(18000, 5000) (18000,)
============================================================================
========================
```

1.3.2 TFIDF W2V featurization of eassy feature

In [8]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickl
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

tfidf_w2v_vectors for Train data

In [9]:

```
tfidf_model = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=50000)
tfidf_model.fit(X_train['essay'])
tfidf_model.transform(X_train['essay'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [10]:

```
tfidf_w2v_vectors_X_train_essay = []; # the avg-w2v for each sentence/review is stored in t
for sentence in tqdm(X_train['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_X_train_essay.append(vector)

print(len(tfidf_w2v_vectors_X_train_essay))
print(len(tfidf_w2v_vectors_X_train_essay[0]))
```

```
100%|██████████| 29400/29400 [03:20<00:00, 146.72it/s]

29400
300
```

tfidf_w2v_vectors for CV data

In [11]:

```
#tfidf_model = TfidfVectorizer()
tfidf_model.transform(X_cv['essay'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [12]:

```
tfidf_w2v_vectors_X_cv_essay = []; # the avg-w2v for each sentence/review is stored in this
for sentence in tqdm(X_cv['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_X_cv_essay.append(vector)

print(len(tfidf_w2v_vectors_X_cv_essay))
print(len(tfidf_w2v_vectors_X_cv_essay[0]))
```

```
100%|██████████| 12600/12600 [01:23<00:00, 150.99it/s]

12600
300
```

tfidf_w2v_vectors For Test data

In [13]:

```
#tfidf_model = TfidfVectorizer()
tfidf_model.transform(X_test['essay'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [14]:

```
tfidf_w2v_vectors_X_test_essay = []; # the avg-w2v for each sentence/review is stored in th
for sentence in tqdm(X_test['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_X_test_essay.append(vector)

print(len(tfidf_w2v_vectors_X_test_essay))
print(len(tfidf_w2v_vectors_X_test_essay[0]))
```

```
100%|██████████| 18000/18000 [02:00<00:00, 148.98it/s]

18000
300
```

In [15]:

```
tfidf_w2v_vectors_X_train_essay = np.array(tfidf_w2v_vectors_X_train_essay)
tfidf_w2v_vectors_X_cv_essay=np.array(tfidf_w2v_vectors_X_cv_essay)
tfidf_w2v_vectors_X_test_essay = np.array(tfidf_w2v_vectors_X_test_essay)
print("After vectorizations")
print(tfidf_w2v_vectors_X_train_essay.shape, y_train.shape)
print(tfidf_w2v_vectors_X_cv_essay.shape, y_cv.shape)
print(tfidf_w2v_vectors_X_test_essay.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(29400, 300) (29400,)
(12600, 300) (12600,)
(18000, 300) (18000,)
============================================================================
========================
```

In [16]:

```
# Converting dense matrix to sparase matrix

import numpy as np
from scipy.sparse import csr_matrix


tfidf_w2v_vectors_X_train_essay = csr_matrix(tfidf_w2v_vectors_X_train_essay)
tfidf_w2v_vectors_X_cv_essay = csr_matrix(tfidf_w2v_vectors_X_cv_essay)
tfidf_w2v_vectors_X_test_essay= csr_matrix(tfidf_w2v_vectors_X_test_essay)

print(tfidf_w2v_vectors_X_train_essay.shape, y_train.shape)
print(tfidf_w2v_vectors_X_cv_essay.shape, y_cv.shape)
print(tfidf_w2v_vectors_X_test_essay.shape, y_test.shape)
print("="*100)
```

```
(29400, 300) (29400,)
(12600, 300) (12600,)
(18000, 300) (18000,)
============================================================================
========================
```

1.4 Make Data Model Ready: encoding numerical, categorical features

encoding categorical features ¶

Response encoding fit and trasform functions

In [17]:

```
def response_encoding_fit(category_list, X1,y1):
    #create array for binary label w.r.t to categroical feature school_state
    positive_class_count=np.zeros(len(category_list))
    negative_class_count=np.zeros(len(category_list))

    #create response table
    for j in  tqdm(range(len(X1))):
        for i in range(len(category_list)):
            if X1[j] == category_list[i] and y1[j] == 1:
                positive_class_count[i] += 1
            if X1[j] == category_list[i] and y1[j] == 0:
                negative_class_count[i] += 1
    # Final probability scores
    final = np.zeros((len(category_list),2))
    for i in tqdm(range(len(category_list))):
        final[i][0] = negative_class_count[i] / (negative_class_count[i]+positive_class_cou
        final[i][1] = positive_class_count[i] / (negative_class_count[i]+positive_class_cou


    res = dict()
    for i in range(len(category_list)):
        res[category_list[i]] = [final[i][0], final[i][1]]
    return res

def response_encoding_transform(vectorizer,X1,y1):
    X_ = []
    # Loop each data points in X
    for i in tqdm(range(len(X1))):         # To check whether this feature in X contain in
        if X1[i] in vectorizer:            # If it is present, call the dict() as a key val
            X_.append(vectorizer[X1[i]])
        else:             # If not, set default value as [0.5,0.5]
              X_.append([0.5,0.5])
    return np.array(X_)
```

1.4.1 encoding categorical features: School State

In [18]:

```
category_list = list(X_train['school_state'].unique())
x_train_school_state_res_fit = response_encoding_fit(category_list,X_train['school_state'].

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_res = response_encoding_transform(x_train_school_state_res_fit,X_train['schoo
X_cv_state_res = response_encoding_transform(x_train_school_state_res_fit,X_cv['school_stat
X_test_state_res = response_encoding_transform(x_train_school_state_res_fit,X_test['school_


print("After vectorizations")
print(X_train_state_res.shape, y_train.shape)
print(X_cv_state_res.shape, y_cv.shape)
print(X_test_state_res.shape, y_test.shape)
print("="*100)
```

```
100%|██████████| 29400/29400 [00:01<00:00, 17734.62it/s]
100%|██████████| 51/51 [00:00<00:00, 9106.79it/s]
100%|██████████| 29400/29400 [00:00<00:00, 350745.19it/s]
100%|██████████| 12600/12600 [00:00<00:00, 279587.72it/s]
100%|██████████| 18000/18000 [00:00<00:00, 351160.83it/s]

After vectorizations
(29400, 2) (29400,)
(12600, 2) (12600,)
(18000, 2) (18000,)
============================================================================
========================
```

1.4.2 encoding categorical features: teacher_prefix

In [19]:

```
category_list = list(X_train['teacher_prefix'].unique())
x_train_school_state_res_fit = response_encoding_fit(category_list,X_train['teacher_prefix'


# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_res = response_encoding_transform(x_train_school_state_res_fit,X_train['tea
X_cv_teacher_res = response_encoding_transform(x_train_school_state_res_fit,X_cv['teacher_p
X_test_teacher_res = response_encoding_transform(x_train_school_state_res_fit,X_test['teach

print("After vectorizations")
print(X_train_teacher_res.shape, y_train.shape)
print(X_cv_teacher_res.shape, y_cv.shape)
print(X_test_teacher_res.shape, y_test.shape)
```

```
100%|██████████| 29400/29400 [00:00<00:00, 76771.17it/s]
100%|██████████| 5/5 [00:00<00:00, 2993.37it/s]
100%|██████████| 29400/29400 [00:00<00:00, 389492.47it/s]
100%|██████████| 12600/12600 [00:00<00:00, 417067.02it/s]
100%|██████████| 18000/18000 [00:00<00:00, 311778.49it/s]

After vectorizations
(29400, 2) (29400,)
(12600, 2) (12600,)
(18000, 2) (18000,)
```

1.4.3 encoding categorical features: project_grade_category

In [20]:

```
category_list = list(X_train['project_grade_category'].unique())
x_train_school_state_res_fit = response_encoding_fit(category_list,X_train['project_grade_c


# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_res = response_encoding_transform(x_train_school_state_res_fit,X_train['proje
X_cv_grade_res = response_encoding_transform(x_train_school_state_res_fit,X_cv['project_gra
X_test_grade_res = response_encoding_transform(x_train_school_state_res_fit,X_test['project

print("After vectorizations")
print(X_train_grade_res.shape, y_train.shape)
print(X_cv_grade_res.shape, y_cv.shape)
print(X_test_grade_res.shape, y_test.shape)
```

```
100%|██████████| 29400/29400 [00:00<00:00, 100350.04it/s]
100%|██████████| 4/4 [00:00<00:00, 1076.01it/s]
100%|██████████| 29400/29400 [00:00<00:00, 457636.85it/s]
100%|██████████| 12600/12600 [00:00<00:00, 497976.28it/s]
100%|██████████| 18000/18000 [00:00<00:00, 342102.04it/s]

After vectorizations
(29400, 2) (29400,)
(12600, 2) (12600,)
(18000, 2) (18000,)
```

1.4.4 encoding categorical features: clean_categories

In [21]:

```
category_list = list(X_train['clean_categories'].unique())
x_train_school_state_res_fit = response_encoding_fit(category_list,X_train['clean_categorie


# we use the fitted CountVectorizer to convert the text to vector
X_train_categories_res = response_encoding_transform(x_train_school_state_res_fit,X_train['
X_cv_categories_res = response_encoding_transform(x_train_school_state_res_fit,X_cv['clean_
X_test_categories_res = response_encoding_transform(x_train_school_state_res_fit,X_test['cl

print("After vectorizations")
print(X_train_categories_res.shape, y_train.shape)
print(X_cv_categories_res.shape, y_cv.shape)
print(X_test_categories_res.shape, y_test.shape)
```

```
100%|██████████| 29400/29400 [00:01<00:00, 27571.90it/s]
100%|██████████| 44/44 [00:00<00:00, 7057.07it/s]
100%|██████████| 29400/29400 [00:00<00:00, 479006.42it/s]
100%|██████████| 12600/12600 [00:00<00:00, 393878.33it/s]
100%|██████████| 18000/18000 [00:00<00:00, 328795.97it/s]

After vectorizations
(29400, 2) (29400,)
(12600, 2) (12600,)
(18000, 2) (18000,)
```

1.4.5 encoding categorical features: clean_subcategories

In [22]:

```
category_list = list(X_train['clean_subcategories'].unique())
x_train_school_state_res_fit = response_encoding_fit(category_list,X_train['clean_subcatego


# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategories_res = response_encoding_transform(x_train_school_state_res_fit,X_trai
X_cv_subcategories_res = response_encoding_transform(x_train_school_state_res_fit,X_cv['cle
X_test_subcategories_res = response_encoding_transform(x_train_school_state_res_fit,X_test[

print("After vectorizations")
print(X_train_subcategories_res.shape, y_train.shape)
print(X_cv_subcategories_res.shape, y_cv.shape)
print(X_test_subcategories_res.shape, y_test.shape)
```

```
100%|██████████| 29400/29400 [00:07<00:00, 3818.33it/s]
100%|██████████| 338/338 [00:00<00:00, 39670.77it/s]
100%|██████████| 29400/29400 [00:00<00:00, 206411.56it/s]
100%|██████████| 12600/12600 [00:00<00:00, 326764.22it/s]
100%|██████████| 18000/18000 [00:00<00:00, 263475.10it/s]

After vectorizations
(29400, 2) (29400,)
(12600, 2) (12600,)
(18000, 2) (18000,)
```

encoding numerical features

1.4.6 encoding numerical features: Price

In [23]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(1,-1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))#Normalize(
X_train_price_norm = X_train_price_norm.reshape(-1,1)
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_cv_price_norm = X_cv_price_norm.reshape(-1,1)
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))
X_test_price_norm = X_test_price_norm.reshape(-1,1)



print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(29400, 1) (29400,)
(12600, 1) (12600,)
(18000, 1) (18000,)
============================================================================
========================
```

1.4.7 encoding numerical features: teacher_number_of_previously_posted_projects

In [24]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)

X_train_projects_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_p
X_train_projects_norm = X_train_projects_norm.reshape(-1,1)
X_cv_projects_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_project
X_cv_projects_norm = X_cv_projects_norm.reshape(-1,1)
X_test_projects_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_pro
X_test_projects_norm = X_test_projects_norm.reshape(-1,1)



print("After vectorizations")
print(X_train_projects_norm.shape, y_train.shape)
print(X_cv_projects_norm.shape, y_cv.shape)
print(X_test_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(29400, 1) (29400,)
(12600, 1) (12600,)
(18000, 1) (18000,)
============================================================================
========================
```

1.4.8 Sentiment featurization for essay feature

In [25]:

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

# import nltk
nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

def sentiment_scores(X):
        """
        This function will give sentiment score of the given array of Texts

        """
        Essay_Sentiment_Fetures = []

        for j in  tqdm(range(len(X))):

            ss = sid.polarity_scores(X[j])
            fe = []
            for k in ss:
                fe.append(ss[k])
            Essay_Sentiment_Fetures.append(fe)
            Essay_Sentiment_Fetures1 = np.array(Essay_Sentiment_Fetures)

        return Essay_Sentiment_Fetures1
```

```
[nltk_data] Downloading package vader_lexicon to C:\Users\My
[nltk_data]     Computer\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [26]:

```
essay_train = list(X_train['essay'].values)
essay_cv = list(X_cv['essay'].values)
essay_test = list(X_test['essay'].values)


# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_sen = np.array(sentiment_scores(essay_train))
X_cv_essay_sen = np.array(sentiment_scores(essay_cv))
X_test_essay_sen = np.array(sentiment_scores(essay_test))

print("After vectorizations")
print(X_train_essay_sen.shape, y_train.shape)
print(X_cv_essay_sen.shape, y_cv.shape)
print(X_test_essay_sen.shape, y_test.shape)
```

```
100%|██████████| 29400/29400 [11:38<00:00, 42.07it/s]
100%|██████████| 12600/12600 [02:45<00:00, 75.96it/s]
100%|██████████| 18000/18000 [04:43<00:00, 63.50it/s]

After vectorizations
(29400, 4) (29400,)
(12600, 4) (12600,)
(18000, 4) (18000,)
```

1.4.9 Concatinating all the features

Set1: Features:categorical, numerical features + eassay (tfidf vec)

In [27]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_tfidf = hstack((X_train_essay_tfidf, X_train_state_res, X_train_teacher_res, X_train_g
X_cr_tfidf = hstack((X_cv_essay_tfidf, X_cv_state_res, X_cv_teacher_res, X_cv_grade_res, X_
X_te_tfidf = hstack((X_test_essay_tfidf, X_test_state_res, X_test_teacher_res, X_test_grade

print("Final Data matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_cr_tfidf.shape, y_cv.shape)
print(X_te_tfidf.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(29400, 5016) (29400,)
(12600, 5016) (12600,)
(18000, 5016) (18000,)
============================================================================
========================
```

Set2 : Features:categorical, numerical features + eassay (TFIDF W2V)

In [28]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_tfidf_W2V = hstack((tfidf_w2v_vectors_X_train_essay,X_train_state_res, X_train_teacher
X_cr_tfidf_W2V = hstack((tfidf_w2v_vectors_X_cv_essay,X_cv_state_res, X_cv_teacher_res, X_c
X_te_tfidf_W2V = hstack((tfidf_w2v_vectors_X_test_essay,X_test_state_res, X_test_teacher_re

print("Final Data matrix")
print(X_tr_tfidf_W2V.shape, y_train.shape)
print(X_cr_tfidf_W2V.shape, y_cv.shape)
print(X_te_tfidf_W2V.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(29400, 316) (29400,)
(12600, 316) (12600,)
(18000, 316) (18000,)
============================================================================
========================
```

In [ ]:

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

1.5.1 Modeling of set1 features

Hyperparameter Tuning

In [29]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 4900
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [30]:

```
import matplotlib.pyplot as plt
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
n_estimators_plot =[]
max_depth_plot = []
n_estimators=[25,50,100,150,200]
max_depth =[2,3,4,5,6,8]
for i in tqdm(n_estimators):
    for j in max_depth:
        n_estimators_plot.append(i)
        max_depth_plot.append(j)
        clf=XGBClassifier(random_state=0,n_estimators=i,max_depth=j)
        clf.fit(X_tr_tfidf, y_train)

        y_train_pred = batch_predict(clf, X_tr_tfidf)
        y_cv_pred = batch_predict(clf, X_cr_tfidf)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
100%|██████████| 5/5 [57:28<00:00, 689.67s/it]
```

In [31]:

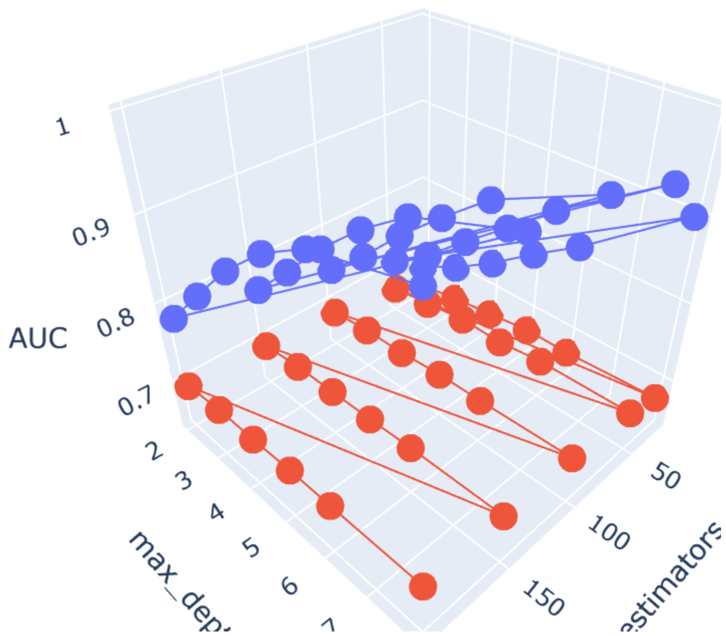
```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [32]:

```
x1 = n_estimators_plot
y1 = max_depth_plot
z1 = train_auc

x2 = n_estimators_plot
y2 = max_depth_plot
z2 = cv_auc
```

In [33]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'Cross validation')
data = [trace1, trace2]

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

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

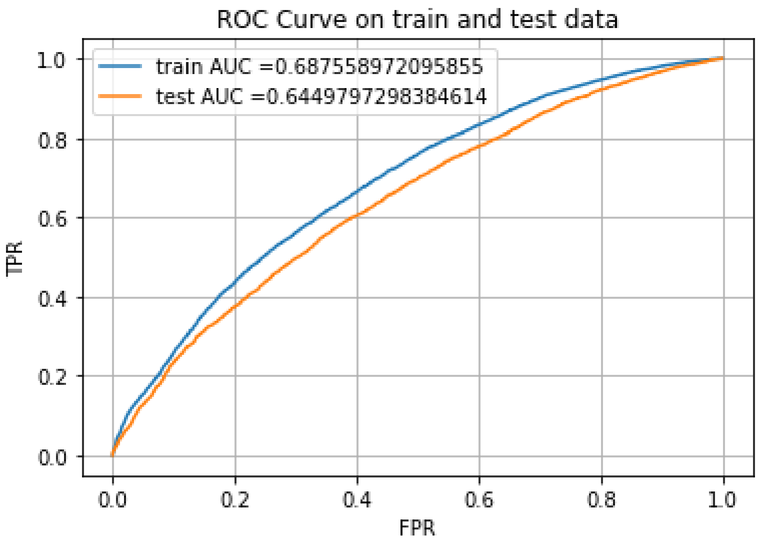


In [34]:

```
# from the error plot we choose best Hyperparameters such that, we will have maximum AUC on

#here we are choosing the best Hyperparameters based on forloop results
n_estimators_best = 25
max_depth_best = 2
```

In [35]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.
from sklearn.metrics import roc_curve, auc
from sklearn.ensemble import GradientBoostingClassifier

clf=GradientBoostingClassifier(random_state=0,n_estimators=n_estimators_best,max_depth=max_
clf.fit(X_tr_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = batch_predict(clf, X_tr_tfidf)
y_test_pred = batch_predict(clf, X_te_tfidf)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, 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("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve on train and test data")
plt.grid()
plt.show()
```



1.5.2 Ploting confusion matrix for set1 features

In [36]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[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
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [37]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
============================================================================
========================
the maximum value of tpr*(1-fpr) 0.40080606969582483 for threshold 0.842
Train confusion matrix
[[ 3033  1639]
 [ 9461 15267]]
Test confusion matrix
[[1891  970]
 [6893 8246]]
```

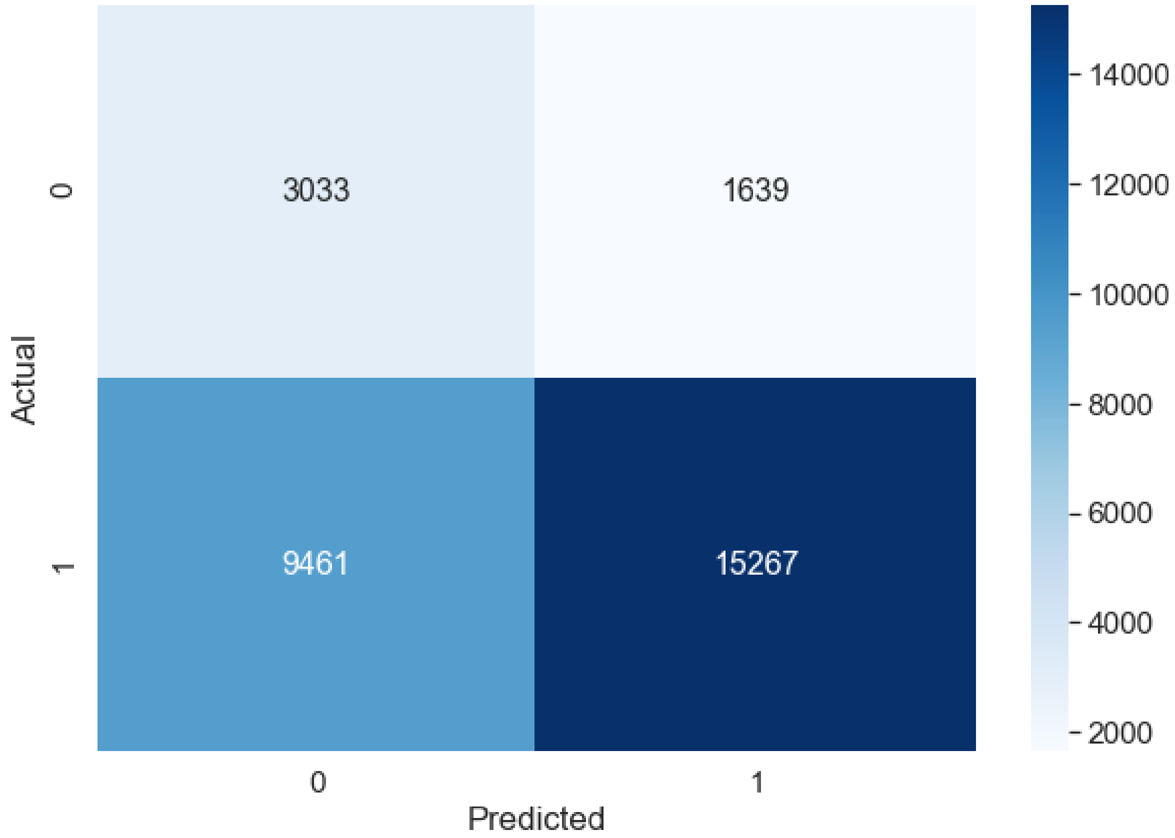
In [38]:

```
from sklearn.metrics import confusion_matrix
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

print("Train confusion matrix")
data = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
df_cm = pd.DataFrame(data, columns=np.unique(y_test), index = np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16},fmt='g')# font size
```

```
Train confusion matrix
```

Out[38]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e3dd8d3f48>
```



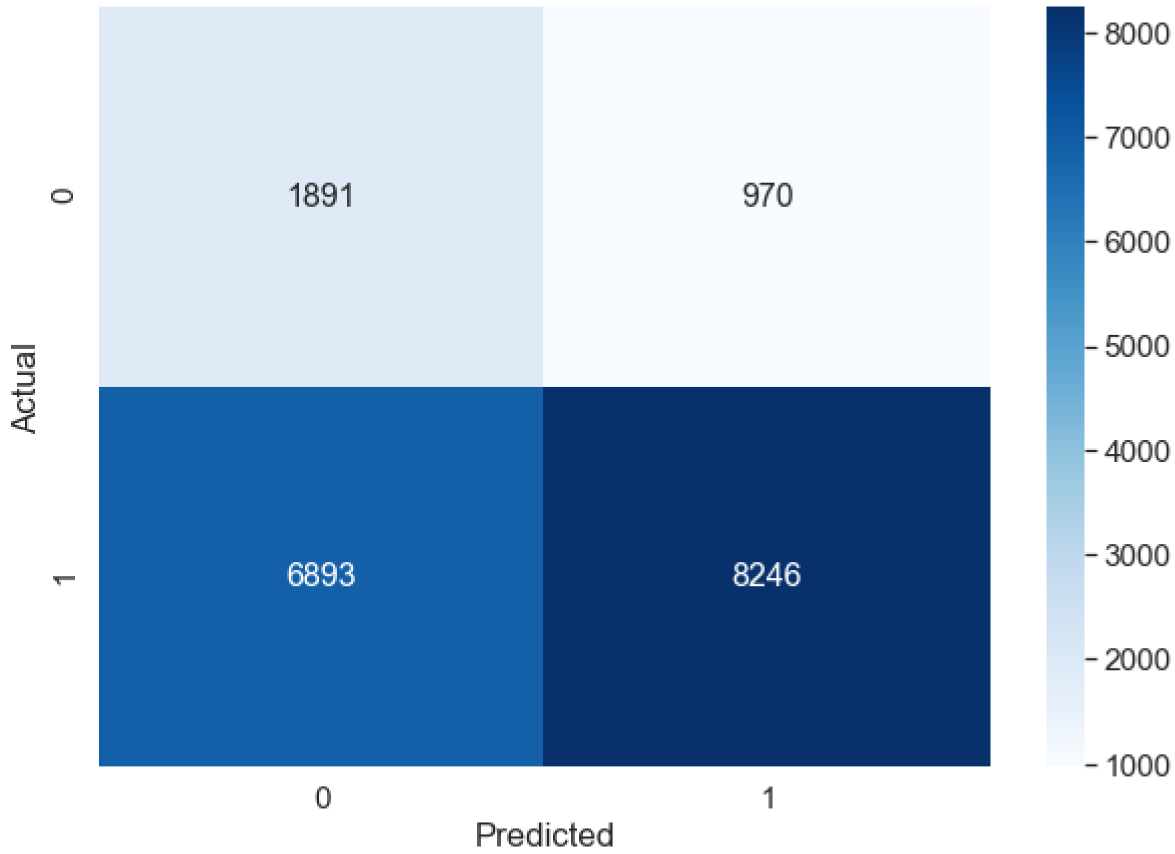
In [39]:

```
print("Test confusion matrix")
data = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
df_cm = pd.DataFrame(data, columns=np.unique(y_test), index = np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
y_pred_test = predict_with_best_t(y_test_pred, best_t)
plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16},fmt='g')# font size
```

```
Test confusion matrix
```

Out[39]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e3df3e57c8>
```



1.5.3 Modeling of set2 features

Hyperparameter Tuning

In [40]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 4900
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [41]:

```
import matplotlib.pyplot as plt
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
n_estimators_plot =[]
max_depth_plot = []
n_estimators=[25,50,100,150,200]
max_depth =[2,3,4,5,6,8]
for i in tqdm(n_estimators):
    for j in max_depth:
        n_estimators_plot.append(i)
        max_depth_plot.append(j)
        clf=XGBClassifier(random_state=0,n_estimators=i,max_depth=j)
        clf.fit(X_tr_tfidf_W2V, y_train)

        y_train_pred = batch_predict(clf, X_tr_tfidf_W2V)
        y_cv_pred = batch_predict(clf, X_cr_tfidf_W2V)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
100%|██████████| 5/5 [1:17:19<00:00, 927.84s/it]
```

In [42]:

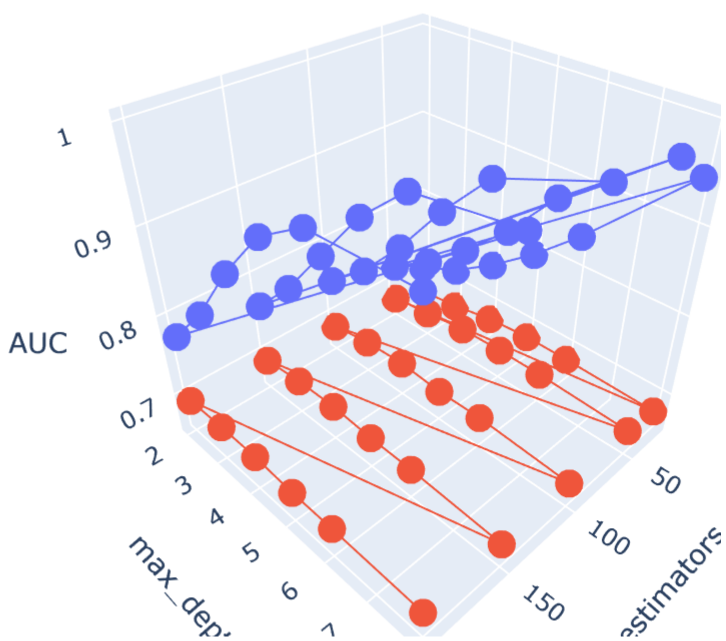
```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [43]:

```
x1 = n_estimators_plot
y1 = max_depth_plot
z1 = train_auc

x2 = n_estimators_plot
y2 = max_depth_plot
z2 = cv_auc
```

In [44]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'Cross validation')
data = [trace1, trace2]

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

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

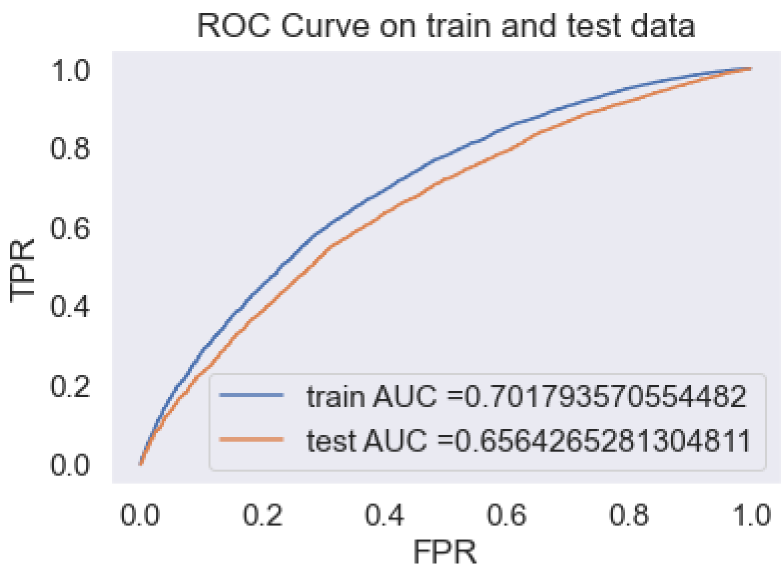


In [45]:

```
# from the error plot we choose best Hyperparameters such that, we will have maximum AUC on

#here we are choosing the best Hyperparameters based on forloop results
n_estimators_best = 25
max_depth_best = 2
```

In [47]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.
from sklearn.metrics import roc_curve, auc
from sklearn.ensemble import GradientBoostingClassifier

clf=GradientBoostingClassifier(random_state=0,n_estimators=n_estimators_best,max_depth=max_
clf.fit(X_tr_tfidf_W2V, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = batch_predict(clf, X_tr_tfidf_W2V)
y_test_pred = batch_predict(clf, X_te_tfidf_W2V)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, 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("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve on train and test data")
plt.grid()
plt.show()
```



1.5.4 Ploting confusion matrix for set2 features

In [48]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[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
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [49]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
============================================================================
========================
the maximum value of tpr*(1-fpr) 0.42127512576558984 for threshold 0.838
Train confusion matrix
[[ 3039  1633]
 [ 8713 16015]]
Test confusion matrix
[[1839 1022]
 [6182 8957]]
```

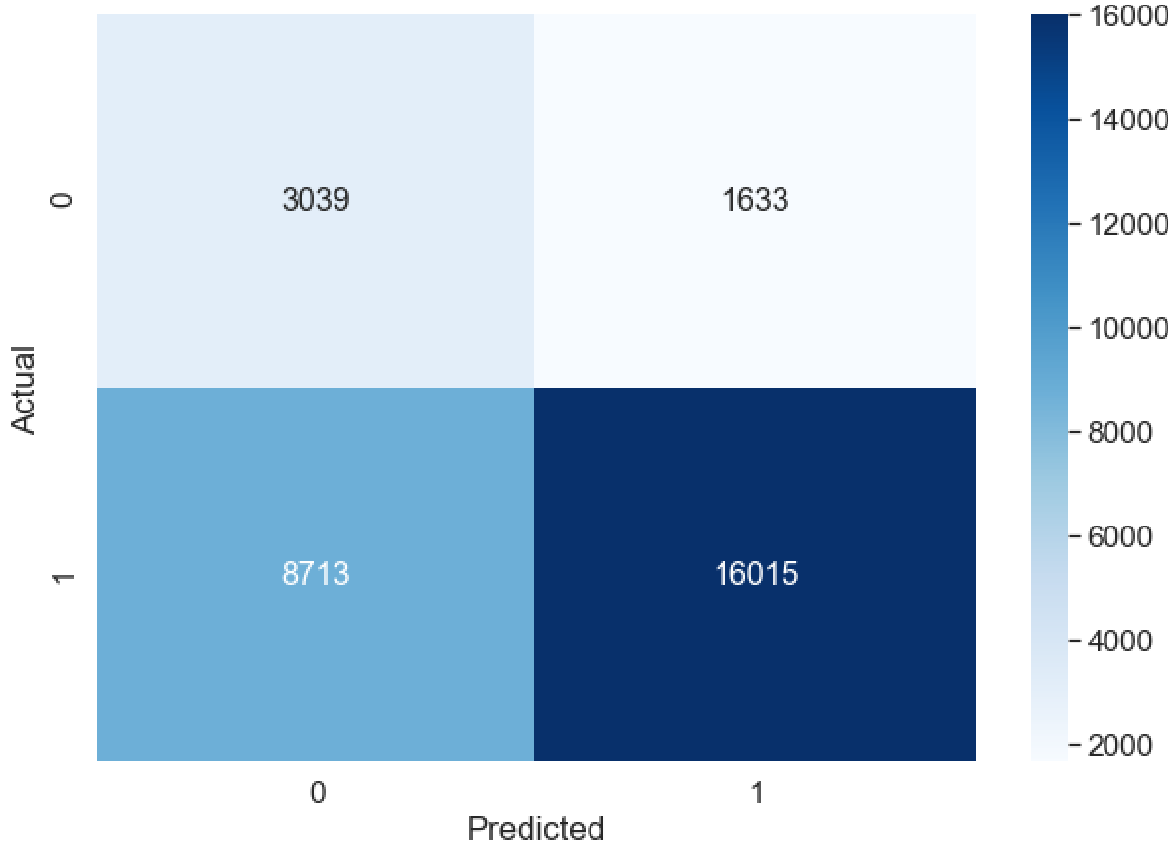
In [50]:

```
from sklearn.metrics import confusion_matrix
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

print("Train confusion matrix")
data = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
df_cm = pd.DataFrame(data, columns=np.unique(y_test), index = np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16},fmt='g')# font size
```

```
Train confusion matrix
```

Out[50]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e3c3deaf48>
```



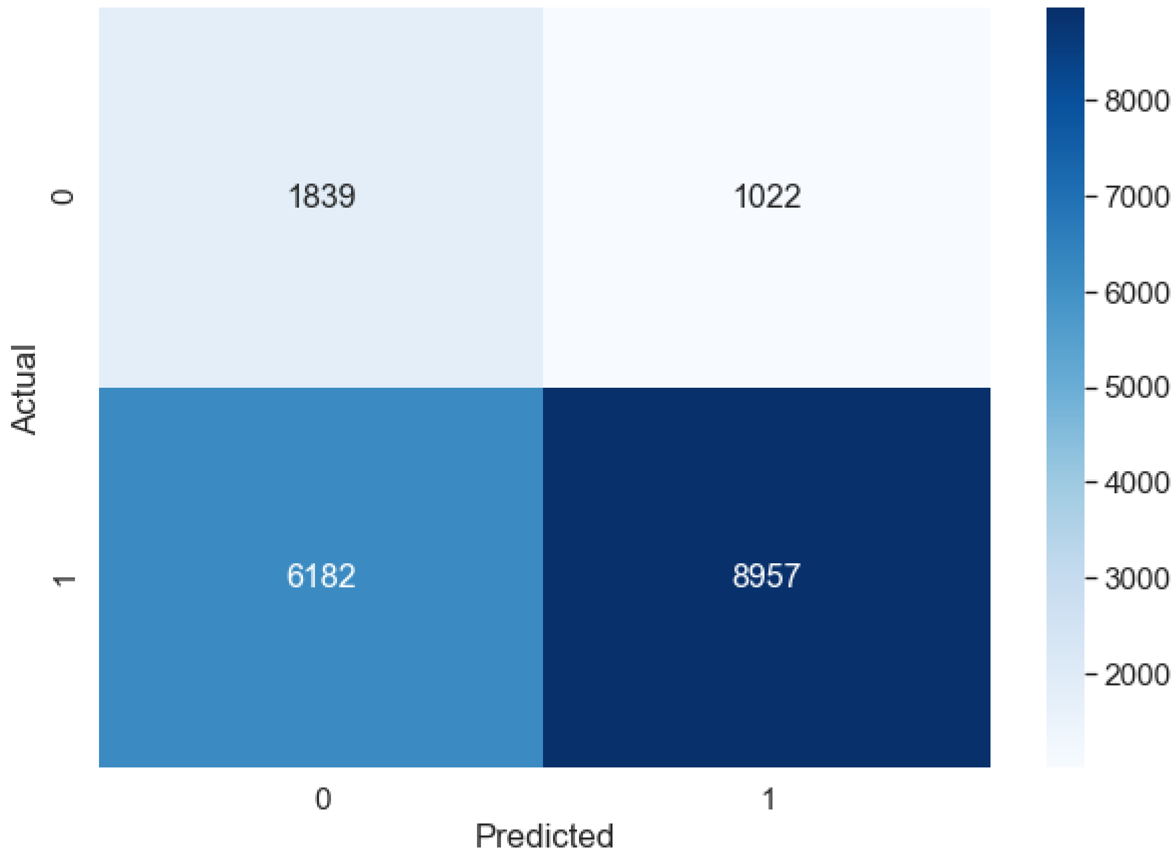
In [51]:

```
print("Test confusion matrix")
data = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
df_cm = pd.DataFrame(data, columns=np.unique(y_test), index = np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
y_pred_test = predict_with_best_t(y_test_pred, best_t)
plt.figure(figsize = (10,7))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16},fmt='g')# font size
```

```
Test confusion matrix
```

Out[51]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e3e55f1088>
```



In [ ]:

3. Summary

In [53]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyperparameter n_estimators","Hyperparameter Max_D

x.add_row(["TFIDF","XGBClassifier", '25','2', '0.68','0.64'])
x.add_row(["TFIDF_W2V", "XGBClassifier", '25','2', '0.70','0.65'])

print(x)
```

```
+------------+---------------+-----------------------------+----------------
----------+-----------+----------+
| Vectorizer |     Model     | Hyperparameter n_estimators | Hyperparameter 
Max_Depth | Train_AUC | Test_AUC |
+------------+---------------+-----------------------------+----------------
----------+-----------+----------+
|   TFIDF    | XGBClassifier |              25             |              2 
|    0.68   |    0.64   |
| TFIDF_W2V  | XGBClassifier |              25             |              2 
|    0.70   |    0.65   |
+------------+---------------+-----------------------------+----------------
----------+-----------+----------+
```

In [ ]:

```
```