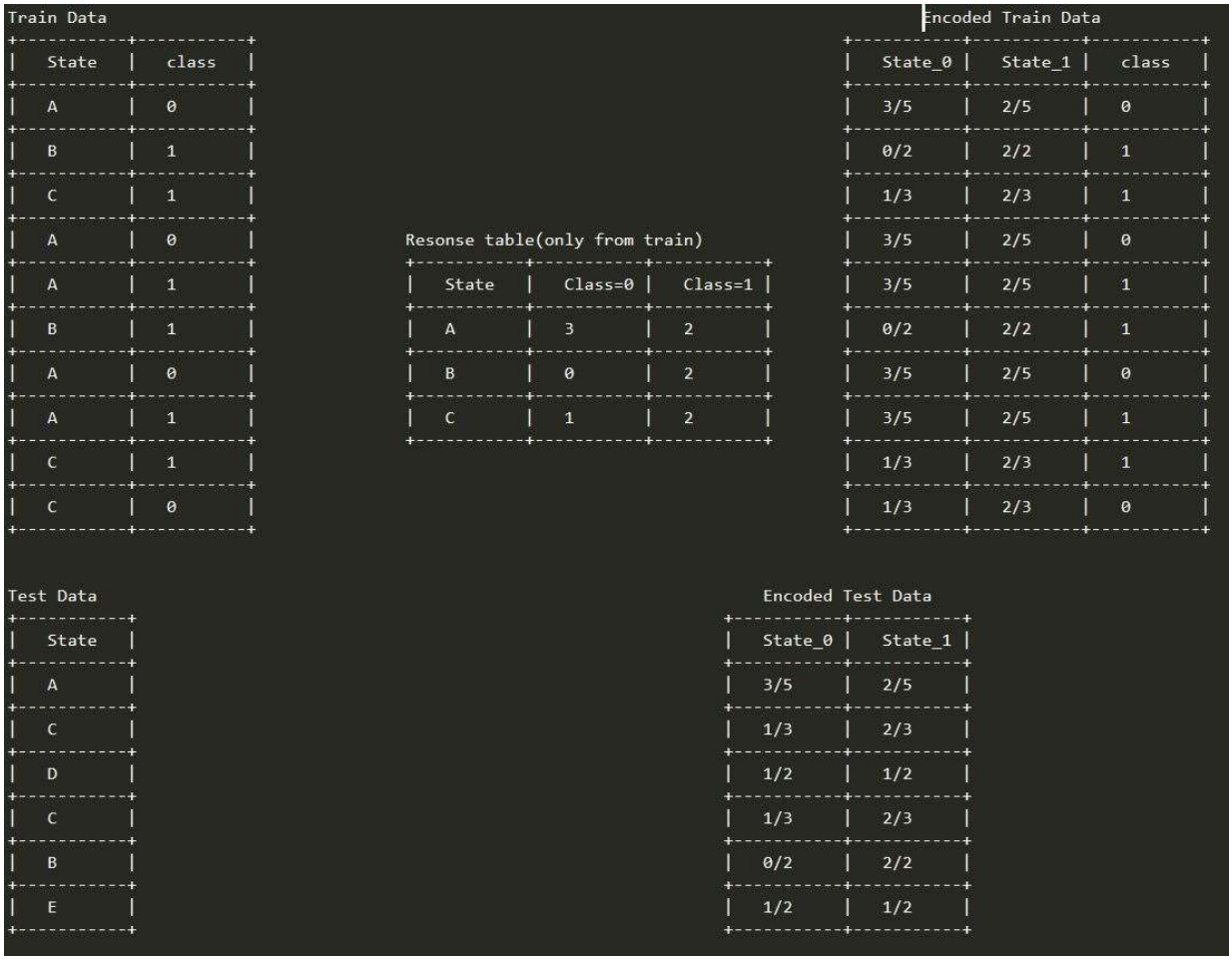
# Assignment 9: GBDT

### Response Coding: Example




> 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/): 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/): use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

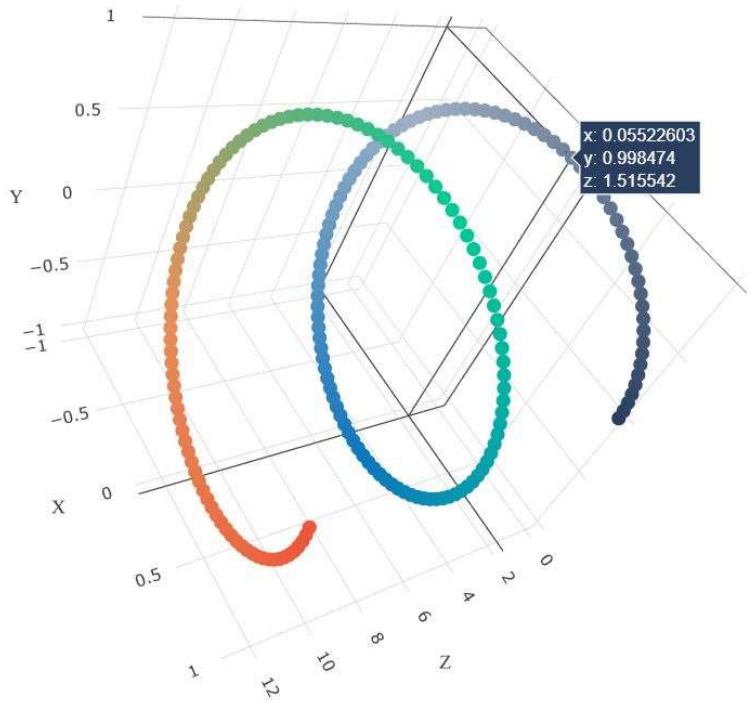
- Here in response encoding you need to apply the **laplase smoothing** value for test set. Laplase smoothing means, If test point is present in test but not in train then you need to apply default 0.5 as probability value for that data point (Refer the Response Encoding Image from above cell)
- Please use atleast **35k** data points

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/) 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**

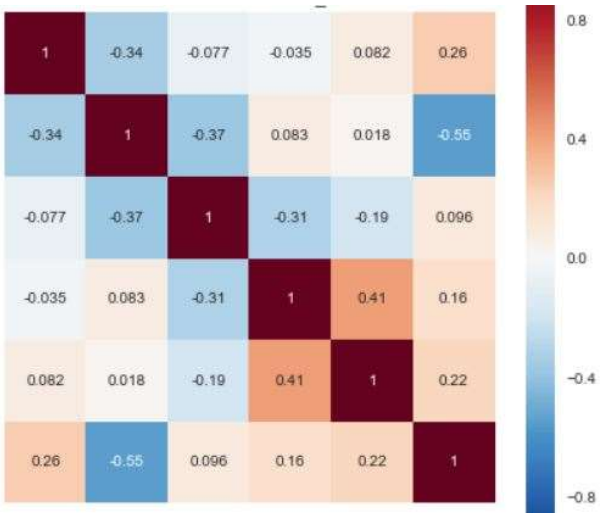
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure




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*

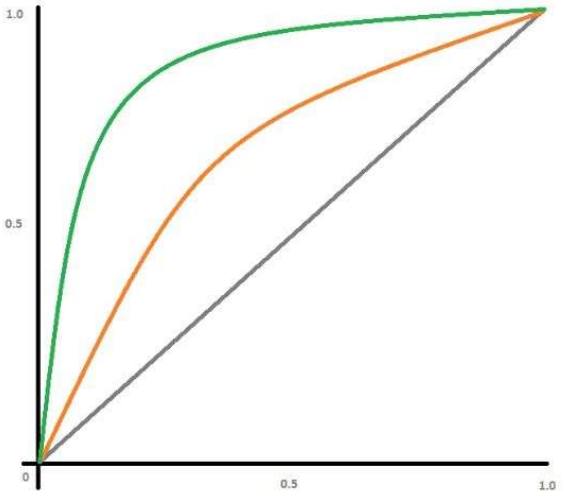
**or**

- 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) 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. Make sure that you are using predict_proba method to calculate AUC curves, because AUC is calcualted on class probabilities and not on class labels.




- 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/) 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 |

# Few Notes

1. Use atleast 35k data points
2. Use classifier.Predict_proba() method instead of predict() method while calculating roc_auc scores
3. Be sure that you are using laplase smoothing in response encoding function. Laplase smoothing means applying the default (0.5) value to test data if the test data is not present in the train set

In [1]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

sample_sentence_1='I am happy.'
ss_1 = sid.polarity_scores(sample_sentence_1)
print('sentiment score for sentence 1',ss_1)

sample_sentence_2='I am sad.'
ss_2 = sid.polarity_scores(sample_sentence_2)
print('sentiment score for sentence 2',ss_2)

sample_sentence_3='I am going to New Delhi tommorow.'
ss_3 = sid.polarity_scores(sample_sentence_3)
print('sentiment score for sentence 3',ss_3)
```

```
sentiment score for sentence 1 {'neg': 0.0, 'neu': 0.213, 'pos': 0.787, 'compou
nd': 0.5719}
sentiment score for sentence 2 {'neg': 0.756, 'neu': 0.244, 'pos': 0.0, 'compou
nd': -0.4767}
sentiment score for sentence 3 {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound':
0.0}
```

# 1. GBDT (xgboost/lightgbm)

# 1.1 Loading Data

In [2]:
```
#importing Packages
import warnings
warnings.filterwarnings("ignore")
import pandas
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 re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/

import pickle
from tqdm import tqdm
import os
from sklearn.model_selection import train_test_split
!pip install chart_studio
from chart_studio import plotly
import plotly.offline as offline
import plotly.graph_objs as go
```

```
Requirement already satisfied: chart_studio in c:\anaconda3\lib\site-packages
(1.1.0)
Requirement already satisfied: six in c:\anaconda3\lib\site-packages (from char
t_studio) (1.15.0)
Requirement already satisfied: requests in c:\anaconda3\lib\site-packages (from
chart_studio) (2.25.1)
Requirement already satisfied: retrying>=1.3.3 in c:\anaconda3\lib\site-package
s (from chart_studio) (1.3.3)
Requirement already satisfied: plotly in c:\anaconda3\lib\site-packages (from c
hart_studio) (5.1.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\anaconda3\lib\site-package
s (from plotly->chart_studio) (6.3.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\anaconda3\lib\site-pack
ages (from requests->chart_studio) (2020.12.5)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\anaconda3\lib\site-p
ackages (from requests->chart_studio) (1.26.4)
Requirement already satisfied: idna<3,>=2.5 in c:\anaconda3\lib\site-packages
(from requests->chart_studio) (2.10)
Requirement already satisfied: chardet<5,>=3.0.2 in c:\anaconda3\lib\site-packa
ges (from requests->chart_studio) (4.0.0)
```

In [3]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv',nrows=35000)  #reading the data fr
```

In [4]:
```
data.head(3)
```

Out[4]:

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projec |
|---|---|---|---|---|
| **0** | ca | mrs | grades_prek_2 | |
| **1** | ut | ms | grades_3_5 | |
| **2** | ca | mrs | grades_prek_2 | |

In [5]:
```
prep_data= pandas.read_csv('train_data.csv', nrows=35000)  #reading data from tra
```

In [6]:
```
prep_data.columns.values          #getting the names of columns
```

Out[6]:
```
array(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'project_subject_categories', 'project_subject_subcategories',
       'project_title', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects',
       'project_is_approved'], dtype=object)
```

In [84]:
```
#pre processing of Project Title
```

In [7]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

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

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

In [8]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', '
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', '
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itsel
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'th
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "di
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn'
            'won', "won't", 'wouldn', "wouldn't"]
```

In [9]:
```
# Combining all the above stundents
from tqdm import tqdm
def preprocess_text(text_data):
    preprocessed_text = []
    # tqdm is for printing the status bar
    for sentance in tqdm(text_data):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\n', ' ')
        sent = sent.replace('\\"', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
        preprocessed_text.append(sent.lower().strip())
    return preprocessed_text
```

In [10]:
```
preprocessed_titles = preprocess_text(prep_data['project_title'].values)
```

```
100%|██████████████████████████████████████████████████████████████████████████████|
35000/35000 [00:01<00:00, 25406.08it/s]
```

In [11]:
```
data['project_title']=preprocessed_titles  #adding the preprocessed title in prep
```

In [12]:
```
data.head(3)  #top 3 rows
```

Out[12]:

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projec |
|---|---|---|---|---|
| **0** | ca | mrs | grades_prek_2 | |
| **1** | ut | ms | grades_3_5 | |
| **2** | ca | mrs | grades_prek_2 | |

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

In [13]:
```
# please write all the code with proper documentation, and proper titles for each
# 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 debuggin
# 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
```

In [14]:
```
#getting the sentiment score

neg = []
pos = []
neu = []
compound = []

for a in data["essay"] :
    neg.append(sid.polarity_scores(a)['neg'])
    pos.append(sid.polarity_scores(a)['pos'])
    neu.append(sid.polarity_scores(a)['neu'])
    compound.append(sid.polarity_scores(a)['compound'])
```

In [15]:
```
#adding to the dataset

data['neg']=neg
data['pos']=pos
data['neu']=neu
data['compound']=compound
```

In [16]:
```
data.head(3)  #getting top 3 rows
```

Out[16]:

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projec |
|---|---|---|---|---|
| **0** | ca | mrs | grades_prek_2 | |
| **1** | ut | ms | grades_3_5 | |
| **2** | ca | mrs | grades_prek_2 | |

In [17]:
```
y= data['project_is_approved'].values
X=data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[17]:

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projec |
|---|---|---|---|---|
| 0 | ca | mrs | grades_prek_2 | |

In [18]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.33, stratif
```

In [19]:
```
print(len(X_train))
print(len(X_test))
print(len(Y_train))
print(len(Y_test))
```

```
23450
11550
23450
11550
```

In [20]:
```
data.columns.values #getting the names of columns
```

Out[20]:
```
array(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects',
       'project_is_approved', 'clean_categories', 'clean_subcategories',
       'essay', 'price', 'project_title', 'neg', 'pos', 'neu', 'compound'],
      dtype=object)
```

## 1.3 Make Data Model Ready: encoding eassay, and project_title

In [21]:
```
# please write all the code with proper documentation, and proper titles for each
# 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 debuggin
# 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
```

In [22]:
```
#project_title(TFIDF)+ preprocessed_eassay (TFIDF)
```

In [23]:
```
#perform tfidf vectorization of text data i. essay

vectorizer = TfidfVectorizer(min_df=10)
text_essay = vectorizer.fit(X_train['essay'])
X_train_essays_tfidf = vectorizer.transform(X_train['essay'])
X_test_essays_tfidf  = vectorizer.transform(X_test['essay'])

print(X_train_essays_tfidf.shape)
print(X_test_essays_tfidf.shape)
```

```
(23450, 8967)
(11550, 8967)
```

In [24]:
```
#performing tfidf vectorization on project title

vectorizer = TfidfVectorizer(min_df=10)
text_essay = vectorizer.fit(X_train['project_title'])
X_train_project_title_tfidf = vectorizer.transform(X_train['project_title'])
X_test_project_title_tfidf  = vectorizer.transform(X_test['project_title'])

print(X_train_project_title_tfidf.shape)
print(X_test_project_title_tfidf.shape)
```

```
(23450, 1208)
(11550, 1208)
```

## 1.4 Make Data Model Ready: encoding numerical, categorical features

In [25]:
```
# please write all the code with proper documentation, and proper titles for each
# 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 debuggi
# 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
```

In [26]:
```
# 5. perform encoding of categorical features.
# school_state
# teacher_prefix
# project_grade_category
# clean_categories
# clean_subcategories
```

In [90]:
```
#Refer : https://www.appliedaicourse.com/

def get_gv_fea_dict(feature,df):

    X_train['class_label']=Y_train # adding 'project_is_approved' column to x_tra
    count = X_train[ feature ].value_counts() # getting value counts(denominator)
    feature_dictionary = dict()
    n=2
    for i, denominator in count.items():
        vector = []
        for j in range(n):
            compare =X_train.loc[ ( X_train['class_label'] == j ) & (X_train[feat
            vector.append( len( compare)  / denominator   )
        feature_dictionary[i] = vector    # adding probability of each class labe
    return feature_dictionary
```

In [91]:
```
#Refer : https://www.appliedaicourse.com/

def get_gv_feature(feature, df ):
    alpha1= 0.5
    alpha2= 0.05
    feature_dictionary = get_gv_fea_dict(feature,df) #calling the function
    count = X_train[feature].value_counts()
    f=[]
    for index,row in df.iterrows():
        if row[feature] in dict( count ).keys():# transform test datas
            f.append( feature_dictionary[row[feature]] )
        else:
            f.append([alpha1, alpha2])  #adding alphas i.e laplace smoothing
    return f
```

In [92]:
```
#school_state
Train_Response_encoding_school_state= np.array(get_gv_feature('school_state',X_tr
Test_Response_encoding_school_state= np.array(get_gv_feature('school_state',X_tes

print(Train_Response_encoding_school_state.shape)
print(Test_Response_encoding_school_state.shape)
```

```
(23450, 2)
(11550, 2)
```

In [30]:
```
#teacher_prefix
Train_Response_encoding_teacher_prefix= np.array(get_gv_feature('teacher_prefix',
Test_Response_encoding_teacher_prefix= np.array(get_gv_feature('teacher_prefix',X

print(Train_Response_encoding_teacher_prefix.shape)
print(Test_Response_encoding_teacher_prefix.shape)
```

```
(23450, 2)
(11550, 2)
```

In [31]:
```
# project_grade_category
Train_Response_encoding_project_grade_category= np.array(get_gv_feature('project_
Test_Response_encoding_project_grade_category= np.array(get_gv_feature('project_g

print(Train_Response_encoding_project_grade_category.shape)
print(Test_Response_encoding_project_grade_category.shape)
```

```
(23450, 2)
(11550, 2)
```

In [32]:
```
# clean_categories
Train_Response_encoding_clean_categories= np.array(get_gv_feature('clean_categori
Test_Response_encoding_clean_categories= np.array(get_gv_feature('clean_categorie

print(Train_Response_encoding_clean_categories.shape)
print(Test_Response_encoding_clean_categories.shape)
```

```
(23450, 2)
(11550, 2)
```

In [33]:
```
# clean_subcategories
Train_Response_encoding_clean_subcategories= np.array(get_gv_feature('clean_subca
Test_Response_encoding_clean_subcategories= np.array(get_gv_feature('clean_subcat

print(Train_Response_encoding_clean_subcategories.shape)
print(Test_Response_encoding_clean_subcategories.shape)
```

```
(23450, 2)
(11550, 2)
```

In [34]:
```
# 6. perform encoding of numerical features
#- price
#- teacher_number_of_previously_posted_projects
#- semantic_score
```

In [35]:
```
from sklearn.preprocessing import Normalizer
```

In [36]:
```
#- price

norm= Normalizer()
norm.fit(X_train['price'].values.reshape(1,-1))

X_train_price_norm = norm.fit_transform(X_train['price'].values.reshape(1,-1)).re
X_test_price_norm= norm.fit_transform(X_test['price'].values.reshape(1,-1)).resha

print('X_train_price_norm :',X_train_price_norm.shape)
print('X_test_price_norm :',X_test_price_norm.shape)
```

```
X_train_price_norm : (23450, 1)
X_test_price_norm : (11550, 1)
```

In [37]:
```
#- teacher_number_of_previously_posted_projects

norm= Normalizer()
norm.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1

X_train_teacher_number_of_previously_posted_projects_norm = norm.fit_transform(X_
X_test_teacher_number_of_previously_posted_projects_norm = norm.fit_transform(X_t

print(X_train_teacher_number_of_previously_posted_projects_norm.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape)
```

```
(23450, 1)
(11550, 1)
```

In [38]:
```
#neg

norm= Normalizer()
norm.fit(X_train['neg'].values.reshape(1,-1))

X_train_neg_norm = norm.fit_transform(X_train['neg'].values.reshape(1,-1)).reshap
X_test_neg_norm = norm.fit_transform(X_test['neg'].values.reshape(1,-1)).reshape(

print(X_train_neg_norm.shape)
print(X_test_neg_norm.shape)
```

```
(23450, 1)
(11550, 1)
```

In [39]:
```
#pos

norm= Normalizer()
norm.fit(X_train['pos'].values.reshape(1,-1))

X_train_pos_norm = norm.fit_transform(X_train['pos'].values.reshape(1,-1)).reshap
X_test_pos_norm = norm.fit_transform(X_test['pos'].values.reshape(1,-1)).reshape(

print(X_train_pos_norm.shape)
print(X_test_pos_norm.shape)
```

```
(23450, 1)
(11550, 1)
```

In [40]:
```
#neu

norm= Normalizer()
norm.fit(X_train['neu'].values.reshape(1,-1))

X_train_neu_norm = norm.fit_transform(X_train['neu'].values.reshape(1,-1)).reshap
X_test_neu_norm = norm.fit_transform(X_test['neu'].values.reshape(1,-1)).reshape(

print(X_train_neu_norm.shape)
print(X_test_neu_norm.shape)
```

```
(23450, 1)
(11550, 1)
```

In [41]:
```
#compound

norm= Normalizer()
norm.fit(X_train['compound'].values.reshape(1,-1))

X_train_compound_norm = norm.fit_transform(X_train['compound'].values.reshape(1,-
X_test_compound_norm = norm.fit_transform(X_test['compound'].values.reshape(1,-1)

print(X_train_compound_norm.shape)
print(X_test_compound_norm.shape)
```

```
(23450, 1)
(11550, 1)
```

In [42]:
```
#set 1
#categorical(instead of one hot encoding, try response coding: use probability va
#numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
#+sentiment Score of eassay(check the bellow example, include all 4 values as 4 f
```

In [43]:
```
from scipy.sparse import hstack
```

In [44]:
```
train_encoding= hstack((X_train_essays_tfidf,X_train_project_title_tfidf,Train_Re
```

In [45]:
```
print(train_encoding.shape)
```

```
(23450, 10191)
```

In [46]:
```
test_encoding= hstack((X_test_essays_tfidf,X_test_project_title_tfidf,Test_Respor
```

In [47]:
```
print(test_encoding.shape)
```

```
(11550, 10191)
```

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 [48]:
```
# please write all the code with proper documentation, and proper titles for each
# 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 debuggi
# 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
```

In [49]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
gb= GradientBoostingClassifier()
param= {'max_depth':[1,2,3],'n_estimators':[1,5,10]}
clf= GridSearchCV(gb, param,cv=3,  scoring='roc_auc',return_train_score=True)
clf.fit(train_encoding,Y_train)
```

Out[49]:
```
GridSearchCV(cv=3, estimator=GradientBoostingClassifier(),
             param_grid={'max_depth': [1, 2, 3], 'n_estimators': [1, 5, 10]},
             return_train_score=True, scoring='roc_auc')
```

In [50]:
```
clf.cv_results_.keys()
```

Out[50]:
```
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_tim
e', 'param_max_depth', 'param_n_estimators', 'params', 'split0_test_score', 'sp
lit1_test_score', 'split2_test_score', 'mean_test_score', 'std_test_score', 'ra
nk_test_score', 'split0_train_score', 'split1_train_score', 'split2_train_scor
e', 'mean_train_score', 'std_train_score'])
```

In [51]:
```
print('Best score: ',clf.best_score_)
print('Best Hyper parameters: ',clf.best_params_)
```

```
Best score:  0.6633857691567537
Best Hyper parameters:  {'max_depth': 3, 'n_estimators': 10}
```

In [56]:
```
mean_train_auc= clf.cv_results_['mean_train_score']
mean_test_auc=clf.cv_results_['mean_test_score']

std_train_auc= clf.cv_results_['std_train_score']
std_test_auc= clf.cv_results_['std_test_score']

print('mean_train_auc1',mean_train_auc)
print('\n')
print('mean_test_auc1',mean_test_auc)
print('\n')
print('std_train_auc1',std_train_auc)
print('\n')
print('std_test_auc1',std_test_auc)
```

```
mean_train_auc1 [0.56543985 0.6338868  0.65200912 0.59951267 0.66075962 0.67810
967
 0.62880132 0.67875712 0.70398653]


mean_test_auc1 [0.54708864 0.61453967 0.63256599 0.58108209 0.63564995 0.651819
16
 0.60698411 0.64673739 0.66338577]


std_train_auc1 [0.00634563 0.00129356 0.00579144 0.00759874 0.00150535 0.003997
76
 0.00861913 0.00390622 0.00309043]


std_test_auc1 [0.00951497 0.00643684 0.00560802 0.00909786 0.00667108 0.0047437
7
 0.01020376 0.00252399 0.00199957]
```

In [52]:
```
clf_test= GridSearchCV(gb, param,cv=3,  scoring='roc_auc',return_train_score=True
clf_test.fit(test_encoding,Y_test)
```

Out[52]:
```
GridSearchCV(cv=3, estimator=GradientBoostingClassifier(),
             param_grid={'max_depth': [1, 2, 3], 'n_estimators': [1, 5, 10]},
             return_train_score=True, scoring='roc_auc')
```

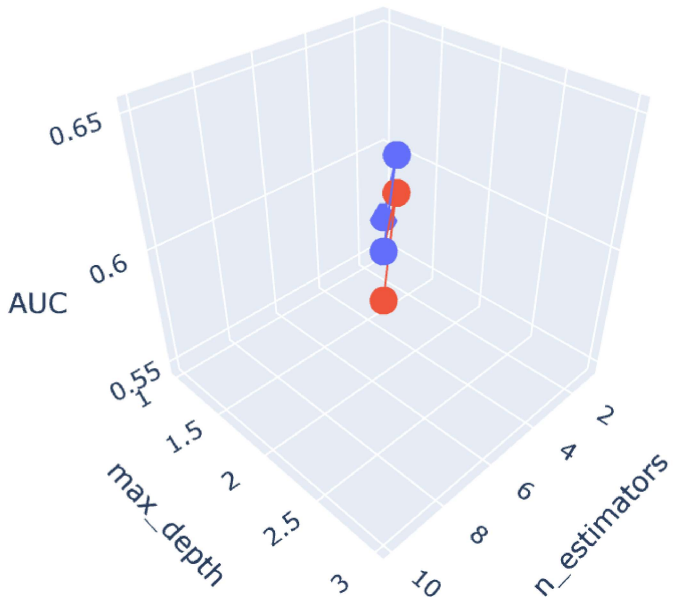
In [53]:
```
mean_train_auc1= clf_test.cv_results_['mean_train_score']
mean_test_auc1=clf_test.cv_results_['mean_test_score']

std_train_auc1= clf_test.cv_results_['std_train_score']
std_test_auc1= clf_test.cv_results_['std_test_score']

print('mean_train_auc1',mean_train_auc1)
print('\n')
print('mean_test_auc1',mean_test_auc1)
print('\n')
print('std_train_auc1',std_train_auc1)
print('\n')
print('std_test_auc1',std_test_auc1)
```

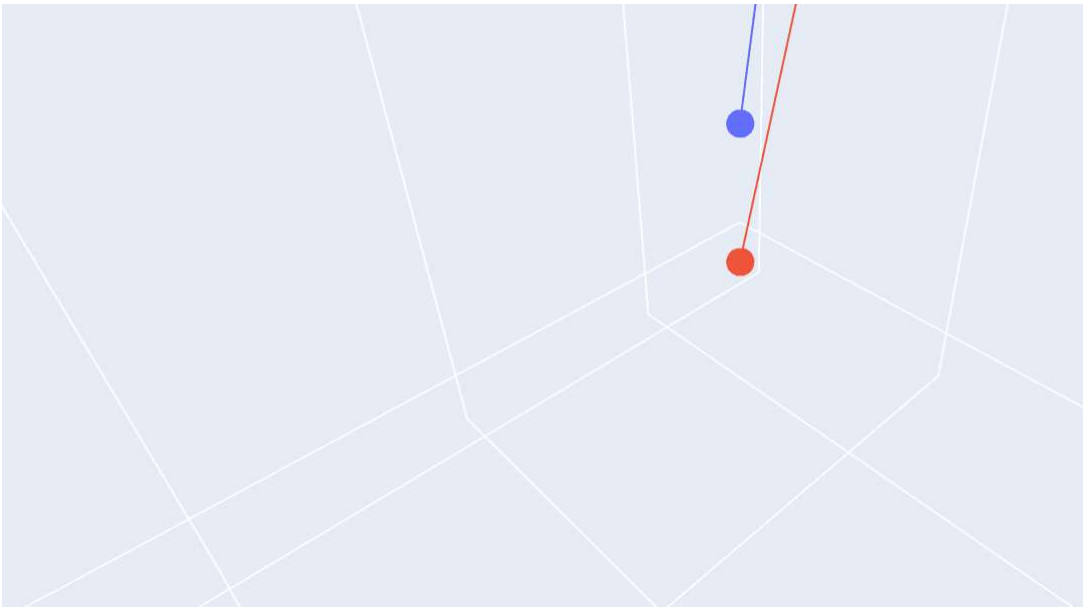
```
mean_train_auc1 [0.54746118 0.6200167  0.63791515 0.57991193 0.65404734 0.68031
361
 0.60944995 0.67805303 0.72424578]


mean_test_auc1 [0.53251247 0.59793629 0.61377144 0.56122681 0.61544312 0.626581
55
 0.5807739  0.61163374 0.6338012 ]


std_train_auc1 [0.01678937 0.00383141 0.00807364 0.02155832 0.00403454 0.009053
97
 0.00246112 0.00843807 0.0083188 ]


std_test_auc1 [0.01491654 0.00737898 0.01131499 0.01590569 0.00904817 0.0127119
3
 0.01105353 0.0123361  0.01743306]
```

In [54]:
```
#X-axis as n_estimators, Y-axis as max_depth, and Z-axis as AUC Score
```

In [57]:
```
x1 = [1,5,10]
y1 = [1,2,3]
z1 = mean_train_auc

x2 =[1,5,10]
y2 = [1,2,3]
z2 = mean_train_auc1
```

In [58]:
```
# 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')
fig.show()
plt.show()
```





In [59]:
```
#project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)
```

In [60]:
```
import pickle
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [61]:
```
# 4. perform tfidf w2v vectorization of text data.
#Refer:: https://www.kaggle.com/nikhilparmar9/decision-tree-donorschoose-dataset#
from scipy import sparse
tfidf_model = TfidfVectorizer()
# tfidf_model.fit(x_train_upsampled['preprocessed_essays'])
tfidf_model.fit(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())

X_train_essays_tfidf_w2v_vectors = [] # the avg-w2v for each sentence/review is s
# for sentence in tqdm(x_train_upsampled['preprocessed_essays']): # for each revi
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 valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_essays_tfidf_w2v_vectors.append(vector)
X_train_essays_tfidf_w2v_vectors=sparse.csr_matrix(X_train_essays_tfidf_w2v_vecto


print("Total number of vectors  : TFIDF-W2V -> Essays: x_train : ",(X_train_essay
```

```
100%|
| 23450/23450 [01:30<00:00, 259.86it/s]

Total number of vectors  : TFIDF-W2V -> Essays: x_train :  (23450, 300)
```

In [62]:
```
# 4. perform tfidf w2v vectorization of text data.
#Refer:: https://www.kaggle.com/nikhilparmar9/decision-tree-donorschoose-dataset#

tfidf_model = TfidfVectorizer()
# tfidf_model.fit(x_train_upsampled['preprocessed_essays'])
tfidf_model.fit(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())

X_test_essays_tfidf_w2v_vectors = [] # the avg-w2v for each sentence/review is st
# for sentence in tqdm(x_train_upsampled['preprocessed_essays']): # for each revi
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 valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_essays_tfidf_w2v_vectors.append(vector)
X_test_essays_tfidf_w2v_vectors=sparse.csr_matrix(X_test_essays_tfidf_w2v_vectors

print("Total number of vectors  : TFIDF-W2V -> Essays: x_train : ",(X_test_essays
```

```
100%|
| 11550/11550 [00:44<00:00, 259.50it/s]

Total number of vectors  : TFIDF-W2V -> Essays: x_train :  (11550, 300)
```

In [63]:
```
# 4. perform tfidf w2v vectorization of text data.
#Refer:: https://www.kaggle.com/nikhilparmar9/decision-tree-donorschoose-dataset#

tfidf_model = TfidfVectorizer()
# tfidf_model.fit(x_train_upsampled['preprocessed_essays'])
tfidf_model.fit(X_test['project_title'])
# 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())

X_test_project_title_tfidf_w2v_vectors = [] # the avg-w2v for each sentence/revie
# for sentence in tqdm(x_train_upsampled['preprocessed_essays']): # for each revi
for sentence in tqdm(X_test['project_title']): # 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 valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_project_title_tfidf_w2v_vectors.append(vector)

X_test_project_title_tfidf_w2v_vectors=sparse.csr_matrix(X_test_project_title_tfi

print("Total number of vectors  : TFIDF-W2V -> project_title: x_test : ",(X_test_
```

```
100%|██████████████████████████████████████████████████████████████████████████|
11550/11550 [00:00<00:00, 25047.28it/s]

Total number of vectors  : TFIDF-W2V -> project_title: x_test :  (11550, 300)
```

In [64]:
```
# 4. perform tfidf w2v vectorization of text data.
#Refer:: https://www.kaggle.com/nikhilparmar9/decision-tree-donorschoose-dataset#

tfidf_model = TfidfVectorizer()
# tfidf_model.fit(x_train_upsampled['preprocessed_essays'])
tfidf_model.fit(X_train['project_title'])
# 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())

X_train_project_title_tfidf_w2v_vectors = [] # the avg-w2v for each sentence/revi
# for sentence in tqdm(x_train_upsampled['preprocessed_essays']): # for each revi
for sentence in tqdm(X_train['project_title']): # 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 valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_project_title_tfidf_w2v_vectors.append(vector)
X_train_project_title_tfidf_w2v_vectors=sparse.csr_matrix(X_train_project_title_t

print("Total number of vectors  : TFIDF-W2V -> project_title: x_train : ",(X_trai
```

```
100%|██████████████████████████████████████████████████████████████████████████|
23450/23450 [00:00<00:00, 24313.07it/s]

Total number of vectors  : TFIDF-W2V -> project_title: x_train :  (23450, 300)
```

In [65]:
```
train_encoding1= hstack((X_train_project_title_tfidf_w2v_vectors,X_train_essays_t
```

In [66]:
```
test_encoding1= hstack((X_test_project_title_tfidf_w2v_vectors,X_test_essays_tfid
```

In [67]:
```
clf_train1= GridSearchCV(gb, param,cv=3,  scoring='roc_auc',return_train_score=Tr
clf_train1.fit(train_encoding1,Y_train)
```

Out[67]:
```
GridSearchCV(cv=3, estimator=GradientBoostingClassifier(),
             param_grid={'max_depth': [1, 2, 3], 'n_estimators': [1, 5, 10]},
             return_train_score=True, scoring='roc_auc')
```

In [68]:
```
mean_train_auc2= clf_train1.cv_results_['mean_train_score']
mean_test_auc2=clf_train1.cv_results_['mean_test_score']

std_train_auc2= clf_train1.cv_results_['std_train_score']
std_test_auc2= clf_train1.cv_results_['std_test_score']

print('mean_train_auc1',mean_train_auc2)
print('\n')
print('mean_test_auc1',mean_test_auc2)
print('\n')
print('std_train_auc1',std_train_auc2)
print('\n')
print('std_test_auc1',std_test_auc2)
```

```
mean_train_auc1 [0.56437025 0.64227479 0.66134409 0.6064097  0.67120236 0.69643
557
 0.63297684 0.69624694 0.72389986]


mean_test_auc1 [0.55586343 0.6249463  0.64368729 0.59112243 0.64775872 0.667490
06
 0.61432488 0.65808914 0.67868246]


std_train_auc1 [0.00796727 0.00662526 0.00878424 0.00703768 0.00476391 0.000630
99
 0.0046549  0.00220644 0.00501613]


std_test_auc1 [0.0082871  0.00626745 0.01227058 0.0067322  0.01015665 0.0062007
3
 0.00683942 0.00782888 0.00483607]
```

In [69]:
```
print('Best score: ',clf_train1.best_score_)
print('Best Hyper parameters: ',clf_train1.best_params_)
```

```
Best score:  0.6786824631767722
Best Hyper parameters:  {'max_depth': 3, 'n_estimators': 10}
```

In [70]:
```
clf_test1= GridSearchCV(gb, param,cv=3,  scoring='roc_auc',return_train_score=Tru
clf_test1.fit(test_encoding1,Y_test)
```

Out[70]:
```
GridSearchCV(cv=3, estimator=GradientBoostingClassifier(),
             param_grid={'max_depth': [1, 2, 3], 'n_estimators': [1, 5, 10]},
             return_train_score=True, scoring='roc_auc')
```

In [71]:
```
mean_train_auc3= clf_test1.cv_results_['mean_train_score']
mean_test_auc3=clf_test1.cv_results_['mean_test_score']

std_train_auc3= clf_test1.cv_results_['std_train_score']
std_test_auc3= clf_test1.cv_results_['std_test_score']

print('mean_train_auc1',mean_train_auc3)
print('\n')
print('mean_test_auc1',mean_test_auc3)
print('\n')
print('std_train_auc1',std_train_auc3)
print('\n')
print('std_test_auc1',std_test_auc3)
```

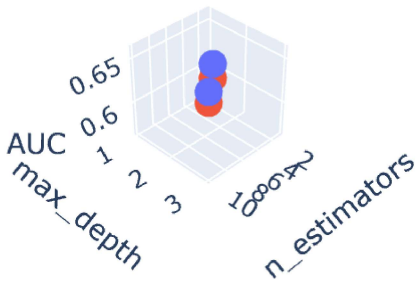
```
mean_train_auc1 [0.57419166 0.62606339 0.64994851 0.61404463 0.67204412 0.69763
423
 0.64477195 0.71212993 0.7424928 ]


mean_test_auc1 [0.5570715  0.59875443 0.61734665 0.58500907 0.62716566 0.645172
72
 0.5904968  0.62904124 0.64527251]


std_train_auc1 [0.0029711  0.00186897 0.0019235  0.00609174 0.00494381 0.001052
52
 0.00601039 0.00897946 0.0059646 ]


std_test_auc1 [0.00275821 0.00124282 0.00406527 0.00884737 0.01126278 0.0066027
 0.00744172 0.01399491 0.00753838]
```

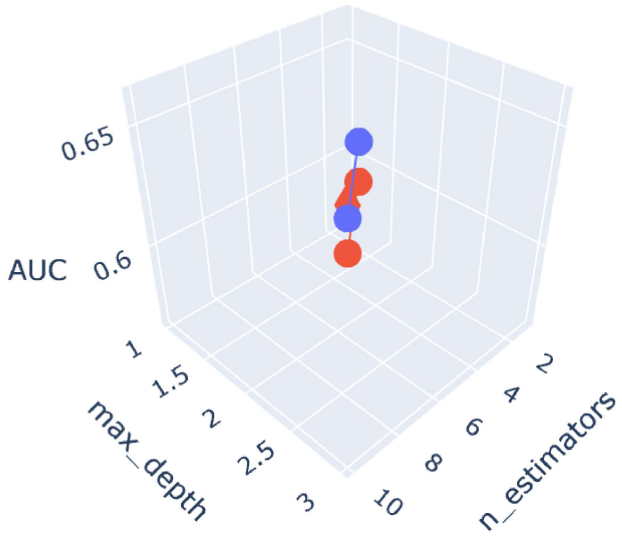
In [72]:
```
print('Best score: ',clf_test1.best_score_)
print('Best Hyper parameters: ',clf_test1.best_params_)
```

```
Best score:  0.6452725106724323
Best Hyper parameters:  {'max_depth': 3, 'n_estimators': 10}
```

In [73]:
```
x1 = [1,5,10]
y1 = [1,2,3]
z1 = mean_train_auc2

x2 =[1,5,10]
y2 = [1,2,3]
z2 = mean_train_auc3
```

In [74]:
```
# 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')
fig.show()
plt.show()
```

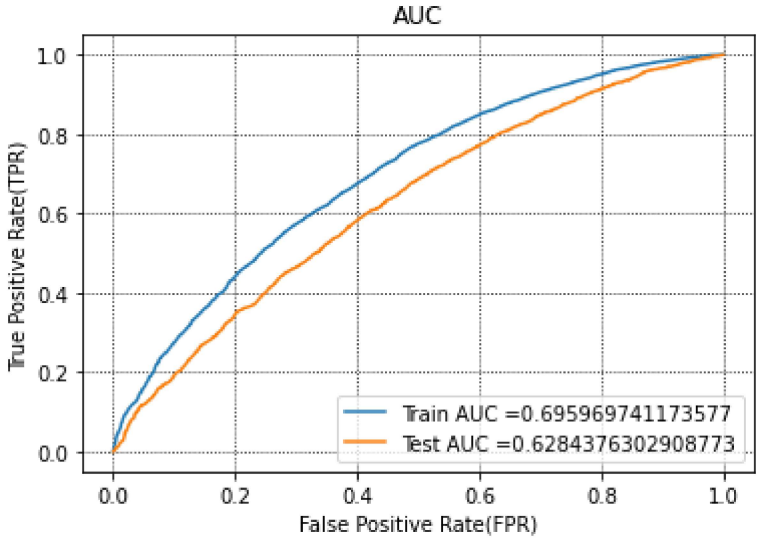





In [75]:
```
bestEstimators= clf.best_params_['n_estimators']
print(bestEstimators)
bestMaxDepth=clf.best_params_['max_depth']
print(bestMaxDepth)
```

```
10
3
```

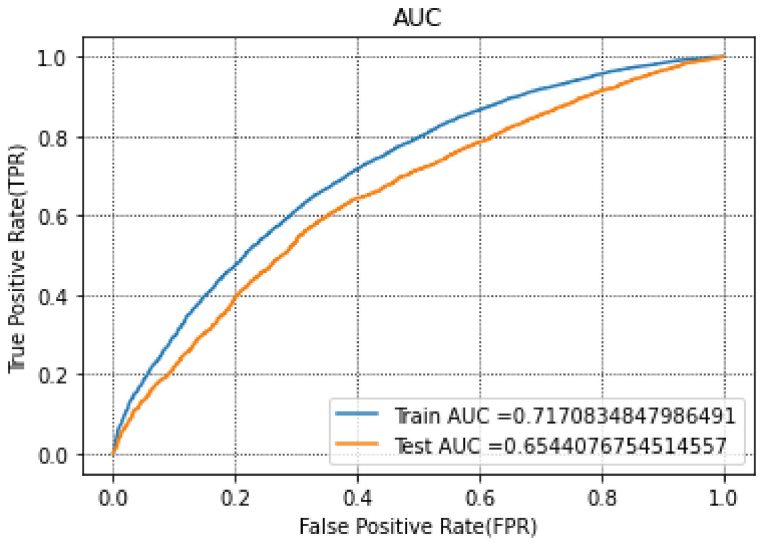
In [76]:
```
from sklearn.metrics import roc_curve, auc

model1 =  GradientBoostingClassifier(max_depth=bestMaxDepth,n_estimators=bestEsti
model1.fit(train_encoding, Y_train)

y_train_pred=model1.predict_proba(train_encoding)[:,1]

y_test_pred=model1.predict_proba(test_encoding)[:,1]



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)

ax = plt.subplot()

auc_set1_train=auc(train_fpr, train_tpr)
auc_set1_test=auc(test_fpr, test_tpr)


ax.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
ax.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid(b=True, which='major', color='k', linestyle=':')
ax.set_facecolor("white")
plt.show()
```




In [77]:
```
bestEstimators1= clf_train1.best_params_['n_estimators']
print(bestEstimators1)
bestMaxDepth1=clf_train1.best_params_['max_depth']
print(bestMaxDepth1)
```

```
10
3
```

In [78]:
```
from sklearn.metrics import roc_curve, auc
model1 =  GradientBoostingClassifier(max_depth=bestMaxDepth1,n_estimators=bestEst
model1.fit(train_encoding1, Y_train)

y_train_pred=model1.predict_proba(train_encoding1)[:,1]

y_test_pred=model1.predict_proba(test_encoding1)[:,1]



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)

ax = plt.subplot()

auc_set3_train=auc(train_fpr, train_tpr)
auc_set3_test=auc(test_fpr, test_tpr)


ax.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
ax.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid(b=True, which='major', color='k', linestyle=':')
ax.set_facecolor("white")
plt.show()
```

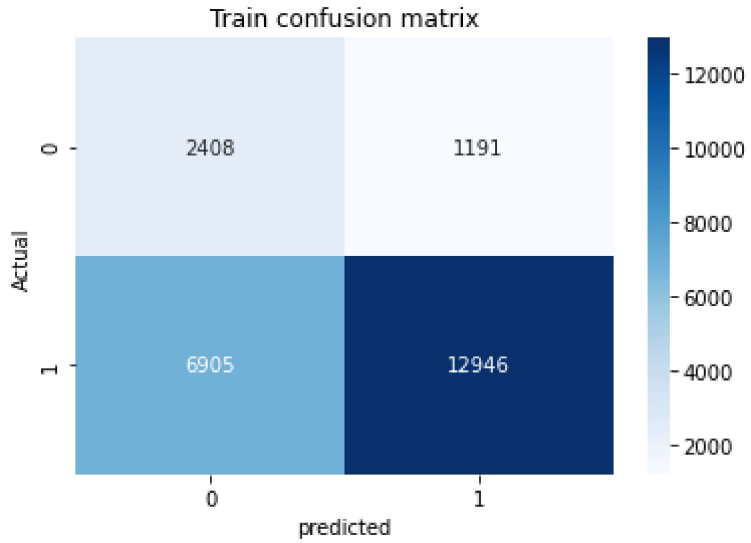


In [79]:
```
def maxthreshold(threshold, fpr, tpr):

    '''
    Calculating maximum threshold
    returning the threshold value

    '''
    t =  threshold[np.argmax(tpr*(1-fpr))]
    #best threshold when 1-fpr is less

    print("the maximum value of tpr*(1-fpr)",max(tpr*(1-fpr)), "for threshold", t
    return t
```

In [80]:
```
def predict(proba, thresh):
    '''
    Predicting
    '''

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

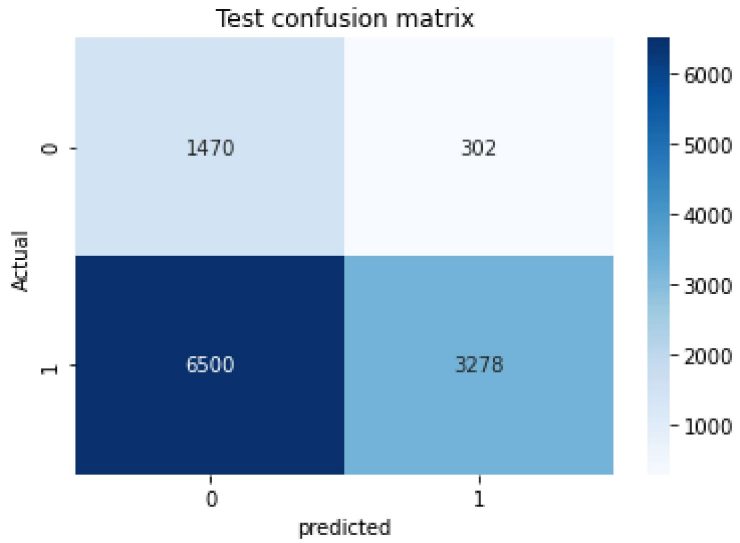
In [81]:
```
#confusion matrix
cf=confusion_matrix
thresh = maxthreshold(tr_thresholds,train_fpr, train_tpr) #best threshold
conf_matr_df_train = cf(Y_train, predict(y_train_pred,thresh))

sns.heatmap(conf_matr_df_train, annot=True,fmt="d",cmap='Blues')
plt.xlabel('predicted')
plt.ylabel('Actual')
plt.title('Train confusion matrix')
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.43634283525630774 for threshold 0.8415788948
772499
```



In [82]:
```
thresh = maxthreshold(tr_thresholds,test_fpr, test_tpr)
conf_matr_df_test = cf(Y_test, predict(y_test_pred,thresh))

sns.heatmap(conf_matr_df_test, annot=True,fmt="d",cmap='Blues')
plt.xlabel('predicted')
plt.ylabel('Actual')
plt.title('Test confusion matrix')
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.3906949862569818 for threshold 0.8607490686427237
```




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

# 3. Summary

as mentioned in the step 4 of instructions

In [83]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyperparameter: Estimators", "Hyperparar
x.add_row(['tf-idf','GBDT',clf.best_params_['n_estimators'], clf.best_params_['ma
x.add_row(['tf-idf-w2v','GBDT',clf_train1.best_params_['n_estimators'], clf_trair
print(x)
```

```
+------------+-------+---------------------------+---------------------------+
--------------------+--------------------+
| Vectorizer | Model | Hyperparameter: Estimators | Hyperparameter: max depth |
Train AUC      |      Test AUC      |
+------------+-------+---------------------------+---------------------------+
--------------------+--------------------+
|   tf-idf   |  GBDT |             10            |             3             |
0.695969741173577  | 0.6284376302908773 |
| tf-idf-w2v |  GBDT |             10            |             3             |
0.7170834847986491 | 0.6544076754514557 |
+------------+-------+---------------------------+---------------------------+
--------------------+--------------------+
```