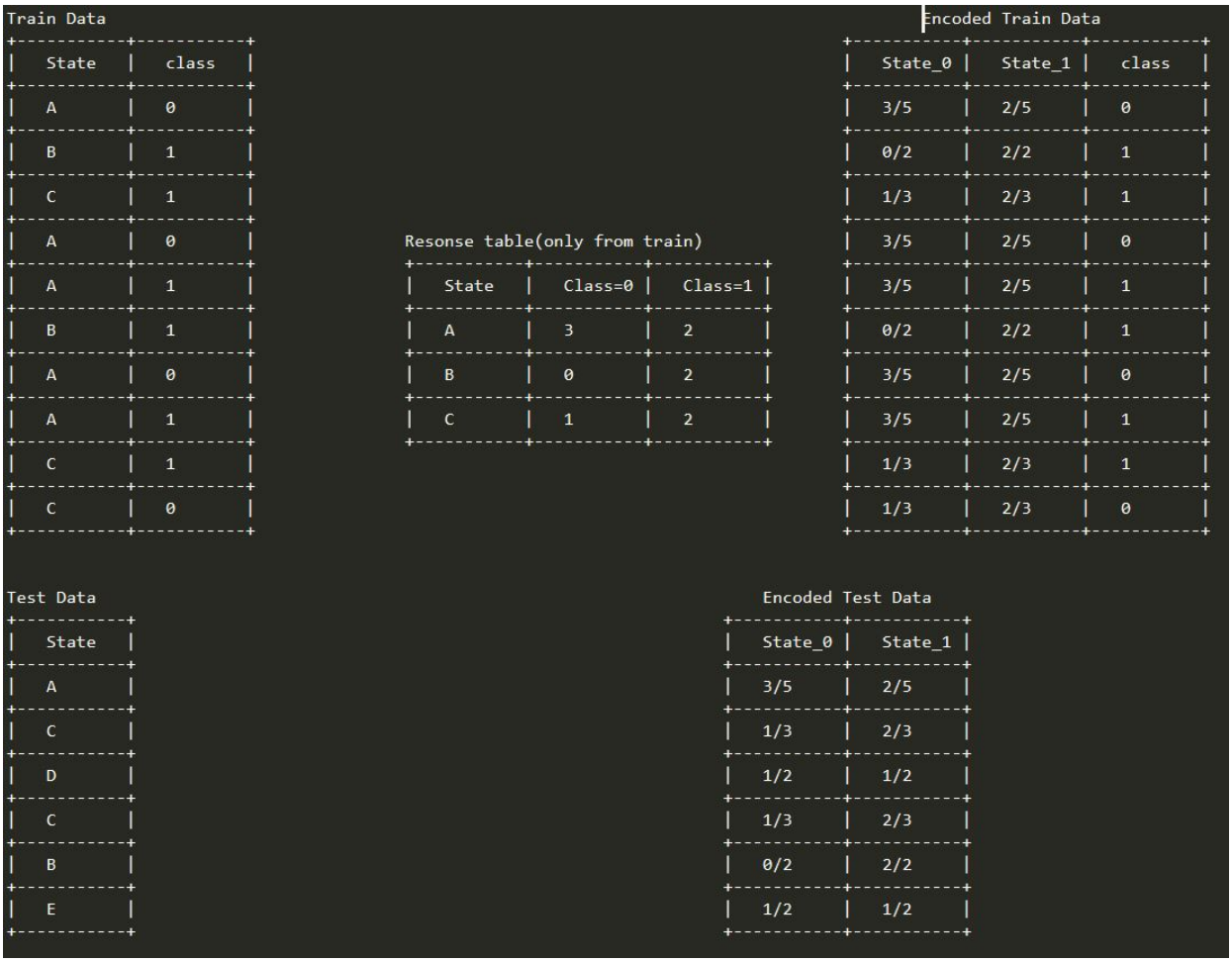
Assignment 9: GBDT

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

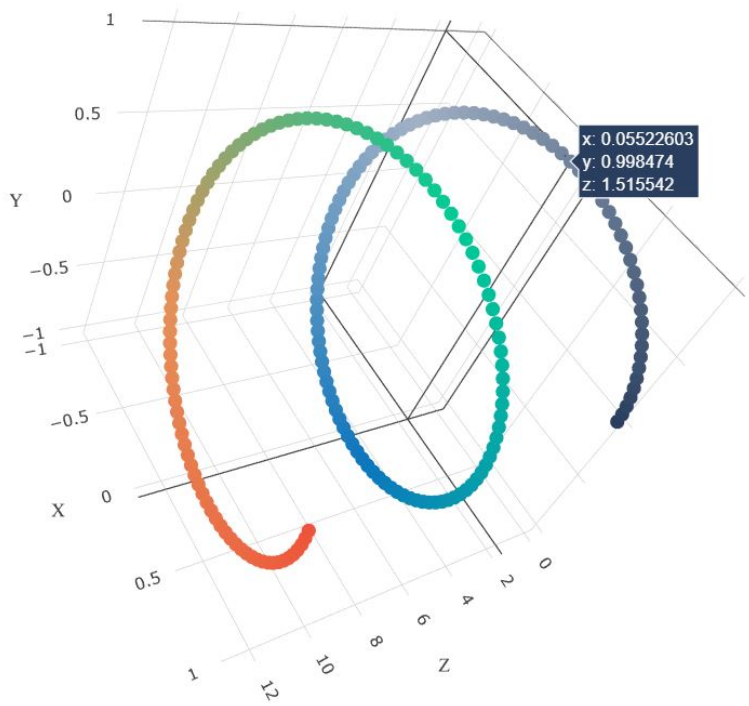
Encoded Test Data

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**

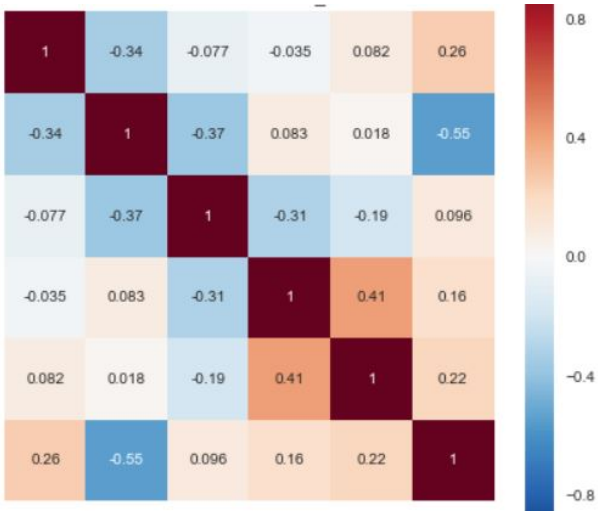
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



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

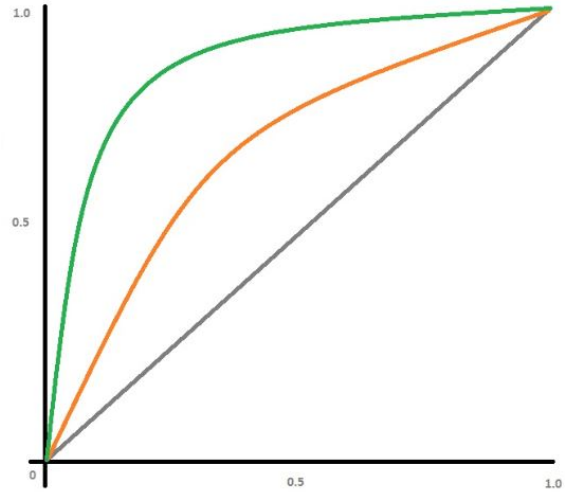
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



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 [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
for learning my students learn in many different ways using all of our senses and multi
of techniques to help all my students succeed students in my class come from a variety
for wonderful sharing of experiences and cultures including native americans our school
learners which can be seen through collaborative student project based learning in and
in my class love to work with hands on materials and have many different opportunities
mastered having the social skills to work cooperatively with friends is a crucial aspec
```

```
montana is the perfect place to learn about agriculture and nutrition my students love
in the early childhood classroom i have had several kids ask me can we try cooking with
and create common core cooking lessons where we learn important math and writing concep
food for snack time my students will have a grounded appreciation for the work that wen
of where the ingredients came from as well as how it is healthy for their bodies this p
nutrition and agricultural cooking recipes by having us peel our own apples to make hom
and mix up healthy plants from our classroom garden in the spring we will also create o
shared with families students will gain math and literature skills as well as a life lo
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,
```

1. GBDT (xgboost/lightgbm)

1.1 Loading Data

In [3]:
```
import pandas
data_td = pandas.read_csv('train_data.csv')
data = pandas.read_csv('preprocessed_data.csv')
```

In [4]:
```
data['project_title'] = data_td['project_title']
```

In [5]:
```
data.head(10)
```

Out[5]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	pr
0	ca	mrs	grades_prek_2	53	
1	ut	ms	grades_3_5	4	
2	ca	mrs	grades_prek_2	10	
3	ga	mrs	grades_prek_2	2	

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	pr
4	wa	mrs	grades_3_5	2	
5	ca	mrs	grades_3_5	6	
6	ca	mrs	grades_3_5	0	
7	ca	ms	grades_3_5	0	
8	ca	ms	grades_prek_2	127	
9	hi	mrs	grades_3_5	41	

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

In [6]:
```
#Libraries
%matplotlib inline

import pandas as pd
import numpy as np
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

from tqdm import tqdm
import os

import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import math
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

In [7]:
```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'],axis =1)
```

In [8]:
```
from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test  = train_test_split( X,y,test_size = 0.33 , stratif
```

1.3 Make Data Model Ready: encoding eassay, and project_title

In [9]:
```
# please write all the code with proper documentation, and proper titles for each subse
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
import pickle
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [10]:
```
#tfidf essay train and test
vectorizer = TfidfVectorizer(min_df = 10)
vectorizer.fit( X_train['essay'].values )
X_train_essay_tfidf = vectorizer.transform( X_train['essay'].values )
X_test_essay_tfidf = vectorizer.transform( X_test['essay'].values )

#tfidf project title train and test
vectorizer = TfidfVectorizer(min_df = 10)
vectorizer.fit( X_train['project_title'].values )
X_train_pt_tfidf = vectorizer.transform( X_train['project_title'].values )
X_test_pt_tfidf = vectorizer.transform( X_test['project_title'].values )
```

In [11]:
```
print(X_train_essay_tfidf.shape)
print(X_train_pt_tfidf.shape)
```

```
print(X_test_essay_tfidf.shape)
print(X_test_pt_tfidf.shape)
```

```
(73196, 14209)
(73196, 2656)
(36052, 14209)
(36052, 2656)
```

In [12]:

```
# train tfidf w2v using pretrained model - essay

tfidf_model = TfidfVectorizer()
tfidf_model.fit( X_train['essay'].values )
dictionary = dict ( zip (tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

X_train_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in thi
for sentence in tqdm(X_train['essay'].values): # 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((sen
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_tfidf_w2v_vectors.append(vector)

print(len(X_train_tfidf_w2v_vectors))
print(len(X_train_tfidf_w2v_vectors[0]))

X_test_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this
for sentence in tqdm(X_test['essay'].values): # 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((sen
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_tfidf_w2v_vectors.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████| 73196/
73196 [04:23<00:00, 278.24it/s]
  0%|                                                                  | 26/
36052 [00:00<02:19, 258.79it/s]
73196
300
100%|████████████████████████████████████████████████████████████████| 36052/
36052 [02:08<00:00, 280.08it/s]
```

In [13]:

```
X_train_tfidf_w2v_vectors = np.array(X_train_tfidf_w2v_vectors)
X_test_tfidf_w2v_vectors = np.array(X_test_tfidf_w2v_vectors)
```

```
print(X_train_tfidf_w2v_vectors.shape)
print(X_test_tfidf_w2v_vectors.shape)
```

```
(73196, 300)
(36052, 300)
```

In [14]:
```
# train/test tfidf w2v using pretrained model - project title

tfidf_model = TfidfVectorizer()
tfidf_model.fit( X_train['project_title'].values )
dictionary = dict ( zip (tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

X_train_tfidf_w2v_vectors_pt = []; # the avg-w2v for each project_title is stored in th

for sentence in tqdm(X_train['project_title'].values): # 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((sen
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_tfidf_w2v_vectors_pt.append(vector)

print(len(X_train_tfidf_w2v_vectors_pt))
print(len(X_train_tfidf_w2v_vectors_pt[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 73196/73
196 [00:01<00:00, 58697.72it/s]
73196
300
```

In [15]:
```
X_test_tfidf_w2v_vectors_pt = []; # the avg-w2v for each sentence/review is stored in t
for sentence in tqdm(X_test['project_title'].values): # 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((sen
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_tfidf_w2v_vectors_pt.append(vector)

print(len(X_test_tfidf_w2v_vectors_pt))
print(len(X_test_tfidf_w2v_vectors_pt[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 36052/36
052 [00:00<00:00, 58943.70it/s]
36052
300
```

In [16]:
```
X_train_tfidf_w2v_vectors_pt = np.array(X_train_tfidf_w2v_vectors_pt)
X_test_tfidf_w2v_vectors_pt = np.array(X_test_tfidf_w2v_vectors_pt)

print(X_train_tfidf_w2v_vectors_pt.shape)
print(X_test_tfidf_w2v_vectors_pt.shape)
```

```
(73196, 300)
(36052, 300)
```

In [17]:
```
#sentiment analyser feature for train
sid = SentimentIntensityAnalyzer()

Train_sent_lst = []
for sentence in tqdm(X_train['essay'].values):
    ss = sid.polarity_scores(sentence)
    temp = []
    for k in ss:
        temp.append(ss[k])
    Train_sent_lst.append(temp)
Train_sent_arr = np.array(Train_sent_lst)
print(Train_sent_arr.shape)
```

```
100%|████████████████████████████████████████████████████████████████████████| 73196/
73196 [03:51<00:00, 316.14it/s]
(73196, 4)
```

In [18]:
```
#sentiment analyser feature for test
sid = SentimentIntensityAnalyzer()

Test_sent_lst = []
for sentence in tqdm(X_test['essay'].values):
    ss = sid.polarity_scores(sentence)
    temp = []
    for k in ss:
        temp.append(ss[k])
    Test_sent_lst.append(temp)
Test_sent_arr = np.array(Test_sent_lst)
print (Test_sent_arr.shape)
```

```
100%|████████████████████████████████████████████████████████████████████████| 36052/
36052 [01:47<00:00, 335.80it/s]
(36052, 4)
```

In [ ]:
```
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [19]:
```
# please write all the code with proper documentation, and proper titles for each subse
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
```

```
    # d. Y-axis label

#Numerical Features
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

print(" Normalizing Price feature")

normalizer.fit(X_train['price'].values.reshape(1,-1)) #reshaping to one column from one

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

X_train_price_norm = X_train_price_norm.reshape(-1,1) #reshaping back to
X_test_price_norm = X_test_price_norm.reshape(-1,1)

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)  #checking to ensure ouput dimensions ar
print(X_test_price_norm.shape, y_test.shape)
print("="*100)

print(" Normalizing Teacher number of previously posted projects feature")

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1

X_train_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X_trai
X_test_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X_test[

X_train_teacher_number_of_previously_posted_projects_norm =X_train_teacher_number_of_pr
X_test_teacher_number_of_previously_posted_projects_norm = X_test_teacher_number_of_pre
```

```
 Normalizing Price feature
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
 Normalizing Teacher number of previously posted projects feature
```

In [20]:

```
#response encoding

# print ("   One Hot Encoding of feature Teacher Prefix")
# vectorizer = CountVectorizer()      #count vectorizer for one hot encoding
# vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train da

# # we use the fitted CountVectorizer to convert the text to vector
# X_train_teacher_prefix_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
# X_test_teacher_prefix_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

# print("After vectorizations")
# print(X_train_teacher_prefix_ohe.shape, y_train.shape)  #checking to ensure both test
# print(X_test_teacher_prefix_ohe.shape, y_test.shape)
# print(vectorizer.get_feature_names())       #checking feature names
# print("="*100)

dict_project_is_approved_0 = {}
dict_project_is_approved_1 = {}
dict_project_is_approved_tot = {}

for i in data['teacher_prefix'].unique():
    dict_project_is_approved_0[i] = data[(data.teacher_prefix == i) & (data.project_is_
```

```
    dict_project_is_approved_1[i] = data[(data.teacher_prefix == i) & (data.project_is_
    dict_project_is_approved_tot[i] = data[(data.teacher_prefix == i) & (data.project_i

train_resp_encoding_teacher_prefix = []

for i in (X_train['teacher_prefix'].values):
    train_resp_encoding_teacher_prefix.append([ dict_project_is_approved_0.get(i)/dict_
train_resp_encoding_teacher_prefix = np.array(train_resp_encoding_teacher_prefix)

test_resp_encoding_teacher_prefix = []

for i in (X_test['teacher_prefix'].values):
    if i not in dict_project_is_approved_0:
        test_resp_encoding_teacher_prefix.append([0.5,0.5])
    else:
        test_resp_encoding_teacher_prefix.append([ dict_project_is_approved_0.get(i)/di
test_resp_encoding_teacher_prefix = np.array(test_resp_encoding_teacher_prefix)

print(train_resp_encoding_teacher_prefix.shape)
print(test_resp_encoding_teacher_prefix.shape)
print(train_resp_encoding_teacher_prefix[:5])
print(test_resp_encoding_teacher_prefix[:5])
```

```
(73196, 2)
(36052, 2)
[[0.15646259 0.84353741]
 [0.20466102 0.79533898]
 [0.15852742 0.84147258]
 [0.15852742 0.84147258]
 [0.14443358 0.85556642]]
[[0.15646259 0.84353741]
 [0.15646259 0.84353741]
 [0.15646259 0.84353741]
 [0.15852742 0.84147258]
 [0.14443358 0.85556642]]
```

In [21]:

```
dict_project_is_approved_0 = {}
dict_project_is_approved_1 = {}
dict_project_is_approved_tot = {}

for i in data['project_grade_category'].unique():
    dict_project_is_approved_0[i] = data[(data.project_grade_category == i) & (data.pro
    dict_project_is_approved_1[i] = data[(data.project_grade_category == i) & (data.pro
    dict_project_is_approved_tot[i] = data[(data.project_grade_category == i) & (data.p

train_resp_encoding_project_grade_category = []

for i in (X_train['project_grade_category'].values):
    train_resp_encoding_project_grade_category.append([ dict_project_is_approved_0.get(
train_resp_encoding_project_grade_category = np.array(train_resp_encoding_project_grade

test_resp_encoding_project_grade_category = []

for i in (X_test['project_grade_category'].values):
    if i not in dict_project_is_approved_0:
        test_resp_encoding_project_grade_category.append([0.5,0.5])
    else:
        test_resp_encoding_project_grade_category.append([ dict_project_is_approved_0.g
test_resp_encoding_project_grade_category = np.array(test_resp_encoding_project_grade_c
```

```
print(train_resp_encoding_project_grade_category.shape)
print(test_resp_encoding_project_grade_category.shape)
print(train_resp_encoding_project_grade_category[:5])
print(test_resp_encoding_project_grade_category[:5])
```

```
(73196, 2)
(36052, 2)
[[0.14562296 0.85437704]
 [0.15124929 0.84875071]
 [0.15747799 0.84252201]
 [0.14562296 0.85437704]
 [0.14562296 0.85437704]]
[[0.14562296 0.85437704]
 [0.15747799 0.84252201]
 [0.15124929 0.84875071]
 [0.16236432 0.83763568]
 [0.16236432 0.83763568]]
```

In [22]:

```
dict_project_is_approved_0 = {}
dict_project_is_approved_1 = {}
dict_project_is_approved_tot = {}

for i in data['school_state'].unique():
    dict_project_is_approved_0[i] = data[(data.school_state == i) & (data.project_is_ap
    dict_project_is_approved_1[i] = data[(data.school_state == i) & (data.project_is_ap
    dict_project_is_approved_tot[i] = data[(data.school_state == i) & (data.project_is_

train_resp_encoding_school_state = []

for i in (X_train['school_state'].values):
    train_resp_encoding_school_state.append([ dict_project_is_approved_0.get(i)/dict_pr
train_resp_encoding_school_state = np.array(train_resp_encoding_school_state)

test_resp_encoding_school_state = []

for i in (X_test['school_state'].values):
    if i not in dict_project_is_approved_0:
        test_resp_encoding_school_state.append([0.5,0.5])
    else:
        test_resp_encoding_school_state.append([ dict_project_is_approved_0.get(i)/dict
test_resp_encoding_school_state = np.array(test_resp_encoding_school_state)



print(train_resp_encoding_school_state.shape)
print(test_resp_encoding_school_state.shape)
print(train_resp_encoding_school_state[:5])
print(test_resp_encoding_school_state[:5])
```

```
(73196, 2)
(36052, 2)
[[0.1449617  0.8550383 ]
 [0.13980745 0.86019255]
 [0.14033889 0.85966111]
 [0.14186379 0.85813621]
 [0.16831043 0.83168957]]
[[0.14186379 0.85813621]
 [0.16116248 0.83883752]
 [0.14398422 0.85601578]
 [0.14186379 0.85813621]
 [0.14186379 0.85813621]]
```

In [23]:

```
dict_project_is_approved_0 = {}
```

```
dict_project_is_approved_1 = {}
dict_project_is_approved_tot = {}

for i in data['clean_categories'].unique():
    dict_project_is_approved_0[i] = data[(data.clean_categories == i) & (data.project_i
    dict_project_is_approved_1[i] = data[(data.clean_categories == i) & (data.project_i
    dict_project_is_approved_tot[i] = data[(data.clean_categories == i) & (data.project

train_resp_encoding_clean_categories = []

for i in (X_train['clean_categories'].values):
    train_resp_encoding_clean_categories.append([ dict_project_is_approved_0.get(i)/dic
train_resp_encoding_clean_categories = np.array(train_resp_encoding_clean_categories)

test_resp_encoding_clean_categories = []

for i in (X_test['clean_categories'].values):
    if i not in dict_project_is_approved_0:
        test_resp_encoding_clean_categories.append([0.5,0.5])
    else:
        test_resp_encoding_clean_categories.append([ dict_project_is_approved_0.get(i)/
test_resp_encoding_clean_categories = np.array(test_resp_encoding_clean_categories)



print(train_resp_encoding_clean_categories.shape)
print(test_resp_encoding_clean_categories.shape)
print(train_resp_encoding_clean_categories[:5])
print(test_resp_encoding_clean_categories[:5])
```

```
(73196, 2)
(36052, 2)
[[0.18047095 0.81952905]
 [0.16531605 0.83468395]
 [0.14498069 0.85501931]
 [0.13253012 0.86746988]
 [0.13253012 0.86746988]]
[[0.12652768 0.87347232]
 [0.18047095 0.81952905]
 [0.14023591 0.85976409]
 [0.18047095 0.81952905]
 [0.15102683 0.84897317]]
```

In [24]:

```
dict_project_is_approved_0 = {}
dict_project_is_approved_1 = {}
dict_project_is_approved_tot = {}

for i in data['clean_subcategories'].unique():
    dict_project_is_approved_0[i] = data[(data.clean_subcategories == i) & (data.projec
    dict_project_is_approved_1[i] = data[(data.clean_subcategories == i) & (data.projec
    dict_project_is_approved_tot[i] = data[(data.clean_subcategories == i) & (data.proj

# print(dict_project_is_approved_0)
# print(dict_project_is_approved_1)
# print(dict_project_is_approved_tot)


train_resp_encoding_clean_subcategories = []

for i in (X_train['clean_subcategories'].values):
    train_resp_encoding_clean_subcategories.append([ dict_project_is_approved_0.get(i)/
train_resp_encoding_clean_subcategories = np.array(train_resp_encoding_clean_subcategor
```

```
test_resp_encoding_clean_subcategories = []

for i in (X_test['clean_subcategories'].values):
    if i not in dict_project_is_approved_0:
        test_resp_encoding_clean_subcategories.append([0.5,0.5])
    else:
        test_resp_encoding_clean_subcategories.append([ dict_project_is_approved_0.get(
test_resp_encoding_clean_subcategories = np.array(test_resp_encoding_clean_subcategorie



print(train_resp_encoding_clean_subcategories.shape)
print(test_resp_encoding_clean_subcategories.shape)
print(train_resp_encoding_clean_subcategories[:5])
print(test_resp_encoding_clean_subcategories[:5])
```

```
(73196, 2)
(36052, 2)
[[0.1691674  0.8308326 ]
 [0.18148148 0.81851852]
 [0.11345109 0.88654891]
 [0.11754164 0.88245836]
 [0.129812   0.870188  ]]
[[0.12195122 0.87804878]
 [0.18218299 0.81781701]
 [0.14047619 0.85952381]
 [0.1691674  0.8308326 ]
 [0.16150628 0.83849372]]
```

In [25]:

```
from scipy.sparse import hstack
from scipy.sparse import coo_matrix  #to solve error while creating set 2 of dense matr

# Set 1: categorical(instead of one hot encoding, try response coding: use probability
# Set 2: categorical(instead of one hot encoding, try response coding: use probability

#Set 1
X_tr_tfidf = hstack((train_resp_encoding_teacher_prefix, train_resp_encoding_project_gr
X_te_tfidf = hstack((test_resp_encoding_teacher_prefix, test_resp_encoding_project_grad

print("Final Data matrix tfidf")
print(X_tr_tfidf.shape, y_train.shape)   #final train matrix after horizontally stackin
print(X_te_tfidf.shape, y_test.shape)    #final test matrix after horizontally stacking
print("="*100)
```

```
Final Data matrix tfidf
(73196, 16881) (73196,)
(36052, 16881) (36052,)
====================================================================================================
============
```

In [26]:

```
#Set 2

X_tr_tfidfw2v = hstack(( train_resp_encoding_teacher_prefix, train_resp_encoding_projec
X_te_tfidfw2v = hstack(( test_resp_encoding_teacher_prefix, test_resp_encoding_project_

print("Final Data matrix tfidfw2v")
print(X_tr_tfidfw2v.shape, y_train.shape)   #final train matrix after horizontally stac
print(X_te_tfidfw2v.shape, y_test.shape)    #final test matrix after horizontally stack
print("="*100)
```

```
Final Data matrix tfidfw2v
(73196, 612) (73196,)
```

```
(36052, 612) (36052,)
==========================================================================================
============
```

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

All the steps for SET - 1 TFIDF

In [26]:
```
# please write all the code with proper documentation, and proper titles for each subse
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

#hyperparameter tuning
gbc = GradientBoostingClassifier(random_state=0)
param_grid = {'max_depth':[1, 5, 10] , 'min_samples_split':[10, 100, 500]}
clf = GridSearchCV( gbc, param_grid, cv=3, scoring='roc_auc', return_train_score=True,
clf.fit(X_tr_tfidf, y_train)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  3.2min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  6.2min
[Parallel(n_jobs=-1)]: Done  15 out of  27 | elapsed: 17.2min remaining: 13.7min
[Parallel(n_jobs=-1)]: Done  18 out of  27 | elapsed: 27.7min remaining: 13.8min
[Parallel(n_jobs=-1)]: Done  21 out of  27 | elapsed: 38.1min remaining: 10.9min
[Parallel(n_jobs=-1)]: Done  24 out of  27 | elapsed: 38.3min remaining:  4.8min
[Parallel(n_jobs=-1)]: Done  27 out of  27 | elapsed: 47.3min remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  27 out of  27 | elapsed: 47.3min finished
```

Out[26]:
```
GridSearchCV(cv=3, estimator=GradientBoostingClassifier(random_state=0),
             n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10],
                         'min_samples_split': [10, 100, 500]},
             return_train_score=True, scoring='roc_auc', verbose=10)
```

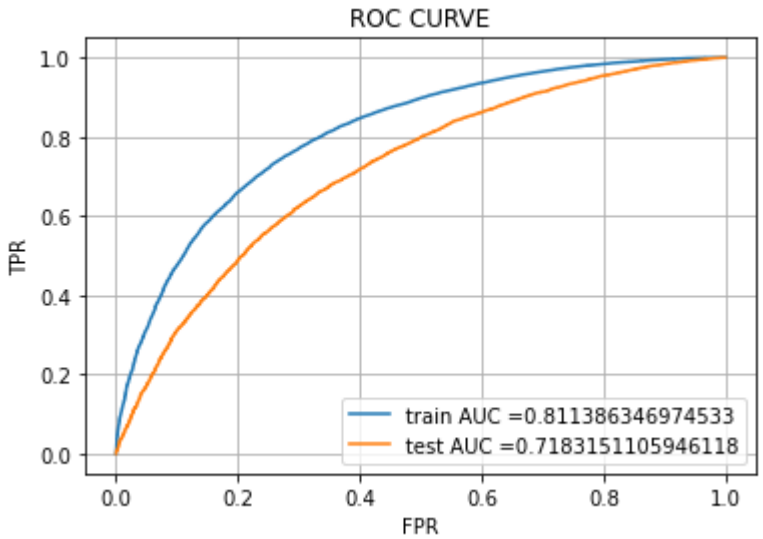
In [27]:
```
print("Best AUC score : ",clf.best_score_)
print("Best params : ",clf.best_params_)
```

```
Best AUC score :  0.7160005187822972
Best params :  {'max_depth': 5, 'min_samples_split': 100}
```

In [28]:
```
results = pd.DataFrame.from_dict(clf.cv_results_)  #storing results of gridsearch in pa

results = results.sort_values(['rank_test_score'])

train_auc= results['mean_train_score']

cv_auc = results['mean_test_score']
```

```
K =  results['params']

max_d = []
min_sam_splt = []

for i in K:
    max_d.append(i.get('max_depth'))        #max depth values
    min_sam_splt.append(i.get('min_samples_split'))     #min sample split values

#plotting 3D plot as per given ipynb

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


x1 = min_sam_splt
y1 = max_d
z1 = train_auc

x2 = min_sam_splt
y2 = max_d
z2 = cv_auc

trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'Train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'CV')
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')
```

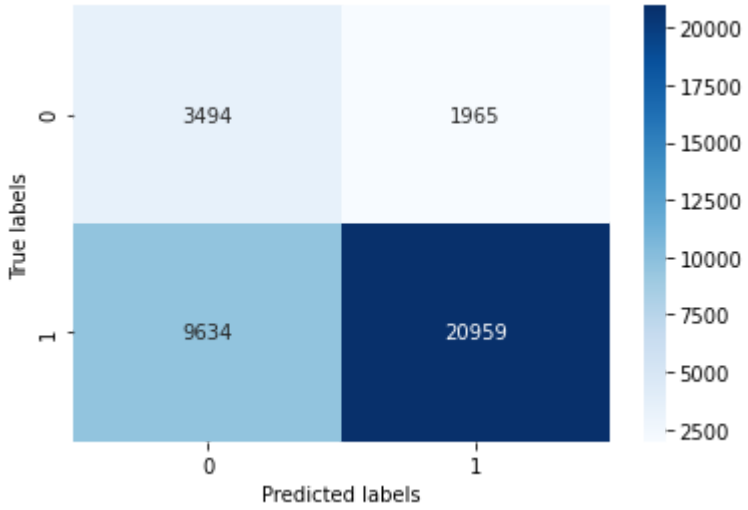
In [29]:
```
from sklearn.metrics import roc_curve, auc

#training on best hyperparameters
best_gbc = GradientBoostingClassifier( max_depth = 5, min_samples_split=100 , random_st
best_gbc.fit(X_tr_tfidf, y_train)

#predicted proabilities
y_train_pred = best_gbc.predict_proba(X_tr_tfidf)[:,1]  # train predicted probabilities
y_test_pred = best_gbc.predict_proba(X_te_tfidf)[:,1]   # test predicted probabilities

#plotting ROC Curve
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)    #train fpr,tr
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)        #test fpr,tes

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))   #p
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")
plt.grid()
plt.show()
```



In [30]:
```
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.rou
    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

print("="*100)

from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)

#plotting confusion matrix
print("Test confusion matrix")
test_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

ax= plt.subplot()
sns.heatmap(test_cm, annot=True,fmt="d",cmap='Blues' , ax =ax)
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
plt.show()
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.543324242778323 for threshold 0.838
Test confusion matrix
```



All the steps for SET - 2 TFIDFW2V

In [27]:

```
# please write all the code with proper documentation, and proper titles for each subse
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

#hyperparameter tuning
```

```
gbc = GradientBoostingClassifier(random_state=0)
param_grid = {'max_depth':[1, 5, 10] , 'min_samples_split':[ 10, 100, 500]}
clf = GridSearchCV( gbc, param_grid, cv=3, scoring='roc_auc', return_train_score=True,
clf.fit(X_tr_tfidfw2v, y_train)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  7.4min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 13.7min
[Parallel(n_jobs=-1)]: Done  15 out of  27 | elapsed: 40.9min remaining: 32.7min
[Parallel(n_jobs=-1)]: Done  18 out of  27 | elapsed: 72.9min remaining: 36.4min
[Parallel(n_jobs=-1)]: Done  21 out of  27 | elapsed: 126.3min remaining: 36.1min
[Parallel(n_jobs=-1)]: Done  24 out of  27 | elapsed: 129.1min remaining: 16.1min
[Parallel(n_jobs=-1)]: Done  27 out of  27 | elapsed: 165.7min remaining:    0.0s
[Parallel(n_jobs=-1)]: Done  27 out of  27 | elapsed: 165.7min finished
```

Out[27]:
```
GridSearchCV(cv=3, estimator=GradientBoostingClassifier(random_state=0),
             n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10],
                         'min_samples_split': [10, 100, 500]},
             return_train_score=True, scoring='roc_auc', verbose=10)
```

In [28]:
```
print("Best AUC score : ",clf.best_score_)
print("Best params : ",clf.best_params_)
```

```
Best AUC score :  0.7089129001699712
Best params :  {'max_depth': 5, 'min_samples_split': 500}
```

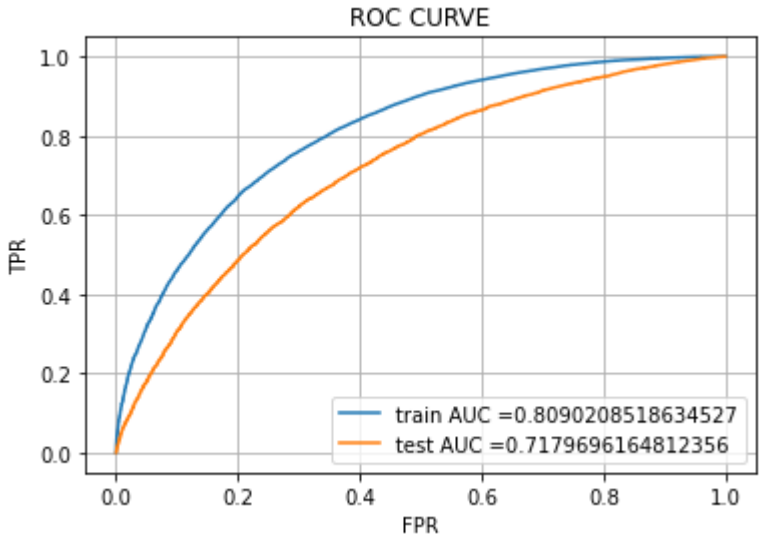
In [29]:
```
results = pd.DataFrame.from_dict(clf.cv_results_)  #storing results of gridsearch in pa

results = results.sort_values(['rank_test_score'])

train_auc= results['mean_train_score']

cv_auc = results['mean_test_score']

K =  results['params']

max_d = []
min_sam_splt = []

for i in K:
    max_d.append(i.get('max_depth'))        #max depth values
    min_sam_splt.append(i.get('min_samples_split'))     #min sample split values

#plotting 3D plot as per given ipynb

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


x1 = min_sam_splt
y1 = max_d
z1 = train_auc

x2 = min_sam_splt
y2 = max_d
z2 = cv_auc

trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'Train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'CV')
```

```
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 [31]:

```
from sklearn.metrics import roc_curve, auc

#training on best hyperparameters
best_gbc = GradientBoostingClassifier( max_depth = 5, min_samples_split=500 , random_st
best_gbc.fit(X_tr_tfidfw2v, y_train)

#predicted proabilities
y_train_pred = best_gbc.predict_proba(X_tr_tfidfw2v)[:,1]  # train predicted probabilit
y_test_pred = best_gbc.predict_proba(X_te_tfidfw2v)[:,1]   # test predicted probabiliti

#plotting ROC Curve
train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)    #train fpr,tr
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)        #test fpr,tes

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))   #p
```

```
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")
plt.grid()
plt.show()
```

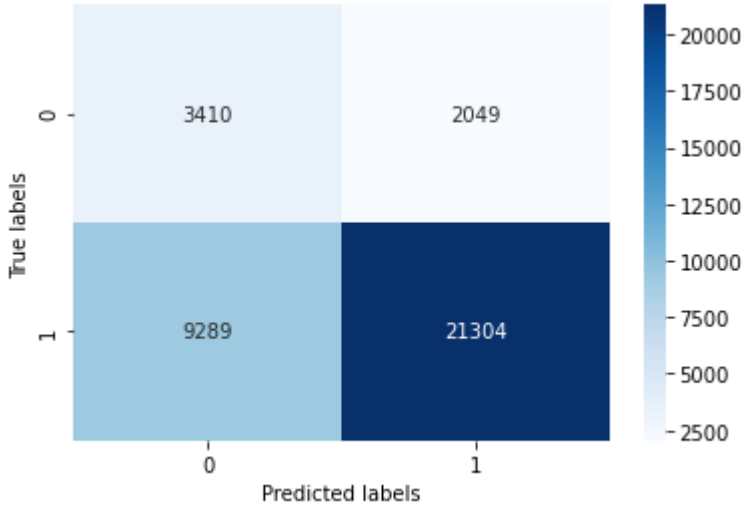


In [32]:

```
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.rou
    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

print("="*100)

from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)

#plotting confusion matrix
print("Test confusion matrix")
test_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

ax= plt.subplot()
sns.heatmap(test_cm, annot=True,fmt="d",cmap='Blues' , ax =ax)
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
plt.show()
```

```
==========================================================================================
============
```

```
the maximum value of tpr*(1-fpr) 0.534649414100378 for threshold 0.834
Test confusion matrix
```



3. Summary

as mentioned in the step 4 of instructions

In [34]:
```
from tabulate import tabulate
print(tabulate([['TFIDF', 'GradientBoostingClassifier' , '5,100' ,'0.72'], ['TFIDFW2V',
```

```
Vectorizer    Model                        Hyper parameter      AUC
------------  ---------------------------  -----------------  -----
TFIDF         GradientBoostingClassifier   5,100                0.72
TFIDFW2V      GradientBoostingClassifier   5,500                0.72
```