# GBDT_Assignment

May 27, 2020

GBDT (xgboost)

## 0.1 Loading Data

[107]:
```
%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

import pickle
from tqdm import tqdm
import os

import chart_studio.plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

[108]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv')
```

[109]:
```
y=data['project_is_approved']
X=data.drop(['project_is_approved'],axis=1)
X.head()
```

[109]:
```
  school_state teacher_prefix project_grade_category  \
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

   teacher_number_of_previously_posted_projects   clean_categories  \
0                                            53       math_science
1                                             4       specialneeds
2                                            10  literacy_language
3                                             2    appliedlearning
4                                             2  literacy_language

                 clean_subcategories  \
0  appliedsciences health_lifescience
1                        specialneeds
2                            literacy
3                    earlydevelopment
4                            literacy

                                               essay   price
0  i fortunate enough use fairy tale stem kits cl…  725.05
1  imagine 8 9 years old you third grade classroo…  213.03
2  having class 24 students comes diverse learner…  329.00
3  i recently read article giving students choice…  481.04
4  my students crave challenge eat obstacles brea…   17.74
```

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

[110]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪stratify=y)

print("shape of train and test after split")
print(X_train.shape,y_train.shape)
print(X_test.shape,y_test.shape)
```

```
shape of train and test after split
(76473, 8) (76473,)
(32775, 8) (32775,)
```

Make Data Model Ready: encoding essay

### 0.2 encoding essay usingTFIDF

[111]:
```
#TFIDF ON Preproccesed Essay

vectorizer_essaytfidf = TfidfVectorizer(min_df=10)
vectorizer_essaytfidf.fit(X_train['essay'].values) # fit has to happen only on␣
↪train data

# we use the fitted TfidfVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer_essaytfidf.transform(X_train['essay'].values)
X_test_essay_tfidf = vectorizer_essaytfidf.transform(X_test['essay'].values)

print("After Tfidf vectorization of Essay")
print(X_train_essay_tfidf.shape, y_train.shape)  #shapes of Train ,Test after␣
↪encoding
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
After Tfidf vectorization of Essay
(76473, 14502) (76473,)
(32775, 14502) (32775,)
```

### 0.3 Using Pretrained Models: TFIDF weighted W2V

[112]:
```
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

[113]:
```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'].values)
# 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())
```

[114]:
```
# average Word2Vec
# compute average word2vec for each review.
def tfidf_weighted_w2v(preprocessed_essay):
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in␣
↪this list
    for sentence in tqdm(preprocessed_essay): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence/
↪review
        for word in sentence.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words):
                vec = model[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and the␣
↪tf value((sentence.count(word)/len(sentence.split())))
```

```
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.
↪split())) # getting the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)
    print(len(tfidf_w2v_vectors))
    print(len(tfidf_w2v_vectors[0]))
    return tfidf_w2v_vectors
```

[115]:
```
X_train_tfidf_weighted_w2v=tfidf_weighted_w2v(X_train['essay'].values)
X_test_tfidf_weighted_w2v=tfidf_weighted_w2v(X_test['essay'].values)
```

```
100%|
| 76473/76473 [02:58<00:00, 429.58it/s]
  1%|
| 428/32775 [00:00<01:07, 482.39it/s]

76473
300

100%|
| 32775/32775 [01:12<00:00, 453.42it/s]

32775
300
```

Make Data Model Ready: encoding numerical, categorical features

### 0.4 Response Coding

[116]:
```
rc_train=pd.DataFrame()
rc_test=pd.DataFrame()
```

[117]:
```
def response_code(data,y_label):
    df=pd.DataFrame({'feature':data.values.tolist(),'label':y_label.values.
↪tolist()})
    cat_values=list(set(df['feature']))
    prob_1={}
    prob_0={}
    for i in cat_values:
        count_1=len(df[(df['feature']==i) & (df['label']==1)])
        count_0=len(df[(df['feature']==i) & (df['label']==0)])
        total=count_1+count_0
        prob_1[i]=count_1/total
        prob_0[i]=count_0/total
```

```
    return prob_0,prob_1
```

Response Coding: School State

[118]:
```
train_state_0,train_state_1=response_code(X_train['school_state'],y_train)
```

[119]:
```
train_state_neg=[]
train_state_pos=[]
for i in X_train['school_state']:
    train_state_neg.append(train_state_0[i])
    train_state_pos.append(train_state_1[i])
rc_train['state_0']=train_state_neg
rc_train['state_1']=train_state_pos

X_train_state0=rc_train['state_0'].values.reshape(-1,1)
X_train_state1=rc_train['state_1'].values.reshape(-1,1)

print("shape after Response Coding")
print(X_train_state0.shape)
print(X_train_state1.shape)
```

```
shape after Response Coding
(76473, 1)
(76473, 1)
```

[120]:
```
state_values=list(set(X_train['school_state']))

test_state_neg=[]
test_state_pos=[]
for i in X_test['school_state']:
    if i in state_values:
        test_state_neg.append(train_state_0[i])
        test_state_pos.append(train_state_1[i])
    else:
        test_state_neg.append(0.5)
        test_state_pos.append(0.5)

rc_test['state_0']=test_state_neg
rc_test['state_1']=test_state_pos

X_test_state0=rc_test['state_0'].values.reshape(-1,1)
X_test_state1=rc_test['state_1'].values.reshape(-1,1)

print("shape after Response Coding")
print(X_test_state0.shape)
print(X_test_state1.shape)
```

```
shape after Response Coding
```

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

Response Coding: Teacher Prefix

[121]:
```
train_prefix_0,train_prefix_1=response_code(X_train['teacher_prefix'],y_train)
```

[122]:
```
train_prefix_neg=[]
train_prefix_pos=[]
for i in X_train['teacher_prefix']:
    train_prefix_neg.append(train_prefix_0[i])
    train_prefix_pos.append(train_prefix_1[i])
rc_train['prefix_0']=train_prefix_neg
rc_train['prefix_1']=train_prefix_pos

X_train_prefix0=rc_train['prefix_0'].values.reshape(-1,1)
X_train_prefix1=rc_train['prefix_1'].values.reshape(-1,1)

print("shape after Response Coding")
print(X_train_prefix0.shape)
print(X_train_prefix1.shape)
```

```
shape after Response Coding
(76473, 1)
(76473, 1)
```

[123]:
```
prefix_values=list(set(X_train['teacher_prefix']))

test_prefix_neg=[]
test_prefix_pos=[]
for i in X_test['teacher_prefix']:
    if i in prefix_values:
        test_prefix_neg.append(train_prefix_0[i])
        test_prefix_pos.append(train_prefix_1[i])
    else:
        test_prefix_neg.append(0.5)
        test_prefix_pos.append(0.5)

rc_test['prefix_0']=test_prefix_neg
rc_test['prefix_1']=test_prefix_pos

X_test_prefix0=rc_test['prefix_0'].values.reshape(-1,1)
X_test_prefix1=rc_test['prefix_1'].values.reshape(-1,1)

print("shape after Response Coding")
print(X_test_prefix0.shape)
print(X_test_prefix1.shape)
```

```
shape after Response Coding
(32775, 1)
(32775, 1)
```

Response Coding: project grade category

[124]:
```
train_grade_0,train_grade_1=response_code(X_train['project_grade_category'],y_train)
```

[125]:
```
train_grade_neg=[]
train_grade_pos=[]
for i in X_train['project_grade_category']:
    train_grade_neg.append(train_grade_0[i])
    train_grade_pos.append(train_grade_1[i])
rc_train['grade_0']=train_grade_neg
rc_train['grade_1']=train_grade_pos

X_train_grade0=rc_train['grade_0'].values.reshape(-1,1)
X_train_grade1=rc_train['grade_1'].values.reshape(-1,1)

print("shape after Response Coding")
print(X_train_grade0.shape)
print(X_train_grade1.shape)
```

```
shape after Response Coding
(76473, 1)
(76473, 1)
```

[126]:
```
grade_values=list(set(X_train['project_grade_category']))

test_grade_neg=[]
test_grade_pos=[]
for i in X_test['project_grade_category']:
    if i in grade_values:
        test_grade_neg.append(train_grade_0[i])
        test_grade_pos.append(train_grade_1[i])
    else:
        test_grade_neg.append(0.5)
        test_grade_pos.append(0.5)

rc_test['grade_0']=test_grade_neg
rc_test['grade_1']=test_grade_pos

X_test_grade0=rc_test['grade_0'].values.reshape(-1,1)
X_test_grade1=rc_test['grade_1'].values.reshape(-1,1)

print("shape after Response Coding")
print(X_test_grade0.shape)
print(X_test_grade1.shape)
```

```
shape after Response Coding
(32775, 1)
(32775, 1)
```

Response Coding: clean categories

[127]:
```
train_categories_0,train_categories_1=response_code(X_train['clean_categories'],y_train)
```

[128]:
```
train_categories_neg=[]
train_categories_pos=[]
for i in X_train['clean_categories']:
    train_categories_neg.append(train_categories_0[i])
    train_categories_pos.append(train_categories_1[i])
rc_train['categories_0']=train_categories_neg
rc_train['categories_1']=train_categories_pos

X_train_categories0=rc_train['categories_0'].values.reshape(-1,1)
X_train_categories1=rc_train['categories_1'].values.reshape(-1,1)

print("shape after Response Coding")
print(X_train_categories0.shape)
print(X_train_categories1.shape)
```

```
shape after Response Coding
(76473, 1)
(76473, 1)
```

[129]:
```
categories_values=list(set(X_train['clean_categories']))

test_categories_neg=[]
test_categories_pos=[]
for i in X_test['clean_categories']:
    if i in categories_values:
        test_categories_neg.append(train_categories_0[i])
        test_categories_pos.append(train_categories_1[i])
    else:
        test_categories_neg.append(0.5)
        test_categories_pos.append(0.5)

rc_test['categories_0']=test_categories_neg
rc_test['categories_1']=test_categories_pos

X_test_categories0=rc_test['categories_0'].values.reshape(-1,1)
X_test_categories1=rc_test['categories_1'].values.reshape(-1,1)

print("shape after Response Coding")
print(X_test_categories0.shape)
print(X_test_categories1.shape)
```

```
shape after Response Coding
(32775, 1)
(32775, 1)
```

Response Coding: clean subcategories

[130]:
```
train_subcategories_0,train_subcategories_1=response_code(X_train['clean_subcategories'],y_tra
```

[131]:
```
train_subcategories_neg=[]
train_subcategories_pos=[]
for i in X_train['clean_subcategories']:
    train_subcategories_neg.append(train_subcategories_0[i])
    train_subcategories_pos.append(train_subcategories_1[i])
rc_train['subcategories_0']=train_subcategories_neg
rc_train['subcategories_1']=train_subcategories_pos

X_train_subcategories0=rc_train['subcategories_0'].values.reshape(-1,1)
X_train_subcategories1=rc_train['subcategories_1'].values.reshape(-1,1)

print("shape after Response Coding")
print(X_train_subcategories0.shape)
print(X_train_subcategories1.shape)
```

```
shape after Response Coding
(76473, 1)
(76473, 1)
```

[132]:
```
subcategories_values=list(set(X_train['clean_subcategories']))

test_subcategories_neg=[]
test_subcategories_pos=[]
for i in X_test['clean_subcategories']:
    if i in subcategories_values:
        test_subcategories_neg.append(train_subcategories_0[i])
        test_subcategories_pos.append(train_subcategories_1[i])
    else:
        test_subcategories_neg.append(0.5)
        test_subcategories_pos.append(0.5)

rc_test['subcategories_0']=test_subcategories_neg
rc_test['subcategories_1']=test_subcategories_pos

X_test_subcategories0=rc_test['subcategories_0'].values.reshape(-1,1)
X_test_subcategories1=rc_test['subcategories_1'].values.reshape(-1,1)

print("shape after Response Coding")
print(X_test_subcategories0.shape)
print(X_test_subcategories1.shape)
```

```
shape after Response Coding
(32775, 1)
(32775, 1)
```

[133]:
```
rc_train.head()
```

[133]:
```
    state_0   state_1  prefix_0  prefix_1   grade_0   grade_1  categories_0  \
0  0.144134  0.855866  0.143451  0.856549  0.151784  0.848216      0.146133
1  0.164474  0.835526  0.143451  0.856549  0.144270  0.855730      0.176643
2  0.143595  0.856405  0.156878  0.843122  0.144270  0.855730      0.150436
3  0.170488  0.829512  0.156878  0.843122  0.158535  0.841465      0.134300
4  0.170488  0.829512  0.162017  0.837983  0.163279  0.836721      0.150436

   categories_1  subcategories_0  subcategories_1
0      0.853867         0.177255         0.822745
1      0.823357         0.176471         0.823529
2      0.849564         0.154589         0.845411
3      0.865700         0.144049         0.855951
4      0.849564         0.189295         0.810705
```

[134]:
```
rc_test.head()
```

[134]:
```
    state_0   state_1  prefix_0  prefix_1   grade_0   grade_1  categories_0  \
0  0.143538  0.856462  0.143451  0.856549  0.151784  0.848216      0.141732
1  0.143538  0.856462  0.156878  0.843122  0.144270  0.855730      0.132333
2  0.143373  0.856627  0.156878  0.843122  0.158535  0.841465      0.118421
3  0.143538  0.856462  0.156878  0.843122  0.151784  0.848216      0.132333
4  0.123576  0.876424  0.156878  0.843122  0.163279  0.836721      0.185542

   categories_1  subcategories_0  subcategories_1
0      0.858268         0.142857         0.857143
1      0.867667         0.132807         0.867193
2      0.881579         0.114014         0.885986
3      0.867667         0.130531         0.869469
4      0.814458         0.117647         0.882353
```

encoding Numerical features: teacher_number_of_previously_posted_projects

[135]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(1,-1))

X_train_prev_proj=(normalizer.
 ↪transform(X_train['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(1,-1))).reshape(-1,1)
```

```
X_test_prev_proj=(normalizer.
 ↪transform(X_test['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(1,-1))).reshape(-1,1)

print("shape of matrix")
print(X_train_prev_proj.shape, y_train.shape)
print(X_test_prev_proj.shape, y_test.shape)
```

```
shape of matrix
(76473, 1) (76473,)
(32775, 1) (32775,)
```

encoding Numerical features: Price

[136]:
```
normalizer = Normalizer()
normalizer.fit(X_train['price'].values.reshape(1,-1))

X_train_price= (normalizer.transform(X_train['price'].values.reshape(1,-1))).
 ↪reshape(-1,1)
X_test_price=(normalizer.transform(X_test['price'].values.reshape(1,-1))).
 ↪reshape(-1,1)

print("Shape of price matrix")
print(X_train_price.shape, y_train.shape)
print(X_test_price.shape, y_test.shape)
```

```
Shape of price matrix
(76473, 1) (76473,)
(32775, 1) (32775,)
```

### 0.5 Sentiment Score of essay

[137]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

nltk.download('punkt')
nltk.download('vader_lexicon')
```

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

[137]: True

[138]:
```
sid = SentimentIntensityAnalyzer()

neg=[]
neu=[]
pos=[]
compound=[]
for for_sentiment in X_train['essay']:
    ss = sid.polarity_scores(for_sentiment)

    neg.append(ss['neg'])
    neu.append(ss['neu'])
    pos.append(ss['pos'])
    compound.append(ss['compound'])
```

[139]:
```
X_train['neg']=neg
X_train['neu']=neu
X_train['pos']=pos
X_train['compound']=compound

X_train_neg=X_train['neg'].values.reshape(-1,1)
X_train_neu=X_train['neu'].values.reshape(-1,1)
X_train_pos=X_train['pos'].values.reshape(-1,1)
X_train_compound=X_train['compound'].values.reshape(-1,1)

print("shape after Sentiment Analysis of essay")
print(X_train_neg.shape)
print(X_train_neu.shape)
print(X_train_pos.shape)
print(X_train_compound.shape)
```

```
shape after Sentiment Analysis of essay
(76473, 1)
(76473, 1)
(76473, 1)
(76473, 1)
```

[140]:
```
X_train.head()
```

[140]:

| | school_state | teacher_prefix | project_grade_category | \ |
|---|---|---|---|---|
| 97905 | mo | mrs | grades_prek_2 | |
| 45564 | ut | mrs | grades_3_5 | |
| 28042 | ca | ms | grades_3_5 | |
| 41186 | la | ms | grades_6_8 | |
| 104929 | la | mr | grades_9_12 | |

| | teacher_number_of_previously_posted_projects | clean_categories | \ |
|---|---|---|---|
| 97905 | 1 | music_arts | |

```
45564                                                 0       math_science
28042                                                15      health_sports
41186                                                 1  literacy_language
104929                                                0      health_sports

                    clean_subcategories  \
97905                         visualarts
45564   environmentalscience mathematics
28042                        gym_fitness
41186                literature_writing
104929                        teamsports

                                                    essay   price    neg  \
97905   i privilege teaching art small school rural co…   60.24  0.037
45564   my students come crowded homes utah families n…  197.11  0.000
28042   imagine class filled energetic high spirited a…  213.66  0.000
41186   my students come work hard every day improving…   11.69  0.025
104929  our high school located rural part parish many…  349.98  0.023

          neu    pos  compound
97905   0.618  0.344    0.9979
45564   0.676  0.324    0.9933
28042   0.533  0.467    0.9970
41186   0.768  0.208    0.9750
104929  0.659  0.318    0.9913
```

[141]:
```
sid = SentimentIntensityAnalyzer()

neg=[]
neu=[]
pos=[]
compound=[]
for for_sentiment in X_test['essay']:
    ss = sid.polarity_scores(for_sentiment)

    neg.append(ss['neg'])
    neu.append(ss['neu'])
    pos.append(ss['pos'])
    compound.append(ss['compound'])
```

[142]:
```
X_test['neg']=neg
X_test['neu']=neu
X_test['pos']=pos
X_test['compound']=compound

X_test_neg=X_test['neg'].values.reshape(-1,1)
X_test_neu=X_test['neu'].values.reshape(-1,1)
```

```
X_test_pos=X_test['pos'].values.reshape(-1,1)
X_test_compound=X_test['compound'].values.reshape(-1,1)

print("shape after Sentiment Analysis of essay")
print(X_test_neg.shape)
print(X_test_neu.shape)
print(X_test_pos.shape)
print(X_test_compound.shape)
```

```
shape after Sentiment Analysis of essay
(32775, 1)
(32775, 1)
(32775, 1)
(32775, 1)
```

[143]:
```
X_test.head()
```

[143]:

| | school_state | teacher_prefix | project_grade_category | \ |
|---|---|---|---|---|
| 105816 | ny | mrs | grades_prek_2 | |
| 7593 | ny | ms | grades_3_5 | |
| 23684 | il | ms | grades_6_8 | |
| 68040 | ny | ms | grades_prek_2 | |
| 1941 | oh | ms | grades_9_12 | |

| | teacher_number_of_previously_posted_projects | \ |
|---|---|---|
| 105816 | 1 | |
| 7593 | 6 | |
| 23684 | 1 | |
| 68040 | 44 | |
| 1941 | 1 | |

| | clean_categories | clean_subcategories | \ |
|---|---|---|---|
| 105816 | appliedlearning literacy_language | earlydevelopment literacy | |
| 7593 | literacy_language math_science | literature_writing mathematics | |
| 23684 | health_sports specialneeds | health_wellness specialneeds | |
| 68040 | literacy_language math_science | literacy mathematics | |
| 1941 | appliedlearning health_sports | extracurricular teamsports | |

| | essay | price | neg | \ |
|---|---|---|---|---|
| 105816 | this first year teaching kindergarten students… | 101.98 | 0.000 | |
| 7593 | i twenty six different wonderful students wide… | 379.95 | 0.012 | |
| 23684 | this first year teaching 6th grade transitioni… | 184.27 | 0.038 | |
| 68040 | i teach 12 sometimes 13 special needs students… | 86.38 | 0.026 | |
| 1941 | my students need constructive outlet participa… | 337.62 | 0.037 | |

| | neu | pos | compound |
|---|---|---|---|
| 105816 | 0.716 | 0.284 | 0.9892 |

| | | | |
|---|---|---|---|
| 7593 | 0.488 | 0.500 | 0.9969 |
| 23684 | 0.689 | 0.273 | 0.9971 |
| 68040 | 0.717 | 0.256 | 0.9825 |
| 1941 | 0.792 | 0.170 | 0.9811 |

### 0.6 Concatinating all the features (essay-TFIDF)

[144]:
```
#concatinating all features- Categorical(response code)+ Numerical Features+␣
 ↪essay(tfidf)+ Sentiment Scores of essay

from scipy.sparse import hstack

X_train_tfidf=hstack((X_train_state0,X_train_state1,X_train_prefix0,X_train_prefix1,X_train_gr
 ↪tocsr()
X_test_tfidf=hstack((X_test_state0,X_test_state1,X_test_prefix0,X_test_prefix1,X_test_grade0,X
 ↪tocsr()

print("Final Data matrix")
print(X_train_tfidf.shape, y_train.shape)
print(X_test_tfidf.shape, y_test.shape)
```

```
Final Data matrix
(76473, 14518) (76473,)
(32775, 14518) (32775,)
```

### 0.7 Concatinating all the features (essay-TFIDF_Weighted_W2V)

[145]:
```
#concatinating all features-Categorical(response code)+ Numerical Features +␣
 ↪essay(tfidf w2v)using hstack

X_train_tfidfw2v=np.
 ↪hstack((X_train_state0,X_train_state1,X_train_prefix0,X_train_prefix1,X_train_grade0,X_trai
X_test_tfidfw2v=np.
 ↪hstack((X_test_state0,X_test_state1,X_test_prefix0,X_test_prefix1,X_test_grade0,X_test_grad

print("Final Data matrix")
print(X_train_tfidfw2v.shape, y_train.shape)
print(X_test_tfidfw2v.shape, y_test.shape)
```

```
Final Data matrix
(76473, 312) (76473,)
(32775, 312) (32775,)
```

# 1 Applying GBDT on Set 1 Essay-TFIDF

[146]:
```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score

xgb_model=XGBClassifier()

parameters={'max_depth':[1, 5, 10, 50],'n_estimators': [10,50,100,200]}
clf=GridSearchCV(xgb_model,parameters,cv=3,scoring='roc_auc',return_train_score='True')
clf.fit(X_train_tfidf,y_train)
```

[146]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=None, max_delta_step=None,
                                     max_depth=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     n_estim…
                                     objective='binary:logistic',
                                     random_state=None, reg_alpha=None,
                                     reg_lambda=None, scale_pos_weight=None,
                                     subsample=None, tree_method=None,
                                     validate_parameters=False,
                                     verbosity=None),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'n_estimators': [10, 50, 100, 200]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score='True',
             scoring='roc_auc', verbose=0)
```

[147]:
```
clf.best_params_
```

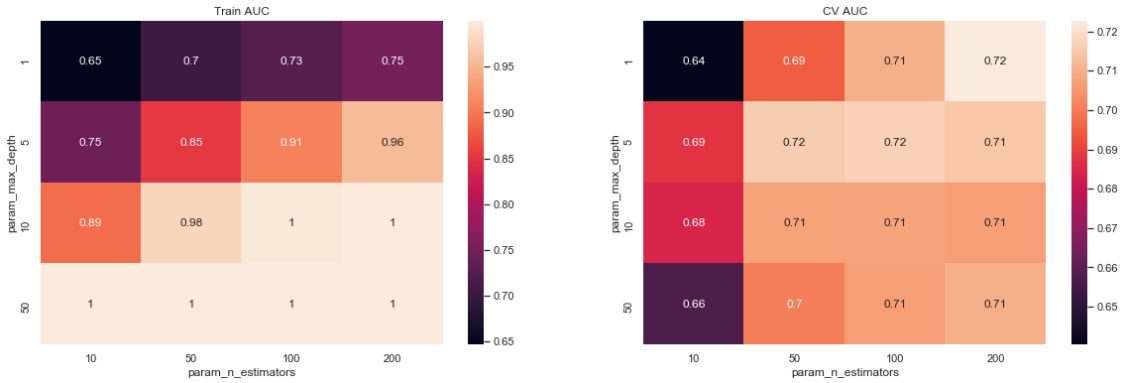
[147]:
```
{'max_depth': 1, 'n_estimators': 200}
```

[148]:
```
best_maxdepth=clf.best_params_['max_depth']
best_n_estimators=clf.best_params_['n_estimators']
print(best_maxdepth,best_n_estimators)
```

```
1 200
```

## 1.1 Representation of Results

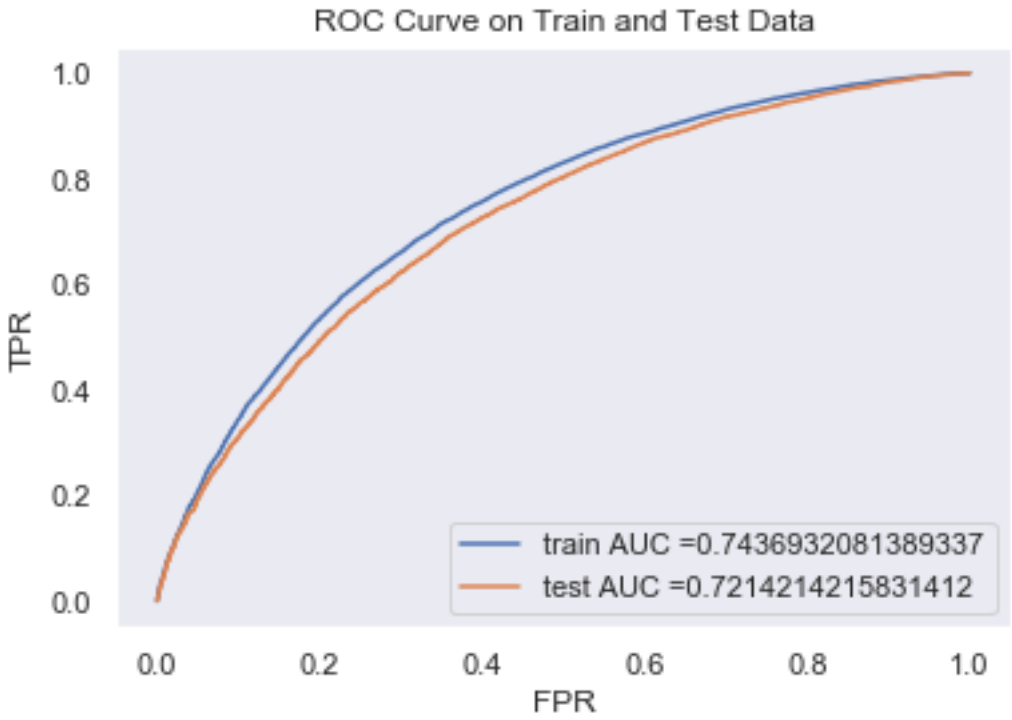
[149]:
```
import seaborn as sns
sns.set()

results=pd.DataFrame(clf.cv_results_).
 ↪groupby(['param_max_depth','param_n_estimators']).max()\
.unstack()[['mean_test_score', 'mean_train_score']]

fig,ax=plt.subplots(1,2,figsize=(20,6))
sns.heatmap(results.mean_train_score,annot = True,ax=ax[0])
sns.heatmap(results.mean_test_score,annot = True,ax=ax[1])
ax[0].set_title('Train AUC')
ax[1].set_title('CV AUC')
plt.show()
```




## 1.2 Testing Performance of the Model with Best Hyper Parameters

[150]:
```
from sklearn.metrics import roc_curve, auc

classfier=XGBClassifier(max_depth=best_maxdepth,n_estimators=best_n_estimators)
classfier.fit(X_train_tfidf,y_train)

y_train_pred=classfier.predict_proba(X_train_tfidf)[:,1]
y_test_pred=classfier.predict_proba(X_test_tfidf)[:,1]

train_fpr, train_tpr, train_thresolds = roc_curve(y_train,y_train_pred)
test_fpr, test_tpr, test_thresolds = roc_curve(y_test,y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr,␣
 ↪train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))␣
 ↪#plotting ROC Curves for train and test datasets
```

```
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve on Train and Test Data")
plt.grid()
plt.show()
```




[151]:
```
print("AUC score on Train data is ")
print(auc(train_fpr, train_tpr))
print("AUC score on Test data is ")
print(auc(test_fpr, test_tpr))
```

```
AUC score on Train data is
0.7436932081389337
AUC score on Test data is
0.7214214215831412
```

[152]:
```
# we will pick a threshold that will give the least fpr
def find_best_thresold(thresold, fpr, tpr):
    t = thresold[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) is ", max(tpr*(1-fpr)), "for␣
 ↪thresold", np.round(t,3))
```

```
    return t

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

[153]:
```
from sklearn.metrics import confusion_matrix
best_t = find_best_thresold(train_thresolds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) is  0.46536958622775537 for thresold 0.838
Train confusion matrix
[[ 7896  3683]
 [20608 44286]]
Test confusion matrix
[[ 3423  1540]
 [10166 17646]]
```

## 2 Applying GBDT on Set 2 Essay-TFIDF W2V

[154]:
```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score

xgb_model=XGBClassifier()

parameters={'max_depth':[1, 5, 10, 50],'n_estimators': [10,50,100,200]}
clf=GridSearchCV(xgb_model,parameters,cv=3,scoring='roc_auc',return_train_score='True')
clf.fit(X_train_tfidfw2v,y_train)
```

[154]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=None,
                                     gpu_id=None, importance_type='gain',
                                     interaction_constraints=None,
```

```
                                   learning_rate=None, max_delta_step=None,
                                   max_depth=None, min_child_weight=None,
                                   missing=nan, monotone_constraints=None,
                                   n_estim…
                                   objective='binary:logistic',
                                   random_state=None, reg_alpha=None,
                                   reg_lambda=None, scale_pos_weight=None,
                                   subsample=None, tree_method=None,
                                   validate_parameters=False,
                                   verbosity=None),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'n_estimators': [10, 50, 100, 200]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score='True',
             scoring='roc_auc', verbose=0)
```

[155]:
```
clf.best_params_
```

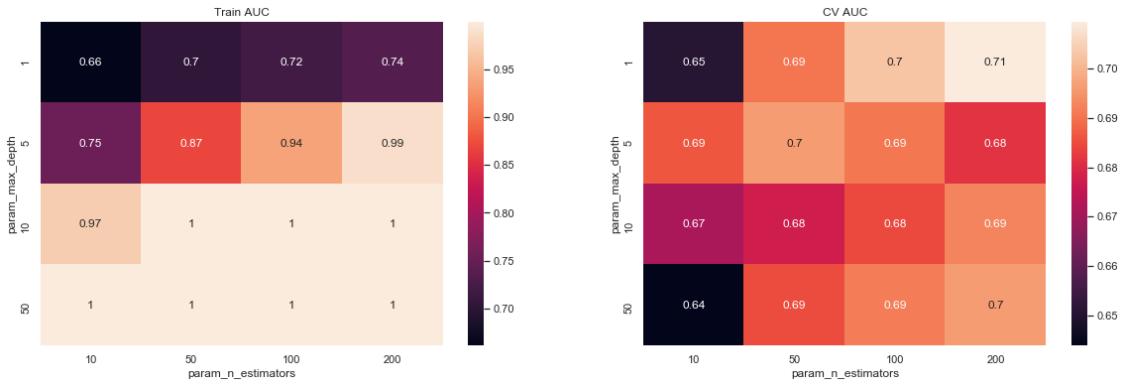
[155]:
```
{'max_depth': 1, 'n_estimators': 200}
```

[156]:
```
best_maxdepth=clf.best_params_['max_depth']
best_n_estimators=clf.best_params_['n_estimators']
print(best_maxdepth,best_n_estimators)
```

```
1 200
```

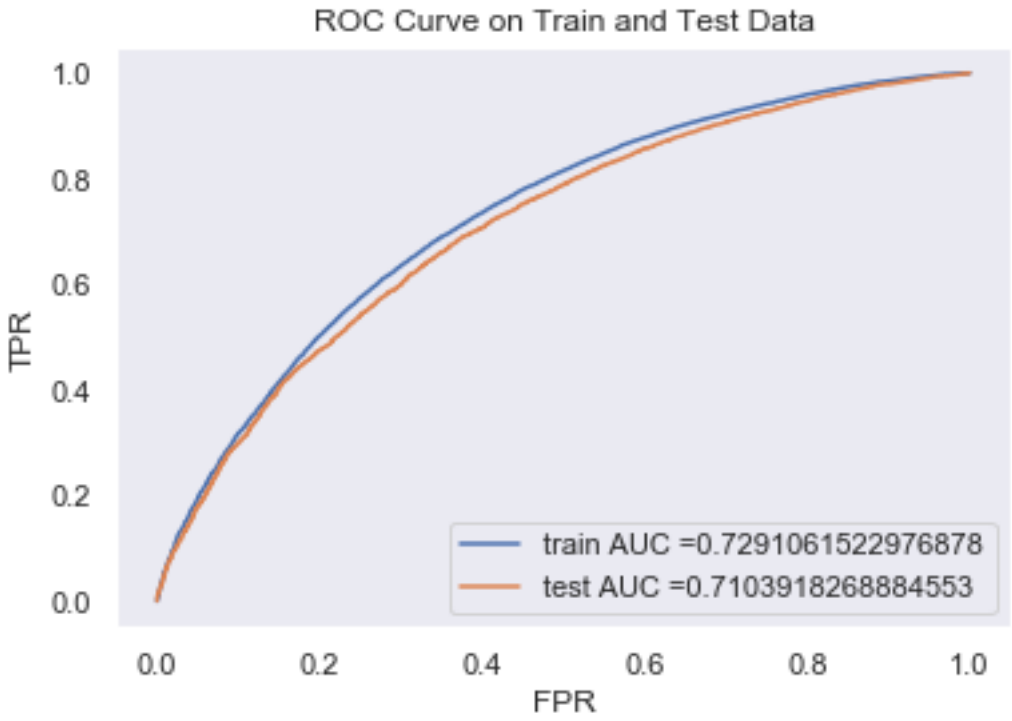
### 2.1 Representation of Results

[157]:
```
import seaborn as sns
sns.set()

results=pd.DataFrame(clf.cv_results_).
 ↪groupby(['param_max_depth','param_n_estimators']).max()\
.unstack()[['mean_test_score', 'mean_train_score']]

fig,ax=plt.subplots(1,2,figsize=(20,6))
sns.heatmap(results.mean_train_score,annot = True,ax=ax[0])
sns.heatmap(results.mean_test_score,annot = True,ax=ax[1])
ax[0].set_title('Train AUC')
ax[1].set_title('CV AUC')
plt.show()
```




## 2.2 Testing Performance of the Model with Best Hyper Parameters

[158]:
```
from sklearn.metrics import roc_curve, auc

classfier=XGBClassifier(max_depth=best_maxdepth,n_estimators=best_n_estimators)
classfier.fit(X_train_tfidfw2v,y_train)

y_train_pred=classfier.predict_proba(X_train_tfidfw2v)[:,1]
y_test_pred=classfier.predict_proba(X_test_tfidfw2v)[:,1]

train_fpr, train_tpr, train_thresolds = roc_curve(y_train,y_train_pred)
test_fpr, test_tpr, test_thresolds = roc_curve(y_test,y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr,␣
 ↪train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))␣
 ↪#plotting ROC Curves for train and test datasets
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve on Train and Test Data")
plt.grid()
plt.show()
```




[159]:
```
print("AUC score on Train data is ")
print(auc(train_fpr, train_tpr))
print("AUC score on Test data is ")
print(auc(test_fpr, test_tpr))
```

```
AUC score on Train data is
0.7291061522976878
AUC score on Test data is
0.7103918268884553
```

[160]:
```
# we will pick a threshold that will give the least fpr
def find_best_thresold(thresold, fpr, tpr):
    t = thresold[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) is ", max(tpr*(1-fpr)), "for␣
 ↪thresold", np.round(t,3))
    return t

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

```
        else:
            predictions.append(0)
    return predictions
```

[161]:
```
from sklearn.metrics import confusion_matrix
best_t = find_best_thresold(train_thresolds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) is  0.4489244563509394 for thresold 0.839
Train confusion matrix
[[ 7663  3916]
 [20874 44020]]
Test confusion matrix
[[ 3316  1647]
 [ 9975 17837]]
```

Summary

[162]:
```
from prettytable import PrettyTable
t = PrettyTable(['Vectorizer', 'Model','Hyper Parameter','AUC'])
t.add_row(['TFIDF', 'GBT','max_depth:1,n_estimators:200',0.7214])
t.add_row(['TFIDF W2V','GBT','max_depth:1,n_estimators:200',0.7103])
print(t)
```

```
+------------+-------+-------------------------------+--------+
| Vectorizer | Model |       Hyper Parameter         |  AUC   |
+------------+-------+-------------------------------+--------+
|   TFIDF    |  GBT  | max_depth:1,n_estimators:200 | 0.7214 |
| TFIDF W2V  |  GBT  | max_depth:1,n_estimators:200 | 0.7103 |
+------------+-------+-------------------------------+--------+
```

[ ]: