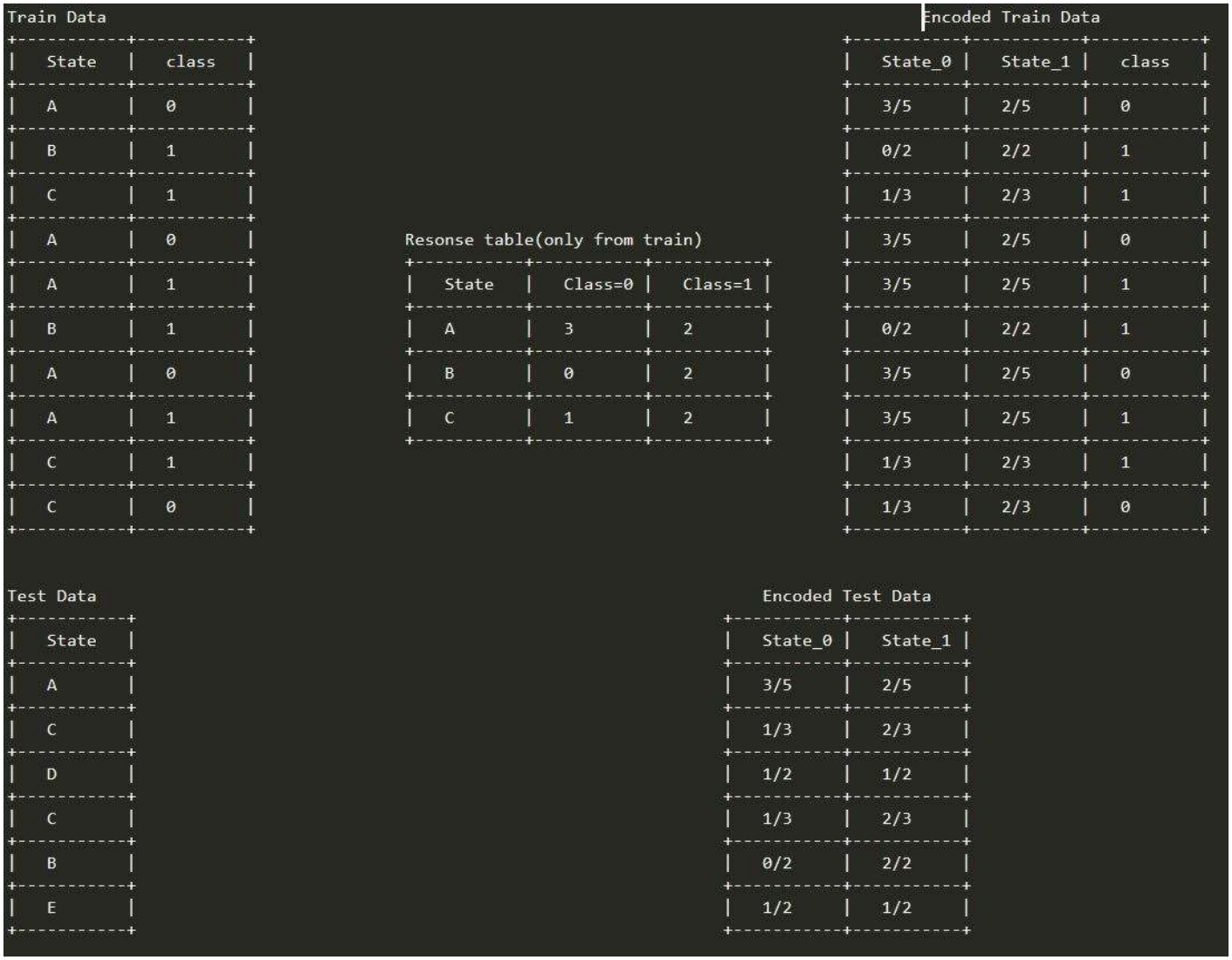
Assignment 9: GBDT

Response Coding: Example

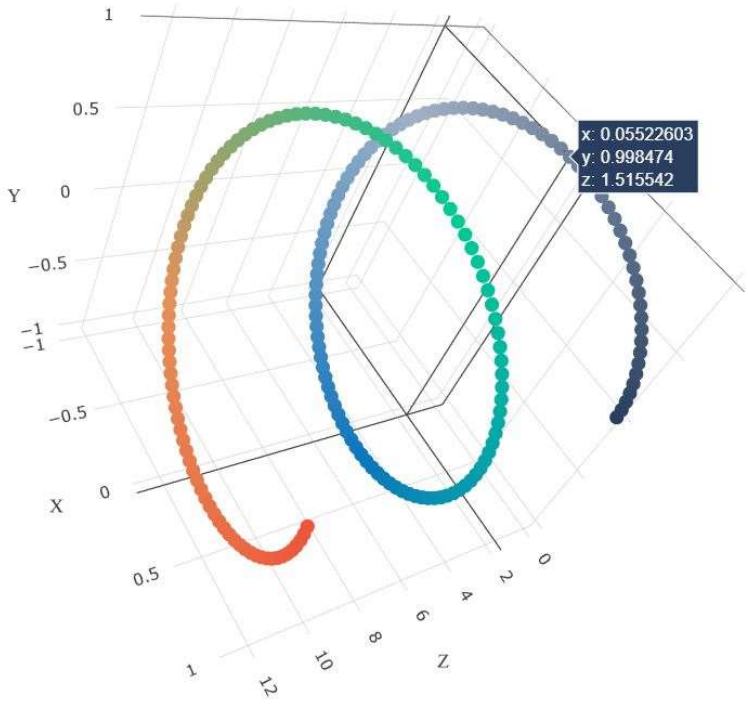


> The response tabel is built only on train dataset. For a category which is not there in train data and present in test data, we will encode them with default values Ex: in our test data if have State: D then we encode it as [0.5, 0.05]

1. **Apply GBDT on these feature sets**
   - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check the bellow example, include all 4 values as 4 features)

- Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

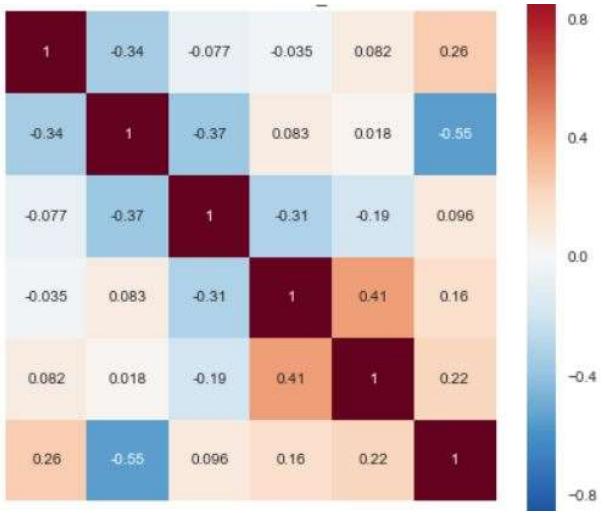
2. **The hyper paramter tuning (Consider any two hyper parameters)**
   - Find the best hyper parameter which will give the maximum AUC value
   - find the best hyper paramter using k-fold cross validation/simple cross validation data
   - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task
3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



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

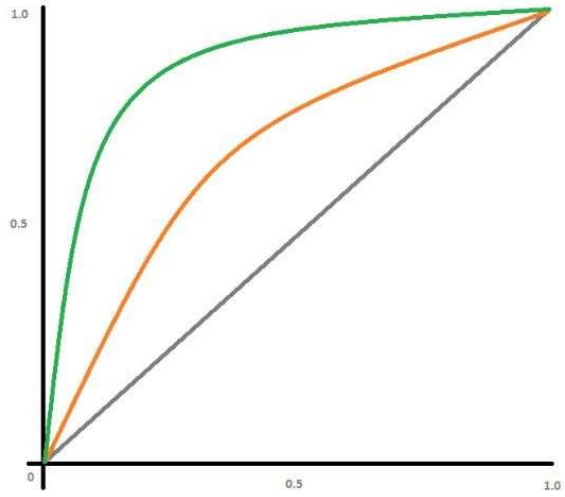
or

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



[seaborn heat maps](#) with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**

- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.



- Along with plotting ROC curve, you need to print the [confusion matrix](#) with predicted and original labels of test data points

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

4. You need to summarize the results at the end of the notebook, summarize it in the table format

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

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

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
True
```

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

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest stude
for learning my students learn in many different ways using all of our senses and multiple in
of techniques to help all my students succeed students in my class come from a variety of dif
for wonderful sharing of experiences and cultures including native americans our school is a
learners which can be seen through collaborative student project based learning in and out of
in my class love to work with hands on materials and have many different opportunities to pra
mastered having the social skills to work cooperatively with friends is a crucial aspect of t
montana is the perfect place to learn about agriculture and nutrition my students love to rol
in the early childhood classroom i have had several kids ask me can we try cooking with real
and create common core cooking lessons where we learn important math and writing concepts whi
food for snack time my students will have a grounded appreciation for the work that went into
of where the ingredients came from as well as how it is healthy for their bodies this project
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade
and mix up healthy plants from our classroom garden in the spring we will also create our own
shared with families students will gain math and literature skills as well as a life long enj
nannan'
ss = sid.polarity_scores(for_sentiment)

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

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975, /usr/local/lib/python3.7/dist-packa
  warnings.warn("The twython library has not been installed. "
```

```
pip install chart_studio
```

```
Collecting chart_studio
  Downloading https://files.pythonhosted.org/packages/ca/ce/330794a6b6ca4b9182c38fc69dd2
     |████████████████████████████████| 71kB 3.2MB/s
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from chart
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (from ch
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (f
Installing collected packages: chart-studio
Successfully installed chart-studio-1.1.0
```

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

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

1. GBDT (xgboost/lightgbm)

1.1 Loading Data

```
import pandas
data = pandas.read_csv('preprocessed_data.csv',nrows = 50000)
```

```
SID = SentimentIntensityAnalyzer()
```

```
negitive = []
positive = []
neutral = []
compound = []
for i in tqdm(data['essay']):
    j = SID.polarity_scores(i)['neg']
    k = SID.polarity_scores(i)['neu']
    l = SID.polarity_scores(i)['pos']

    negitive.append(j)
    positive.append(k)
    neutral.append(l)
```

```
 100%|██████████| 50000/50000 [05:19<00:00, 156.51it/s]
```

```
data['negitive'] = negitive
data['positive'] = positive
data['neutral'] = neutral
data.head(2)
```

```
school_state  teacher_prefix  project_grade_category  teacher_number_of_previously_p
```

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

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, data["project_is_approved"],
                                                    test_size = 0.33, stratify = data["projec
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_
```

1.3 Make Data Model Ready: encoding eassay, and project_title

```
preprocessed_essays = data['essay'].values
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
text_tfidf = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (50000, 12122)
```

```
tfid_vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
tfid_vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfid = tfid_vectorizer.transform(X_train['essay'].values)
X_cv_essay_tfid = tfid_vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfid = tfid_vectorizer.transform(X_test['essay'].values)
```

```
print(X_train_essay_tfid.shape, y_train.shape)
print(X_cv_essay_tfid.shape, y_cv.shape)
print(X_test_essay_tfid.shape, y_test.shape)
```

```
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
```

TFIDF W2V

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays)
# 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())


# average Word2Vec
# compute average word2vec for each review.
def w2v(preprocessed_essays):
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(preprocessed_essays): # 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((sente
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gett
                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)
    return tfidf_w2v_vectors


X_train_essay_w2v =  w2v(X_train['essay'].values)
X_test_essay_w2v =  w2v(X_test['essay'].values)
X_cv_essay_w2v =  w2v(X_cv['essay'].values)
```

```
100%|██████████| 22445/22445 [00:49<00:00, 457.61it/s]
100%|██████████| 16500/16500 [00:35<00:00, 458.90it/s]
100%|██████████| 11055/11055 [00:24<00:00, 454.31it/s]
```

1.4 Make Data Model Ready: encoding numerical, categorical features

Vectorizing Categorical Data using Respond Coding

```
def trainResponseEncoding(working_data, cat_type):


    working_data.loc[working_data[cat_type].isnull(), cat_type] = 'nan' #Imputation

    data_0 = working_data[working_data['project_is_approved']==0].groupby(cat_type).size()
    data_1 = working_data[working_data['project_is_approved']==1].groupby(cat_type).size()
```

```
    return data_0, data_1

def getResponseEnconding(data_0, data_1, working_data, cat_type):
    working_data.loc[working_data[cat_type].isnull(), cat_type] = 'nan'

    colwise_dict = {'class=0':[], 'class=1': []}
    pos = []
    neg = []
    for row in working_data[cat_type]:
        colwise_dict['class=0'].append(data_0.get(row, 0.5))
        colwise_dict['class=1'].append(data_1.get(row, 0.5))
        i = data_0.get(row, 0.5)
        j = data_1.get(row, 0.5)
        working_data[cat_type+"0"] = i / (i + j)
        working_data[cat_type+"1"] = j / (i + j)


    response_enc = pd.DataFrame(colwise_dict)
    print("Shape response encoding ",response_enc.shape)
    #print(response_enc)   # for response table


cols_dict = {'cat_cols': ['school_state','clean_categories', 'clean_subcategories', 'project_

for col_type, cols_name in cols_dict.items():
    if col_type == 'cat_cols':
        for cat_type in cols_name:
            print(cat_type)
            data_0, data_1 = trainResponseEncoding(X_train, cat_type)


            for data_type, data_part in [('X_train', X_train), ('X_cv',X_cv), ('X_test',X_tes
                getResponseEnconding(data_0, data_1, data_part, cat_type)
```

```
school_state
Shape response encoding  (22445, 2)
Shape response encoding  (11055, 2)
Shape response encoding  (16500, 2)
clean_categories
Shape response encoding  (22445, 2)
Shape response encoding  (11055, 2)
Shape response encoding  (16500, 2)
clean_subcategories
Shape response encoding  (22445, 2)
Shape response encoding  (11055, 2)
Shape response encoding  (16500, 2)
project_grade_category
Shape response encoding  (22445, 2)
Shape response encoding  (11055, 2)
Shape response encoding  (16500, 2)
teacher_prefix
Shape response encoding  (22445, 2)
```

```
Shape response encoding  (11055, 2)
```

```
print(X_train.columns)
```

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price', 'negitive',
       'positive', 'neutral', 'school_state0', 'school_state1',
       'clean_categories0', 'clean_categories1', 'clean_subcategories0',
       'clean_subcategories1', 'project_grade_category0',
       'project_grade_category1', 'teacher_prefix0', 'teacher_prefix1'],
      dtype='object')
```

```
X_train.head(2)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previous
11443	tx	ms	grades_3_5	
40185	ga	ms	grades_3_5	

School State [0]

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['school_state0'].values.reshape(1,-1))

X_train_state_ohe = normalizer.transform(X_train['school_state0'].values.reshape(-1,1))
X_cv_state_ohe = normalizer.transform(X_cv['school_state0'].values.reshape(-1,1))
X_test_state_ohe = normalizer.transform(X_test['school_state0'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

School State [1]

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['school_state1'].values.reshape(1,-1))

X_train_state1_ohe = normalizer.transform(X_train['school_state1'].values.reshape(-1,1))
X_cv_state1_ohe = normalizer.transform(X_cv['school_state1'].values.reshape(-1,1))
X_test_state1_ohe = normalizer.transform(X_test['school_state1'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_state1_ohe.shape, y_train.shape)
print(X_cv_state1_ohe.shape, y_cv.shape)
print(X_test_state1_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Clean Categories [0]

```
from sklearn.preprocessing import Normalizer
```

```
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['clean_categories0'].values.reshape(1,-1))

X_train_clean_cat_ohe = normalizer.transform(X_train['clean_categories0'].values.reshape(-1,1
X_cv_clean_cat_ohe = normalizer.transform(X_cv['clean_categories0'].values.reshape(-1,1))
X_test_clean_cat_ohe = normalizer.transform(X_test['clean_categories0'].values.reshape(-1,1))



print("After vectorizations")
print(X_train_clean_cat_ohe.shape, y_train.shape)
print(X_cv_clean_cat_ohe.shape, y_cv.shape)
print(X_test_clean_cat_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Clean Categories [1]

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['clean_categories1'].values.reshape(1,-1))

X_train_clean_cat1_ohe = normalizer.transform(X_train['clean_categories1'].values.reshape(-1,
X_cv_clean_cat1_ohe = normalizer.transform(X_cv['clean_categories1'].values.reshape(-1,1))
X_test_clean_cat1_ohe = normalizer.transform(X_test['clean_categories1'].values.reshape(-1,1)



print("After vectorizations")
print(X_train_clean_cat1_ohe.shape, y_train.shape)
print(X_cv_clean_cat1_ohe.shape, y_cv.shape)
print(X_test_clean_cat1_ohe.shape, y_test.shape)
```

```
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Clean Sub_Categories [0]

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['clean_subcategories0'].values.reshape(1,-1))

X_train_clean_subcat_ohe = normalizer.transform(X_train['clean_subcategories0'].values.reshap
X_cv_clean_subcat_ohe = normalizer.transform(X_cv['clean_subcategories0'].values.reshape(-1,1
X_test_clean_subcat_ohe = normalizer.transform(X_test['clean_subcategories0'].values.reshape(
```

```
print("After vectorizations")
print(X_train_clean_subcat_ohe.shape, y_train.shape)
print(X_cv_clean_subcat_ohe.shape, y_cv.shape)
print(X_test_clean_subcat_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Clean Sub_Categories [1]

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

```
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['clean_subcategories1'].values.reshape(1,-1))

X_train_clean_subcat1_ohe = normalizer.transform(X_train['clean_subcategories1'].values.resha
X_cv_clean_subcat1_ohe = normalizer.transform(X_cv['clean_subcategories1'].values.reshape(-1,
X_test_clean_subcat1_ohe = normalizer.transform(X_test['clean_subcategories1'].values.reshape



print("After vectorizations")
print(X_train_clean_subcat1_ohe.shape, y_train.shape)
print(X_cv_clean_subcat1_ohe.shape, y_cv.shape)
print(X_test_clean_subcat1_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Project Grade Category [0]

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['project_grade_category0'].values.reshape(1,-1))

X_train_grade_ohe = normalizer.transform(X_train['project_grade_category0'].values.reshape(-1
X_cv_grade_ohe = normalizer.transform(X_cv['project_grade_category0'].values.reshape(-1,1))
X_test_grade_ohe = normalizer.transform(X_test['project_grade_category0'].values.reshape(-1,1



print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
```

```
(16500, 1) (16500,)
==========================================================================================
```

Project Grade Category [1]

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['project_grade_category1'].values.reshape(1,-1))

X_train_grade1_ohe = normalizer.transform(X_train['project_grade_category1'].values.reshape(-
X_cv_grade1_ohe = normalizer.transform(X_cv['project_grade_category1'].values.reshape(-1,1))
X_test_grade1_ohe = normalizer.transform(X_test['project_grade_category1'].values.reshape(-1,



print("After vectorizations")
print(X_train_grade1_ohe.shape, y_train.shape)
print(X_cv_grade1_ohe.shape, y_cv.shape)
print(X_test_grade1_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
==========================================================================================
```

Teacher Prefix [0]

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_prefix0'].values.reshape(1,-1))

X_train_teacher_ohe = normalizer.transform(X_train['teacher_prefix0'].values.reshape(-1,1))
```

```
X_train_teacher_ohe = normalizer.transform(X_train['teacher_prefix0'].values.reshape(-1,1))
X_cv_teacher_ohe = normalizer.transform(X_cv['teacher_prefix0'].values.reshape(-1,1))
X_test_teacher_ohe = normalizer.transform(X_test['teacher_prefix0'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Teacher Prefix [1]

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_prefix1'].values.reshape(1,-1))

X_train_teacher1_ohe = normalizer.transform(X_train['teacher_prefix1'].values.reshape(-1,1))
X_cv_teacher1_ohe = normalizer.transform(X_cv['teacher_prefix1'].values.reshape(-1,1))
X_test_teacher1_ohe = normalizer.transform(X_test['teacher_prefix1'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_teacher1_ohe.shape, y_train.shape)
print(X_cv_teacher1_ohe.shape, y_cv.shape)
print(X_test_teacher1_ohe.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Numerical Feature Vectorizing

Price

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

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

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

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Teacher_number_of_previously_posted_projects

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

```
X_train_teacher_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_proj
X_cv_teacher_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects']
X_test_teacher_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projec
```

```
print("After vectorizations")
print(X_train_teacher_norm.shape, y_train.shape)
print(X_cv_teacher_norm.shape, y_cv.shape)
print(X_test_teacher_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Positive

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

X_train_positive_norm = normalizer.transform(X_train['positive'].values.reshape(-1,1))
X_cv_positive_norm = normalizer.transform(X_cv['positive'].values.reshape(-1,1))
X_test_positive_norm = normalizer.transform(X_test['positive'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_positive_norm.shape, y_train.shape)
print(X_cv_positive_norm.shape, y_cv.shape)
print(X_test_positive_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Negative

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

X_train_negitive_norm = normalizer.transform(X_train['negitive'].values.reshape(-1,1))
X_cv_negitive_norm = normalizer.transform(X_cv['negitive'].values.reshape(-1,1))
X_test_negitive_norm = normalizer.transform(X_test['negitive'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_negitive_norm.shape, y_train.shape)
print(X_cv_negitive_norm.shape, y_cv.shape)
print(X_test_negitive_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Neutral

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

X_train_neutral_norm = normalizer.transform(X_train['neutral'].values.reshape(-1,1))
X_cv_neutral_norm = normalizer.transform(X_cv['neutral'].values.reshape(-1,1))
X_test_neutral_norm = normalizer.transform(X_test['neutral'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_neutral_norm.shape, y_train.shape)
print(X_cv_neutral_norm.shape, y_cv.shape)
print(X_test_neutral_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Set 1: Categorical, Numerical features + preprocessed_eassay (TFIDF)+ Sentiment Score of eassay

```
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfid, X_train_clean_cat_ohe, X_train_clean_cat1_ohe, X_train_cle
               X_train_grade_ohe, X_train_grade1_ohe, X_train_state_ohe, X_train_state1_ohe,
               X_train_price_norm, X_train_teacher_norm, X_train_positive_norm, X_train_negit

X_cr = hstack((X_cv_essay_tfid, X_cv_clean_cat_ohe, X_cv_clean_cat1_ohe, X_cv_clean_subcat_oh
               X_cv_grade1_ohe, X_cv_state_ohe, X_cv_state1_ohe, X_cv_teacher_ohe, X_cv_teach
               X_cv_positive_norm, X_cv_negitive_norm, X_cv_neutral_norm)).tocsr()

X_te = hstack((X_test_essay_tfid, X_test_clean_cat_ohe, X_test_clean_cat1_ohe, X_test_clean_s
               X_test_grade_ohe, X_test_grade1_ohe, X_test_state_ohe, X_test_state1_ohe, X_te
               X_test_price_norm, X_test_teacher_norm, X_test_positive_norm, X_test_negitive_

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 5015) (22445,)
(11055, 5015) (11055,)
(16500, 5015) (16500,)
====================================================================================================
```

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

Apply GBDT on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

TFID

Hyper paramter tuning

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred


from sklearn.model_selection import train_test_split
from xgboost.sklearn import XGBClassifier
from sklearn.model_selection import GridSearchCV
import seaborn as sea
DT = XGBClassifier()

grid_params = {'learning_rate' : [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3] , 'n_estimators' : [5,1


classifier = GridSearchCV(DT, grid_params, cv=3, scoring='roc_auc')
classifier.fit(X_tr, y_train)

     GridSearchCV(cv=3, error_score=nan,
                  estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                          colsample_bylevel=1, colsample_bynode=1,
                                          colsample_bytree=1, gamma=0,
                                          learning_rate=0.1, max_delta_step=0,
                                          max_depth=3, min_child_weight=1,
                                          missing=None, n_estimators=100, n_jobs=1,
                                          nthread=None, objective='binary:logistic',
                                          random_state=0, reg_alpha=0, reg_lambda=1,
```

```
                                scale_pos_weight=1, seed=None, silent=None,
                                subsample=1, verbosity=1),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3],
                         'n_estimators': [5, 10, 50, 75, 100, 200]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

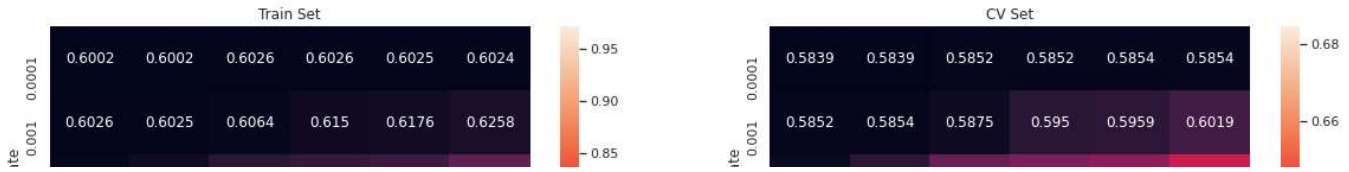
Seaborn heat maps

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from xgboost.sklearn import XGBClassifier
dt1 = XGBClassifier(class_weight = 'balanced')

grid_params = {'learning_rate' : [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3] , 'n_estimators' : [5,1

clf1 = GridSearchCV(dt1, grid_params, cv=3, scoring='roc_auc',return_train_score=True)
se1 = clf1.fit(X_tr, y_train)


import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf1.cv_results_).groupby(['param_learning_rate' , 'param_n_estima
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```




```
print(clf1.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf1.score(X_tr ,y_train))
print(clf1.score(X_te ,y_test))
```

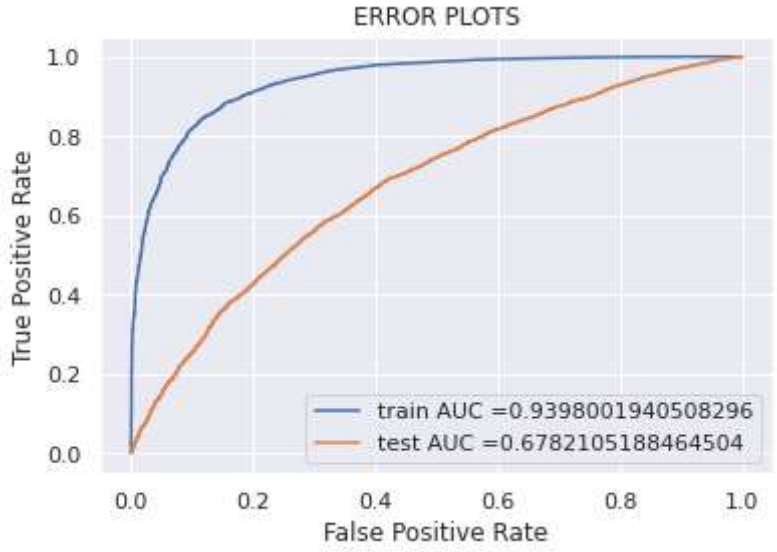
```
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=200, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
0.8621340131260998
0.6877604346191591
```

```
# Best tune parameters
best_tune_parameters=[{'learning_rate': [0.3], 'n_estimators':[200] } ]
```

Finding Best AUC

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.met
from sklearn.metrics import roc_curve, auc
from xgboost.sklearn import XGBClassifier
from sklearn.metrics import roc_curve, auc
clf11= GridSearchCV(XGBClassifier(class_weight = 'balanced'),best_tune_parameters)
clfV1=XGBClassifier (class_weight = 'balanced',max_depth=5, n_estimators=50)
clf11.fit(X_tr, y_train)
# for visulation
clfV1.fit(X_tr, y_train)
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#skl
y_train_pred1 = clf11.predict_proba(X_tr) [:,1]
y_test_pred1 = clf11.predict_proba(X_te) [:,1]
train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)
plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))


plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid(True)
plt.show()
```

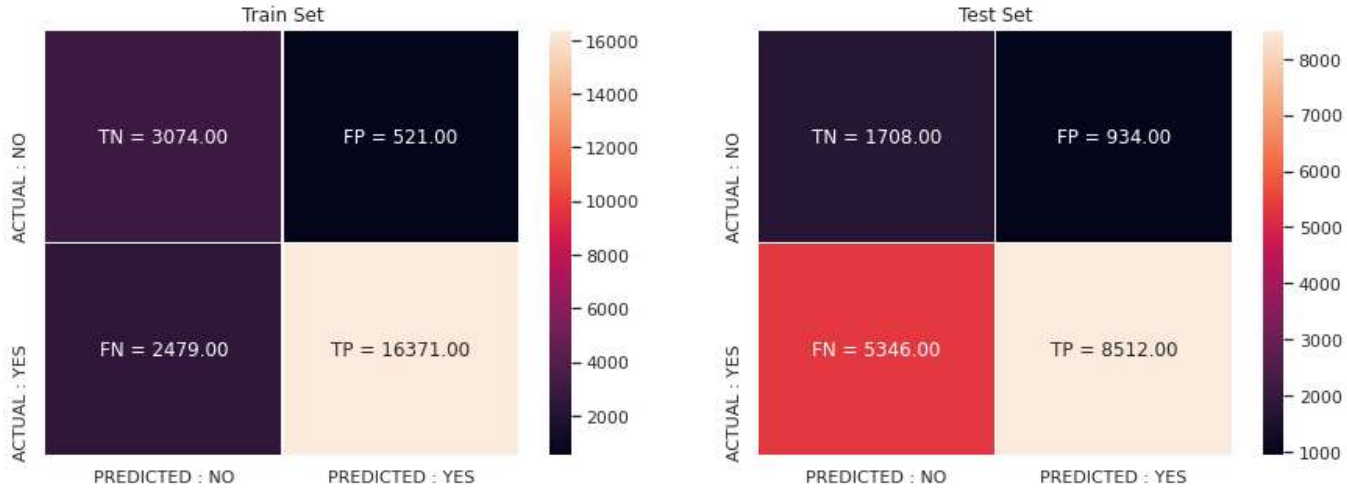


Confusion Matrix

```
def predict(proba, threshould, fpr, tpr):
    t = threshould[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", np.round(max(tpr*(1-fpr)),2) , "for threshold",
    predictions = []
    global predictions1 # make it global
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1= predictions
    return predictions


import seaborn as sns;
sns.set()
con_m_train = confusion_matrix(y_train, predict(y_train_pred1, tr_thresholds1, train_fpr1, tr
con_m_test = confusion_matrix(y_test, predict(y_test_pred1, te_thresholds1, test_fpr1, test_t
key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(15,5))
labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.fla
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flat
sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],yti
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],ytic
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.75 for threshold 0.79
the maximum value of tpr*(1-fpr) 0.4 for threshold 0.85
```



TFIDF W2V

Set 2: Categorical, numerical features +preprocessed_eassay (TFIDF W2V) + Sentiment Score of eassay

```
from scipy.sparse import hstack
X_tr1 = hstack((X_train_essay_tfid, X_train_clean_cat_ohe, X_train_clean_cat1_ohe, X_train_cl
               X_train_grade_ohe, X_train_grade1_ohe, X_train_state_ohe, X_train_state1_ohe,
               X_train_price_norm, X_train_teacher_norm, X_train_positive_norm, X_train_negit

X_cr1 = hstack((X_cv_essay_tfid, X_cv_clean_cat_ohe, X_cv_clean_cat1_ohe, X_cv_clean_subcat_o
               X_cv_grade1_ohe, X_cv_state_ohe, X_cv_state1_ohe, X_cv_teacher_ohe, X_cv_teach
               X_cv_positive_norm, X_cv_negitive_norm, X_cv_neutral_norm)).tocsr()

X_te1 = hstack((X_test_essay_tfid, X_test_clean_cat_ohe, X_test_clean_cat1_ohe, X_test_clean_
               X_test_grade_ohe, X_test_grade1_ohe, X_test_state_ohe, X_test_state1_ohe, X_te
               X_test_price_norm, X_test_teacher_norm, X_test_positive_norm, X_test_negitive_

print("Final Data matrix")
print(X_tr1.shape, y_train.shape)
print(X_cr1.shape, y_cv.shape)
print(X_te1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 5015) (22445,)
(11055, 5015) (11055,)
```

```
(16500, 5015) (16500,)
==========================================================================================
```

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred


from sklearn.model_selection import train_test_split
from xgboost.sklearn import XGBClassifier
from sklearn.model_selection import GridSearchCV
import seaborn as sea
DT = XGBClassifier()

grid_params = {'learning_rate' : [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3] , 'n_estimators' : [5,1


classifier = GridSearchCV(DT, grid_params, cv=3, scoring='roc_auc')
classifier.fit(X_tr1, y_train)
```

```
GridSearchCV(cv=3, error_score=nan,
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3],
                         'n_estimators': [5, 10, 50, 75, 100, 200]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

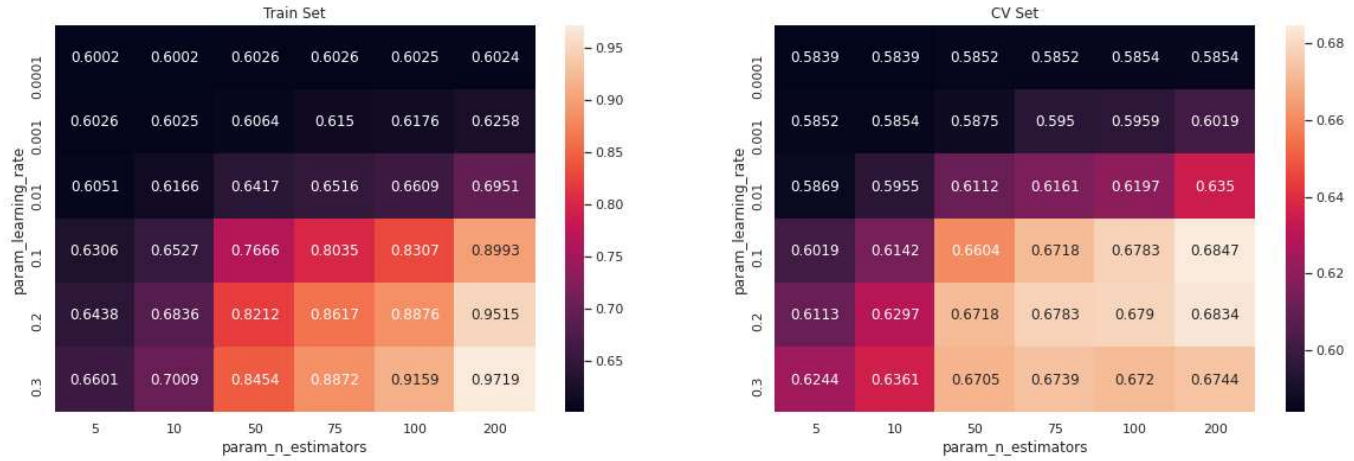
Seaborn heat maps

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from xgboost.sklearn import XGBClassifier
dt1 = XGBClassifier(class_weight = 'balanced')

grid_params = {'learning_rate' : [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3] , 'n_estimators' : [5,1

clf1 = GridSearchCV(dt1, grid_params, cv=3, scoring='roc_auc',return_train_score=True)
se1 = clf1.fit(X_tr1, y_train)


import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf1.cv_results_).groupby(['param_learning_rate' , 'param_n_estima
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```




```
print(clf1.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf1.score(X_tr1 ,y_train))
print(clf1.score(X_te1 ,y_test))
```

```
      XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
                    colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
                    gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=5,
                    min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
                    nthread=None, objective='binary:logistic', random_state=0,
                    reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                    silent=None, subsample=1, verbosity=1)
      0.9039404345115342
      0.6800961143791211


best_tune_parameters=[{'learning_rate': [0.3], 'n_estimators':[200] } ]
```

Finding Best AUC

```
# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.met
from sklearn.metrics import roc_curve, auc
from xgboost.sklearn import XGBClassifier
from sklearn.metrics import roc_curve, auc
clf11= GridSearchCV(XGBClassifier(class_weight = 'balanced'),best_tune_parameters)
clfV1=XGBClassifier (class_weight = 'balanced',max_depth=5, n_estimators=50)
clf11.fit(X_tr1, y_train)
# for visulation
clfV1.fit(X_tr1, y_train)
#https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#skl
y_train_pred1 = clf11.predict_proba(X_tr1) [:,1]
y_test_pred1 = clf11.predict_proba(X_te1) [:,1]
train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)
plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))


plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid(True)
plt.show()
```

Confusion Matrix

```
def predict(proba, threshould, fpr, tpr):
    t = threshould[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", np.round(max(tpr*(1-fpr)),2) , "for threshold",
    predictions = []
    global predictions1 # make it global
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1= predictions
    return predictions


import seaborn as sns;
sns.set()
con_m_train = confusion_matrix(y_train, predict(y_train_pred1, tr_thresholds1, train_fpr1, tr
con_m_test = confusion_matrix(y_test, predict(y_test_pred1, te_thresholds1, test_fpr1, test_t
key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(15,5))
labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.fla
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flat
sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],yti
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],ytic
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.75 for threshold 0.79
the maximum value of tpr*(1-fpr) 0.4 for threshold 0.85
```

CONCLUSION

3. Summary

as mentioned in the step 4 of instructions

```
from prettytable import PrettyTable
tb = PrettyTable()
tb.field_names= (" Vectorizer ", " Max_depth ", " Min_sample_split "," Test -AUC ")
tb.add_row([" Tf - Idf", 5 , 50 ,0.66 ])

tb.add_row(["A VGW - Tf - Idf", 10 , 100 ,0.67])

print(tb.get_string(titles = "GBDT - Observations"))
```

```
+------------------+-------------+--------------------+-------------+
|    Vectorizer    |  Max_depth  |  Min_sample_split  |  Test -AUC  |
+------------------+-------------+--------------------+-------------+
|     Tf - Idf     |      5      |         50         |     0.66    |
| A VGW - Tf - Idf |      10     |        100         |     0.67    |
+------------------+-------------+--------------------+-------------+
```