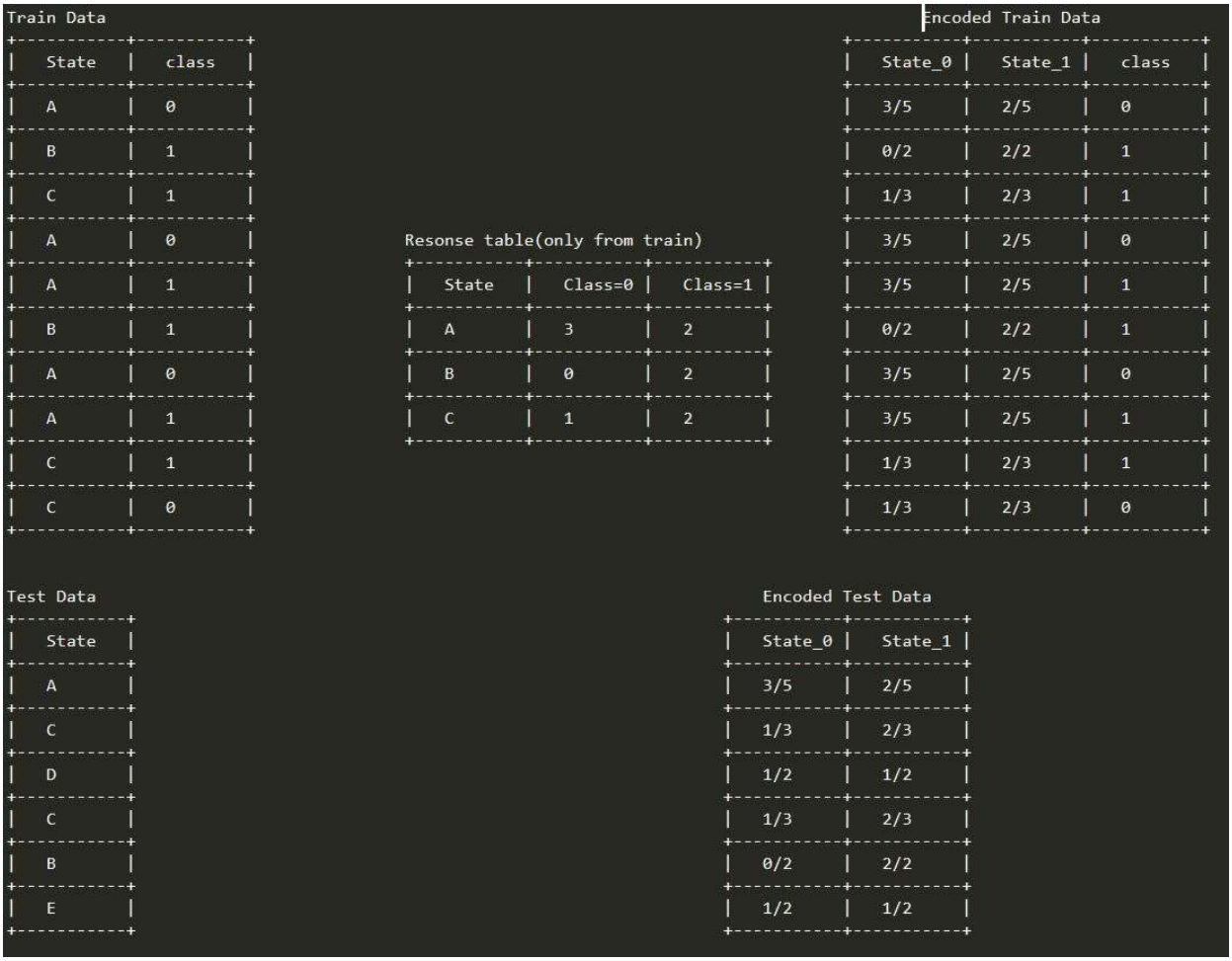
Assignment 9: GBDT

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

Resone 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/): 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)

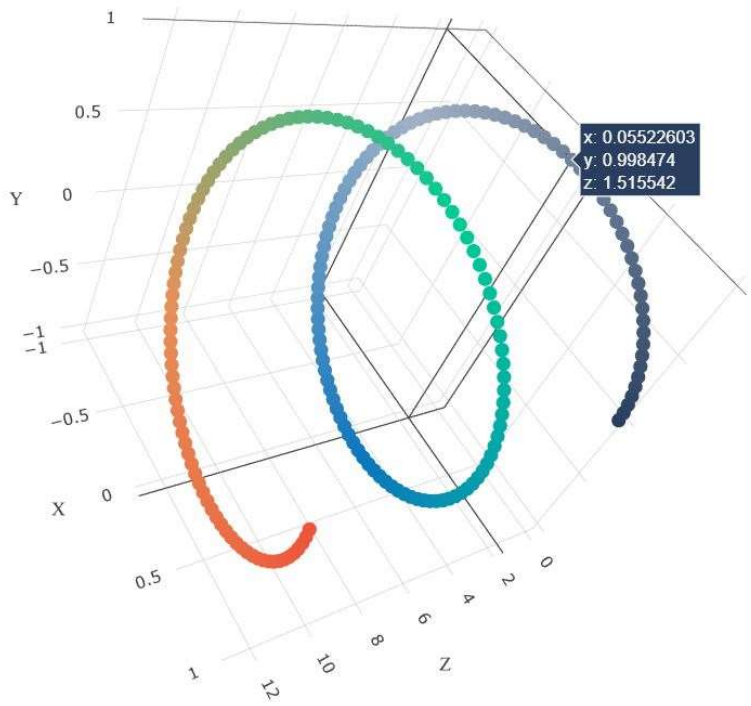
- Here in response encoding you need to apply the **laplase smoothing** value for test set. Laplase smoothing means, If test point is present in test but not in train then you need to apply default 0.5 as probability value for that data point (Refer the Response Encoding Image from above cell)
- Please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

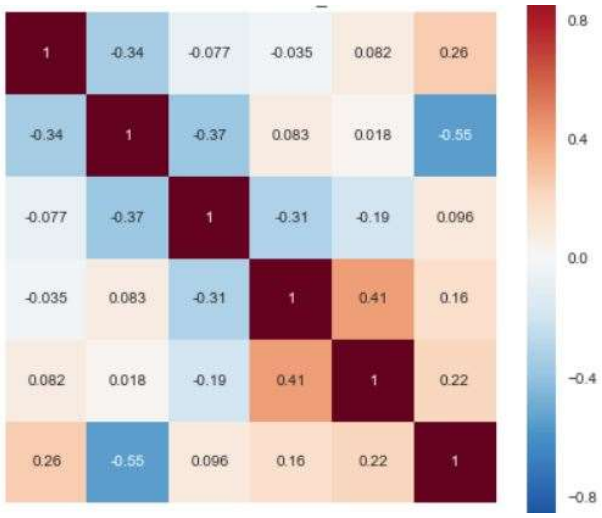
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



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*

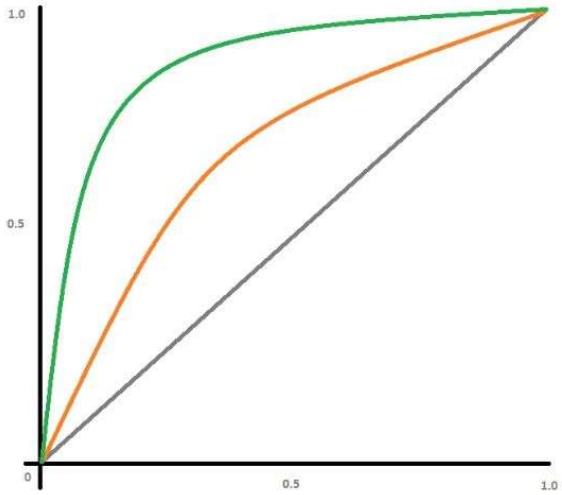
or

- 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 [2]:
```
import os
os.getcwd()
```

Out[2]:
```
'C:\\Users\\amakh'
```

In [3]:
```
os.chdir("C:\\Users\\amakh\\Downloads")
```

In [4]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import warnings
warnings.filterwarnings("ignore")
# 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}

C:\Users\amakh\Anaconda3\lib\site-packages\nltk\twitter\__init__.py:20: UserWar
ning: The twython library has not been installed. Some functionality from the t
witter package will not be available.
  warnings.warn("The twython library has not been installed. "
```

In [5]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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
```

In [6]:
```
#Loading 'glove_vectors
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

1. GBDT (xgboost/lightgbm)

1.1 Loading Data

In [7]:
```
#Loading dataset
import pandas
data = pandas.read_csv('preprocessed_data.csv')
data.head()
```

Out[7]:

	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	
3	ga	mrs	grades_prek_2	
4	wa	mrs	grades_3_5	

In [8]:
```
#Checking the column names for the dataset
data.columns
```

Out[8]:
```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

In [9]:
```
#Calculating sentiment scores using vader lexicon
import nltk
nltk.download('vader_lexicon')

import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

sid = SentimentIntensityAnalyzer()

#initializing empty set for each of the polarities
pos = []
neg = []
neu = []
compound = []

for i in tqdm(range(len(data['essay']))):
    sentiment_scores = sid.polarity_scores(data['essay'][i])
    neg.append(sentiment_scores['neg'])
    pos.append(sentiment_scores['pos'])
    neu.append(sentiment_scores['neu'])
    compound.append(sentiment_scores['compound'])
```

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

100%|██████████████████████████████████████████████████████████████████████████████|
109248/109248 [04:08<00:00, 438.85it/s]
```

In [10]:
```
#Adding new columns on the dataset for each of the sentiment scores for feature :

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

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

In [74]:
```
#Taking "project_is_approved" as y and all other features as x

y = data["project_is_approved"].values
x = data
```

In [75]:
```
from sklearn.model_selection import train_test_split
#Taking test data as 30%
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, stratify

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

```
(76473, 13) (76473,)
(32775, 13) (32775,)
```

1.3 Make Data Model Ready: encoding eassay

1.3.1 Applying TFIDF vectorizer on essay feature

In [76]:
```
#TFIDF vectozier on text data
essay_tfidf_vectorizer = TfidfVectorizer(min_df= 10,max_features= 5000)
essay_tfidf_vectorizer.fit(x_train['essay'].values)

#converting text to vector using vectorizer
x_train_essay_tfidf = essay_tfidf_vectorizer.transform(x_train['essay'].values)
x_test_essay_tfidf = essay_tfidf_vectorizer.transform(x_test['essay'].values)

#Checking if the train and test data size remains same after applying encoding

print(x_train_essay_tfidf.shape)
print(x_test_essay_tfidf.shape)
```

```
(76473, 5000)
(32775, 5000)
```

1.3.2 Applying TFIDF W2V on essay feature

In [77]:
```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(x_train["essay"].values)
#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 [78]:
```
# average Word2Vec
# computing average word2vec for each review in x_train
tfidf_w2v_essay_x_train = []; # the avg-w2v for each sentence/review is stored in
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
    tfidf_w2v_essay_x_train.append(vector)

print(len(tfidf_w2v_essay_x_train))
print(len(tfidf_w2v_essay_x_train[0]))
```

```
100%|
| 76473/76473 [04:48<00:00, 264.63it/s]

76473
300
```

In [79]:
```
# average Word2Vec
# computing average word2vec for each review in x_test
tfidf_w2v_essay_x_test = []; # the avg-w2v for each sentence/review is stored in
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
    tfidf_w2v_essay_x_test.append(vector)

print(len(tfidf_w2v_essay_x_test))
print(len(tfidf_w2v_essay_x_test[0]))
```

```
100%|
| 32775/32775 [02:09<00:00, 252.61it/s]

32775
300
```

1.4 Make Data Model Ready: encoding numerical, categorical features

1.4.1 Encoding of numerical features : teacher_number_of_previously_posted_projects

In [80]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Standard:

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaler.fit(x_train['teacher_number_of_previously_posted_projects'].values.reshape
x_train_no_pre_projects_scaled = scaler.fit_transform(x_train['teacher_number_of_
x_test_no_pre_projects_scaled = scaler.transform(x_test['teacher_number_of_previo

#Checking if the train and test data size remains same after applying scaling

print(x_train_no_pre_projects_scaled.shape)
print(x_test_no_pre_projects_scaled.shape)
```

```
(76473, 1)
(32775, 1)

C:\Users\amakh\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: Dat
aConversionWarning: Data with input dtype int64 was converted to float64 by Sta
ndardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\amakh\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: Dat
aConversionWarning: Data with input dtype int64 was converted to float64 by Sta
ndardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\amakh\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: Dat
aConversionWarning: Data with input dtype int64 was converted to float64 by Sta
ndardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\amakh\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: Dat
aConversionWarning: Data with input dtype int64 was converted to float64 by Sta
ndardScaler.
  warnings.warn(msg, DataConversionWarning)
```

1.4.2 Encoding of numerical features : price

In [81]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Standard

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaler.fit(x_train['price'].values.reshape(-1,1))
x_train_price_scaled = scaler.fit_transform(x_train['price'].values.reshape(-1,1)
x_test_price_scaled = scaler.transform(x_test['price'].values.reshape(-1,1))

#Checking if the train and test data size remains same after applying scaling

print(x_train_price_scaled.shape)
print(x_test_price_scaled.shape)
```

```
(76473, 1)
(32775, 1)
```

1.4.3 Encoding of numerical features : Sentiment scores

In [82]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Standard

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaler.fit(x_train['pos'].values.reshape(-1,1))
x_train_pos_scaled = scaler.fit_transform(x_train['pos'].values.reshape(-1,1))
x_test_pos_scaled = scaler.transform(x_test['pos'].values.reshape(-1,1))

#Checking if the train and test data size remains same after applying scaling

print(x_train_pos_scaled.shape)
print(x_test_pos_scaled.shape)
```

```
(76473, 1)
(32775, 1)
```

In [83]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Standard

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaler.fit(x_train['neg'].values.reshape(-1,1))
x_train_neg_scaled = scaler.fit_transform(x_train['neg'].values.reshape(-1,1))
x_test_neg_scaled = scaler.transform(x_test['neg'].values.reshape(-1,1))

#Checking if the train and test data size remains same after applying scaling

print(x_train_neg_scaled.shape)
print(x_test_neg_scaled.shape)
```

```
(76473, 1)
(32775, 1)
```

In [84]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Standard

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaler.fit(x_train['neu'].values.reshape(-1,1))
x_train_neu_scaled = scaler.fit_transform(x_train['neu'].values.reshape(-1,1))
x_test_neu_scaled = scaler.transform(x_test['neu'].values.reshape(-1,1))

#Checking if the train and test data size remains same after applying scaling

print(x_train_neu_scaled.shape)
print(x_test_neu_scaled.shape)
```

```
(76473, 1)
(32775, 1)
```

In [85]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Standard

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

scaler.fit(x_train['compound'].values.reshape(-1,1))
x_train_compound_scaled = scaler.fit_transform(x_train['compound'].values.reshape
x_test_compound_scaled = scaler.transform(x_test['compound'].values.reshape(-1,1)

#Checking if the train and test data size remains same after applying scaling

print(x_train_compound_scaled.shape)
print(x_test_compound_scaled.shape)
```

```
(76473, 1)
(32775, 1)
```

1.4.4 Encoding of categorical features : school_state

In [86]:
```
#https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-cours
#using response encoding for categorical variables

#Code for response encoding
def get_fea_dict(feature, df):
    value_count = x_train[feature].value_counts()
    #creating dictionary which contains probability of each feature
    fea_dict = dict()
    for i, denominator in value_count.items():
        vec = []
        for k in range(1,3):
            class_count = x_train.loc[(x_train['project_is_approved']==k) & (x_tr
            vec.append((class_count.shape[0])/ (denominator))
        fea_dict[i]=vec
    return fea_dict

def get_feature(feature, df):
    fea_dict = get_fea_dict(feature, df)
    value_count = x_train[feature].value_counts()

    fea_val = []
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            fea_val.append(fea_dict[row[feature]])
        else:
            #giving probability of 1/2 if the feature is not in train data but in
            fea_val.append([1/2,1/2])
    return fea_val
```

In [87]:
```
#using response encoding for categorical variables

x_train_school_state_encoded = np.array(get_feature("school_state", x_train))
x_test_school_state_encoded = np.array(get_feature("school_state", x_test))

#Checking if the train and test data size remains same after applying response en

print(x_train_school_state_encoded.shape)
print(x_test_school_state_encoded.shape)
```

```
(76473, 2)
(32775, 2)
```

1.4.5 Encoding of categorical features : teacher_prefix

In [88]:
```
#using response encoding for categorical variables

x_train_teacher_prefix_encoded = np.array(get_feature("teacher_prefix", x_train))
x_test_teacher_prefix_encoded = np.array(get_feature("teacher_prefix", x_test))

#Checking if the train and test data size remains same after applying response en

print(x_train_teacher_prefix_encoded.shape)
print(x_test_teacher_prefix_encoded.shape)
```

```
(76473, 2)
(32775, 2)
```

1.4.6 Encoding of categorical features : project_grade_category

In [89]:
```
#using response encoding for categorical variables

x_train_project_grade_category_encoded = np.array(get_feature("project_grade_cate
x_test_project_grade_category_encoded = np.array(get_feature("project_grade_categ

#Checking if the train and test data size remains same after applying response en

print(x_train_project_grade_category_encoded.shape)
print(x_test_project_grade_category_encoded.shape)
```

```
(76473, 2)
(32775, 2)
```

1.4.7 Encoding of categorical features : clean_categories

In [90]:
```
#using response encoding for categorical variables

x_train_clean_categories_encoded = np.array(get_feature("clean_categories", x_tra
x_test_clean_categories_encoded = np.array(get_feature("clean_categories", x_test

#Checking if the train and test data size remains same after applying response en

print(x_train_clean_categories_encoded.shape)
print(x_test_clean_categories_encoded.shape)
```

```
(76473, 2)
(32775, 2)
```

1.4.8 Encoding of categorical features : clean_subcategories

In [91]:
```
#using response encoding for categorical variables

x_train_clean_subcategories_encoded = np.array(get_feature("clean_subcategories",
x_test_clean_subcategories_encoded = np.array(get_feature("clean_subcategories",

#Checking if the train and test data size remains same after applying response en

print(x_train_clean_subcategories_encoded.shape)
print(x_test_clean_subcategories_encoded.shape)
```

```
(76473, 2)
(32775, 2)
```

1.5 Concatenating features : Set 1 (categorical, numerical features + essay (TFIDF) + Sentiment scores(essay))

In [29]:
```
#Using hstack to conact the features

from scipy.sparse import hstack
x_train_set1 = hstack((x_train_essay_tfidf,x_train_no_pre_projects_scaled,x_train
x_test_set1 = hstack((x_test_essay_tfidf,x_test_no_pre_projects_scaled,x_test_pri

print("Concated Data Matrix")

print(x_train_set1.shape,y_train.shape)
print(x_test_set1.shape,y_test.shape)
```

```
Concated Data Matrix
(76473, 5016) (76473,)
(32775, 5016) (32775,)
```

1.6 hyper paramter tuning on Set 1

In [50]:
```
#https://www.youtube.com/watch?v=9HomdnM12o4
#https://blog.cambridgespark.com/hyperparameter-tuning-in-xgboost-4ff9100a3b2f
#https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgb

#Using Xgboost Classifier

from sklearn.model_selection import GridSearchCV
import xgboost

GBDT_1 = xgboost.XGBClassifier()

#Defining parameters
params = {
"n_estimators": [5,10,25,50],
"max_depth": [1,5,10,50],
}

#UsingGridsearch_CV
clf_1 = GridSearchCV(GBDT_1, params, cv=3, scoring='roc_auc',return_train_score =
clf_1.fit(x_train_set1,y_train)
```

```
[07:23:29] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
tric used with the objective 'binary:logistic' was changed from 'error' to 'l
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.

C:\Users\amakh\Anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarni
ng: The use of label encoder in XGBClassifier is deprecated and will be remov
ed in a future release. To remove this warning, do the following: 1) Pass opt
ion use_label_encoder=False when constructing XGBClassifier object; and 2) En
code your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_cla
ss - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[07:23:31] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
tric used with the objective 'binary:logistic' was changed from 'error' to 'l
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.
```

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 [52]:
```
#interpreting the results of GridSearchCV
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSe

print("best score :",clf_1.best_score_)
print("best parameter:",clf_1.best_params_)
```

```
best score : 0.7186845110175968
best parameter: {'max_depth': 5, 'n_estimators': 50}
```

In [53]:
```
#Checking the Train and Test AUC scores
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSe

print('Train AUC scores:')
print(clf_1.cv_results_['mean_train_score'])

print('Test AUC scores:')
print(clf_1.cv_results_['mean_test_score'])
```

```
Train AUC scores:
[0.6207129  0.65081959 0.68203604 0.70543875 0.70824647 0.74647312
 0.80562046 0.85699326 0.83317696 0.89343293 0.9583716  0.98383182
 0.9992703  0.99999952 1.         1.        ]
Test AUC scores:
[0.61714115 0.6456551  0.67588972 0.69680209 0.67157762 0.6900368
 0.71116451 0.71868451 0.66971238 0.69028595 0.70554669 0.70896742
 0.62003875 0.65696977 0.6926535  0.7039984 ]
```

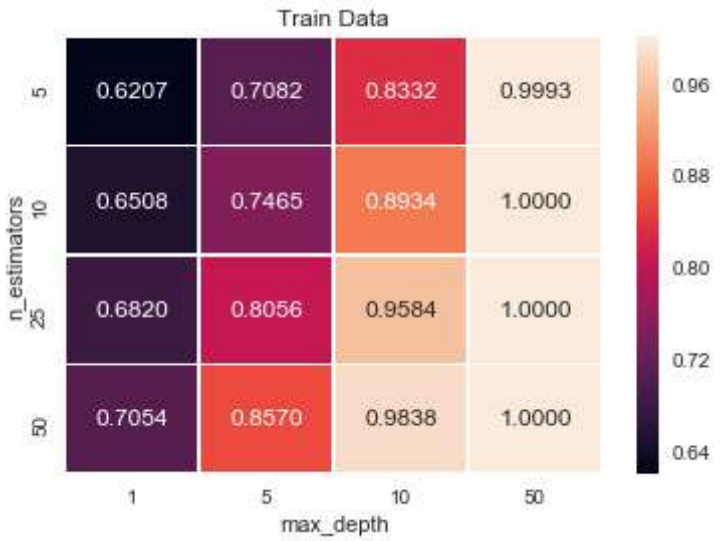
In [54]:
```
#creating dataframe to summarize the results of GridSearchCV
results = pd.DataFrame.from_dict(clf_1.cv_results_)
results.head()
```

Out[54]:

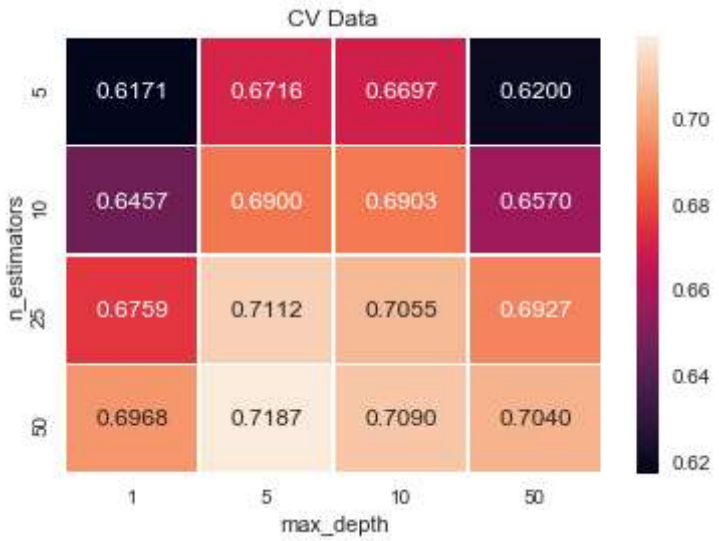
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_est
0	1.209001	0.064011	0.045700	0.001685	1	
1	2.064865	0.051182	0.053936	0.005727	1	
2	6.303310	2.192891	0.077714	0.031062	1	
3	12.469689	1.581122	0.079868	0.012331	1	

1.6.1 Plotting the results using heatmap

In [57]:
```
#heatmaps with rows as min_sample_split, columns as max_depth, and values inside
#https://seaborn.pydata.org/generated/seaborn.heatmap.html

np.random.seed(0)
sns.set()

#heatmap on train data
heatmap_data = pd.DataFrame({'n_estimators': results["param_n_estimators"], 'max_
heatmap_data = heatmap_data.pivot("n_estimators","max_depth","Z")
hm= sns.heatmap(heatmap_data, annot= True, fmt= ".4f",linewidths=.5)

plt.title("Train Data")
plt.show()
```



In [56]:
```
#heatmap on cv data
heatmap_data = pd.DataFrame({'n_estimators': results["param_n_estimators"], 'max_
heatmap_data = heatmap_data.pivot("n_estimators","max_depth","Z")
hm= sns.heatmap(heatmap_data, annot= True, fmt= ".4f",linewidths=.5)

plt.title("CV Data")
plt.show()
```



1.6.2 Finding best parameters and fit the model

In [59]:
```
#Finding the best parameters from classifier clf_1

best_max_depth= clf_1.best_params_['max_depth']
best_n_estimators= clf_1.best_params_['n_estimators']

print("best_max_depth= ",best_max_depth)
print("best_n_estimators= ",best_n_estimators)
```

```
best_max_depth=  5
best_n_estimators=  50
```

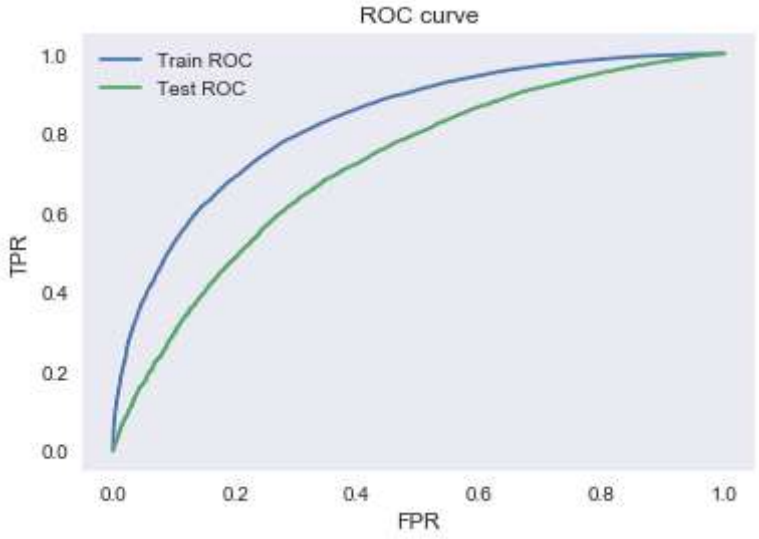
In [60]:
```
GBDT_set1 = xgboost.XGBClassifier(max_depth= best_max_depth, n_estimators= best_n
GBDT_set1.fit(x_train_set1,y_train)
```

```
C:\Users\amakh\Anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarnin
g: The use of label encoder in XGBClassifier is deprecated and will be removed
in a future release. To remove this warning, do the following: 1) Pass option u
se_label_encoder=False when constructing XGBClassifier object; and 2) Encode yo
ur labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[08:28:59] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[60]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
       gamma=0, gpu_id=-1, importance_type=None,
       interaction_constraints='', learning_rate=0.300000012,
       max_delta_step=0, max_depth=5, min_child_weight=1, missing=nan,
       monotone_constraints='()', n_estimators=50, n_jobs=8,
       num_parallel_tree=1, objective='binary:logistic', predictor='auto',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       subsample=1, tree_method='exact', use_label_encoder=True,
       validate_parameters=1, verbosity=None)
```

1.6.3 Plotting ROC-AUC curve using predict proba method

In [63]:
```
y_train_predicted = GBDT_set1.predict_proba(x_train_set1)[:,1]
y_test_predicted = GBDT_set1.predict_proba(x_test_set1)[:,1]
ytest_predicted = GBDT_set1.predict(x_test_set1)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_predicted)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_predicted)

plt.plot(train_fpr, train_tpr,label="Train ROC")
plt.plot(test_fpr, test_tpr,label="Test ROC")
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```



In [64]:
```
#Finding the area under curve ROC = AUC

train_AUC = auc(train_fpr, train_tpr)
test_AUC = auc(test_fpr, test_tpr)

print("AUC for train data :",train_AUC)
print("AUC for test data :",test_AUC)
```

```
AUC for train data : 0.8288230614790166
AUC for test data : 0.7190109441826948
```

1.6.4 Confusion Matrix

In [65]:
```
from sklearn.metrics import confusion_matrix
cm_test = confusion_matrix(y_test, ytest_predicted)
sns.heatmap(cm_test, annot=True,fmt="d",xticklabels= ['Predicted No', 'Predicted
```

Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x197fff27eb8>

In [66]:
```
#Getting all the false positive values

false_positive_data = []
for i in range (len(y_test)):
    if (y_test[i] == 0) & (ytest_predicted[i] == 1):
        false_positive_data.append(i)

#checking if the false positive value count matches with that on confusion matrix
print(len(false_positive_data))
```

4757

In [67]:
```
#getting the corresponding false positive words from "essay"
false_positive_essay1= []
for i in false_positive_data:
    false_positive_essay1.append(x_test['essay'].values[i])

#checking if the false positive value count matches with that on confusion matrix
print(len(false_positive_essay1))
```

4757

1.7 Concatenating features : Set 2 (categorical, numerical features + essay (TFIDF W2V))

In [104]:
```
#Using scipy.sparse.csr_matrix tfidf-w2v vectors into sparse
#hstack needs at least one input as sparse
from scipy.sparse import csr_matrix

tfidf_w2v_essay_x_train = csr_matrix((76473,5000))
tfidf_w2v_essay_x_test = csr_matrix((32775, 5000))
```

In [105]:
```
#Using hstack to conact the features
from scipy.sparse import hstack
x_train_set2 = hstack((tfidf_w2v_essay_x_train,x_train_no_pre_projects_scaled,x_t
x_test_set2 = hstack((tfidf_w2v_essay_x_test,x_test_no_pre_projects_scaled,x_test

print("Concated Data Matrix")

print(x_train_set2.shape,y_train.shape)
print(x_test_set2.shape,y_test.shape)
```

```
Concated Data Matrix
(76473, 5012) (76473,)
(32775, 5012) (32775,)
```

1.8 hyper paramter tuning on Set 1

In [106]:
```
#https://www.youtube.com/watch?v=9HomdnM12o4
#https://blog.cambridgespark.com/hyperparameter-tuning-in-xgboost-4ff9100a3b2f
#https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgb

#Using Xgboost Classifier

from sklearn.model_selection import GridSearchCV
import xgboost

GBDT_2 = xgboost.XGBClassifier()

#Defining parameters
params = {
"n_estimators": [5,10,25,50],
"max_depth": [1,5,10,50],
}

#UsingGridsearch_CV
clf_2 = GridSearchCV(GBDT_2, params, cv=3, scoring='roc_auc',return_train_score =
clf_2.fit(x_train_set2,y_train)
```

```
ng: The use of label encoder in XGBClassifier is deprecated and will be remov
ed in a future release. To remove this warning, do the following: 1) Pass opt
ion use_label_encoder=False when constructing XGBClassifier object; and 2) En
code your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_cla
ss - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[15:43:32] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.
5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation me
tric used with the objective 'binary:logistic' was changed from 'error' to 'l
ogloss'. Explicitly set eval_metric if you'd like to restore the old behavio
r.



C:\Users\amakh\Anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarni
ng: The use of label encoder in XGBClassifier is deprecated and will be remov
ed in a future release. To remove this warning, do the following: 1) Pass opt
```

In [107]:
```
#interpreting the results of GridSearchCV
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSe

print("best score :",clf_2.best_score_)
print("best parameter:",clf_2.best_params_)
```

```
best score : 0.6347250416667848
best parameter: {'max_depth': 5, 'n_estimators': 25}
```

In [108]:
```
#Checking the Train and Test AUC scores
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSe

print('Train AUC scores:')
print(clf_2.cv_results_['mean_train_score'])

print('Test AUC scores:')
print(clf_2.cv_results_['mean_test_score'])
```

```
Train AUC scores:
[0.61930291 0.62559688 0.63634194 0.64025365 0.64924816 0.65967252
 0.67529117 0.70359241 0.73942544 0.78764546 0.83671868 0.88667964
 0.95624304 0.99647725 0.99998737 0.99999947]
Test AUC scores:
[0.61670602 0.62240417 0.63148451 0.63449015 0.63138885 0.63452944
 0.63472504 0.63310521 0.62123341 0.61913346 0.61768173 0.61352368
 0.5850995  0.58207309 0.58344726 0.58416482]
```

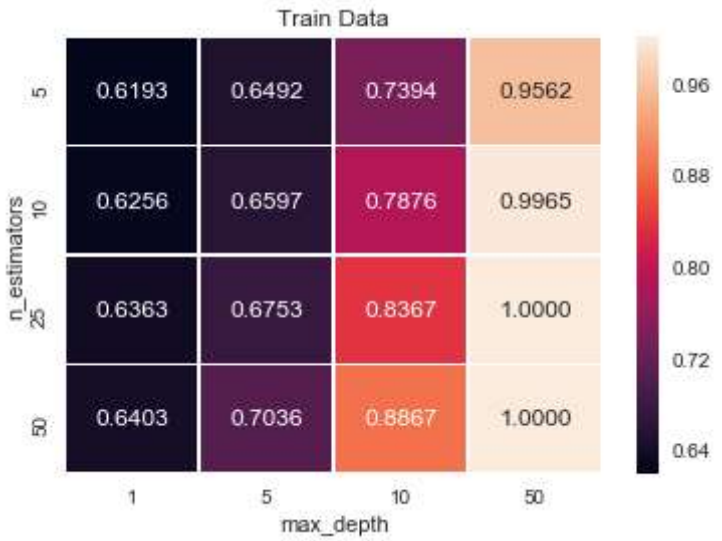
In [109]:
```
#creating dataframe to summarize the results of GridSearchCV
results = pd.DataFrame.from_dict(clf_2.cv_results_)
results.head()
```

Out[109]:

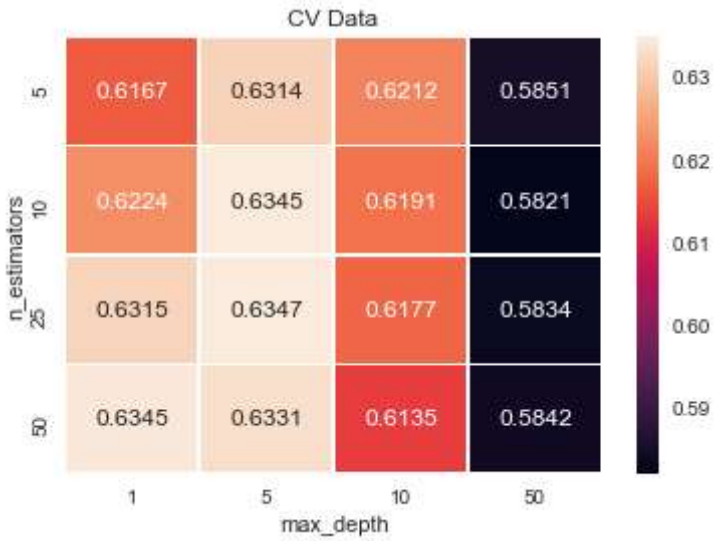
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_est
0	0.108610	0.044812	0.020839	0.007368	1	
1	0.113485	0.010761	0.012006	0.003435	1	
2	0.226642	0.001252	0.018078	0.002233	1	
3	0.417833	0.002677	0.020839	0.007367	1	

1.8.1 Plotting the results using heatmap

In [111]:
```
#heatmaps with rows as min_sample_split, columns as max_depth, and values inside t
#https://seaborn.pydata.org/generated/seaborn.heatmap.html

np.random.seed(0)
sns.set()

#heatmap on train data
heatmap_data2 = pd.DataFrame({'n_estimators': results["param_n_estimators"], 'max_
heatmap_data2 = heatmap_data2.pivot("n_estimators","max_depth","Z")
hm= sns.heatmap(heatmap_data2, annot= True, fmt= ".4f",linewidths=.5)

plt.title("Train Data")
plt.show()
```



In [113]:
```
#heatmap on cv data
heatmap_data2 = pd.DataFrame({'n_estimators': results["param_n_estimators"], 'max
heatmap_data2= heatmap_data2.pivot("n_estimators","max_depth","Z")
hm= sns.heatmap(heatmap_data2, annot= True, fmt= ".4f",linewidths=.5)

plt.title("CV Data")
plt.show()
```



1.8.2 Finding best parameters and fit the model

In [114]:
```
#Finding the best parameters from classifier clf_1

best_max_depth= clf_2.best_params_['max_depth']
best_n_estimators= clf_2.best_params_['n_estimators']

print("best_max_depth= ",best_max_depth)
print("best_n_estimators= ",best_n_estimators)
```

```
best_max_depth=  5
best_n_estimators=  25
```

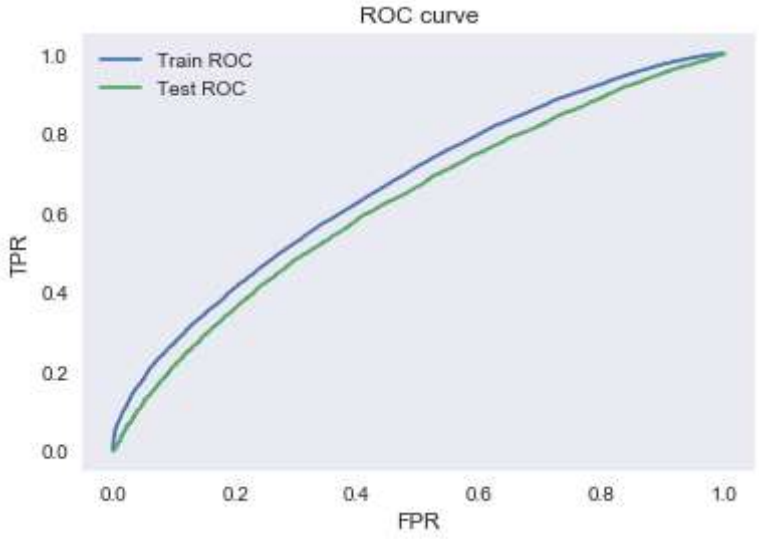
In [115]:
```
GBDT_set2 = xgboost.XGBClassifier(max_depth= best_max_depth, n_estimators= best_n
GBDT_set2.fit(x_train_set2,y_train)
```

```
C:\Users\amakh\Anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarnin
g: The use of label encoder in XGBClassifier is deprecated and will be removed
in a future release. To remove this warning, do the following: 1) Pass option u
se_label_encoder=False when constructing XGBClassifier object; and 2) Encode yo
ur labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)


[16:28:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.
1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[115]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
       gamma=0, gpu_id=-1, importance_type=None,
       interaction_constraints='', learning_rate=0.300000012,
       max_delta_step=0, max_depth=5, min_child_weight=1, missing=nan,
       monotone_constraints='()', n_estimators=25, n_jobs=8,
       num_parallel_tree=1, objective='binary:logistic', predictor='auto',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       subsample=1, tree_method='exact', use_label_encoder=True,
       validate_parameters=1, verbosity=None)
```

1.8.3 Plotting ROC-AUC curve using predict proba method

In [116]:
```
y_train_predicted = GBDT_set2.predict_proba(x_train_set2)[:,1]
y_test_predicted = GBDT_set2.predict_proba(x_test_set2)[:,1]
ytest_predicted = GBDT_set2.predict(x_test_set2)

train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_predicted)
test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_predicted)

plt.plot(train_fpr, train_tpr,label="Train ROC")
plt.plot(test_fpr, test_tpr,label="Test ROC")
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```



In [117]:
```
#Finding the area under curve ROC = AUC

train_AUC = auc(train_fpr, train_tpr)
test_AUC = auc(test_fpr, test_tpr)

print("AUC for train data :",train_AUC)
print("AUC for test data :",test_AUC)
```

```
AUC for train data : 0.6662716941869312
AUC for test data : 0.6231782238760992
```

1.8.4 Confusion Matrix

In [118]:
```
from sklearn.metrics import confusion_matrix
cm_test = confusion_matrix(y_test, ytest_predicted)
sns.heatmap(cm_test, annot=True,fmt="d",xticklabels= ['Predicted No', 'Predicted
```

Out[118]: <matplotlib.axes._subplots.AxesSubplot at 0x197bfc02ef0>

In [119]:
```
#Getting all the false positive values

false_positive_data = []
for i in range (len(y_test)):
    if (y_test[i] == 0) & (ytest_predicted[i] == 1):
        false_positive_data.append(i)

#checking if the false positive value count matches with that on confusion matrix
print(len(false_positive_data))
```

4953

In [120]:
```
#getting the corresponding false positive words from "essay"
false_positive_essay1= []
for i in false_positive_data:
    false_positive_essay1.append(x_test['essay'].values[i])

#checking if the false positive value count matches with that on confusion matrix
print(len(false_positive_essay1))
```

4953

3. Summary

In [122]:
```
#Tabulating the results using PtrettyTable
#https://pypi.org/project/prettytable/

from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Max_Depth", "n_estimators","test_AUC"]
x.add_row(["TFIDF", "GBDT", "5", "50","0.719"])
x.add_row(["TFIDF_W2V", "GBDT", "5", "25","0.623"])
print(x)
```

```
+------------+-------+-----------+--------------+----------+
| Vectorizer | Model | Max_Depth | n_estimators | test_AUC |
+------------+-------+-----------+--------------+----------+
|   TFIDF    |  GBDT |     5     |      50      |  0.719   |
| TFIDF_W2V  |  GBDT |     5     |      25      |  0.623   |
+------------+-------+-----------+--------------+----------+
```

- The Accuracy score for TFIDF_W2V might be low as sentiment scores were not used in the model

In [ ]: