Assignment 9: GBDT

Response Coding: Example

Train Data

State	class
A	0
B	1
C	1
A	0
A	1
B	1
A	0
A	1
C	1
C	0

Resonse table(only from train)

State	Class=0	Class=1
A	3	2
B	0	2
C	1	2

Encoded Train Data

State_0	State_1	class
3/5	2/5	0
0/2	2/2	1
1/3	2/3	1
3/5	2/5	0
3/5	2/5	1
0/2	2/2	1
3/5	2/5	0
3/5	2/5	1
1/3	2/3	1
1/3	2/3	0

Test Data

State
A
C
D
C
B
E

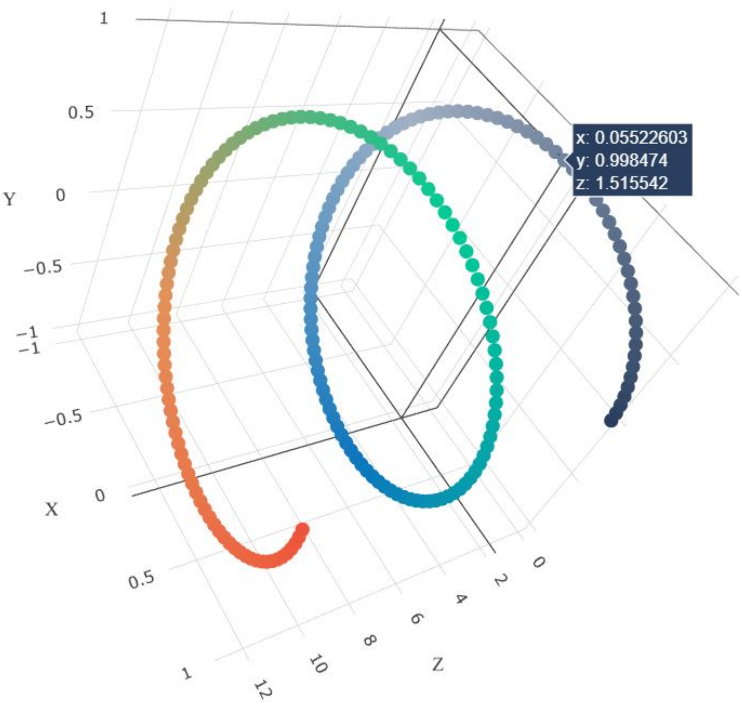
Encoded Test Data

State_0	State_1
3/5	2/5
1/3	2/3
1/2	1/2
1/3	2/3
0/2	2/2
1/2	1/2

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

1. **Apply GBDT on these feature sets**
    - Set 1: categorical(instead of one hot encoding, try [response coding](): use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check the bellow example, include all 4 values as 4 features)
    - Set 2: categorical(instead of one hot encoding, try [response coding](): use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

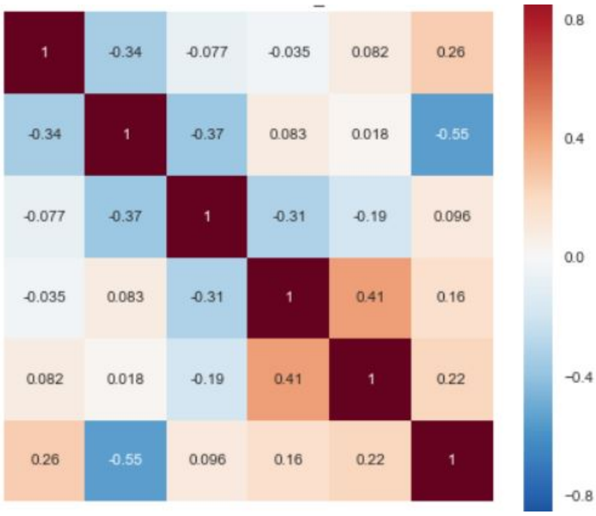
2. **The hyper paramter tuning (Consider any two hyper parameters)**
    - Find the best hyper parameter which will give the maximum AUC value
    - find the best hyper paramter using k-fold cross validation/simple cross validation data
    - use gridsearch cv or randomsearch cv or you can write your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



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

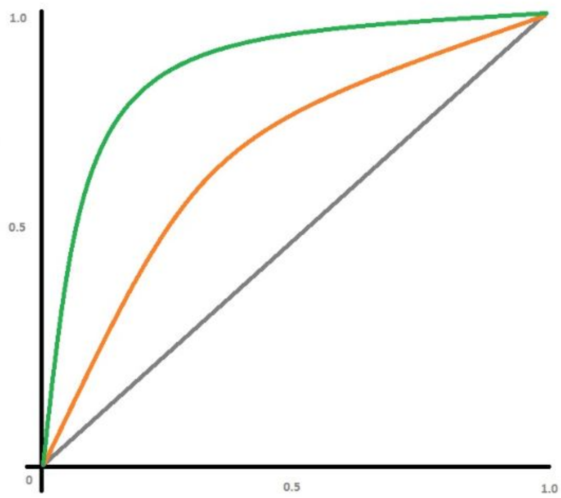
or

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



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

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



- Along with plotting ROC curve, you need to print the [confusion matrix](#) with predicted

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

and original labels of test data points

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
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 st
for learning my students learn in many different ways using all of our senses and multiple
of techniques to help all my students succeed students in my class come from a variety of
for wonderful sharing of experiences and cultures including native americans our school is
learners which can be seen through collaborative student project based learning in and out
in my class love to work with hands on materials and have many different opportunities to
mastered having the social skills to work cooperatively with friends is a crucial aspect o
montana is the perfect place to learn about agriculture and nutrition my students love to
in the early childhood classroom i have had several kids ask me can we try cooking with re
and create common core cooking lessons where we learn important math and writing concepts
food for snack time my students will have a grounded appreciation for the work that went i
of where the ingredients came from as well as how it is healthy for their bodies this proj
nutrition and agricultural cooking recipes by having us peel our own apples to make homema
and mix up healthy plants from our classroom garden in the spring we will also create our
shared with families students will gain math and literature skills as well as a life long
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
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

1. GBDT (xgboost/lightgbm)

```
%matplotlib inline
```

```
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

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

1.1 Loading Data

```
from google.colab import files
files=files.upload()
```

Choose Files preprocessed_data.csv

- **preprocessed_data.csv**(application/vnd.ms-excel) - 124454659 bytes, last modified: 11/11/2019 - 100% done

Saving preprocessed_data.csv to preprocessed_data (1).csv

```
preprocessed_data= pd.read_csv("preprocessed_data (1).csv")
```

```
preprocessed_data.head(3)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previousl
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	
2	ca	mrs	grades_prek_2	

```
sid    = SentimentIntensityAnalyzer()

negative_sentiments = []
positive_sentiments = []
neutral_sentiments = []
compound_sentiments = []

for i in tqdm(preprocessed_data['essay']):
  sid_sentiments = sid.polarity_scores(i)
  negative_sentiments.append(sid_sentiments['neg'])
  positive_sentiments.append(sid_sentiments['pos'])
  neutral_sentiments.append(sid_sentiments['neu'])
  compound_sentiments.append(sid_sentiments['compound'])

# Now append these sentiments columns/freatures to original preprocessed dataframe
preprocessed_data['negative_sent'] = negative_sentiments
preprocessed_data['positive_sent'] = positive_sentiments
preprocessed_data['neutral_sent'] = neutral_sentiments
preprocessed_data['compound_sent'] = compound_sentiments

preprocessed_data.head(1)
```

```
100%|██████████| 109248/109248 [03:30<00:00, 519.42it/s]
```

```
y = preprocessed_data['project_is_approved'].values
X = preprocessed_data.drop(['project_is_approved'], axis = 1)
X.head(2)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previousl
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	

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

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

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)


print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(73196, 12)
(36052, 12)
(73196,)
(36052,)
```

1.3 Make Data Model Ready: encoding "essay"

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4)) #Apply Tfidf Vectorizer
vectorizer.fit(X_train['essay'].values)

X_train_essay_Tfidf = vectorizer.transform(X_train['essay'].values)
X_test_essay_Tfidf= vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_Tfidf.shape, y_train.shape)
print(X_test_essay_Tfidf.shape, y_test.shape)
```

```
After vectorizations
(73196, 259010) (73196,)
(36052, 259010) (36052,)
```

```
def response_encoding_fit(x_train_feature_total, x_train_feature_0, x_train_feature_1):
  feature_counter_total = Counter()
  feature_counter_total.update(i for i in x_train_feature_total)

                                                  # For 'school_state' feature, abov
                                                  # Includes both class 0 and 1 in t
                                                  # ({'ca': 469, 'mi': 80, 'ny': 199
                                                  # Noting The counter is a sub-clas
                                                  # we can count the key-value pairs

  feature_counter_0 = Counter()                   # Create a dict variable to act as
  feature_counter_0.update(i for i in x_train_feature_0)# adding values to the Counter fro

                                                  # it will be of form => Counter({'
                                                  # Now update feature_counter_0 wit
                                                  # i.e. keys that exist in feature_

  for i in feature_counter_total:                 # i is each key (e.g. 'ca', 'fl' e
    if i not in feature_counter_0:                # If a key is not there in feature
      feature_counter_0[i] = 0                    # set the value of that key be 0 i

                                                  # Similary do the same for x_train

  feature_counter_1 = Counter()
  feature_counter_1.update(i for i in x_train_feature_1)

  for i in feature_counter_total:
    if i not in feature_counter_1:
      feature_counter_1[i] = 0

  return feature_counter_total, feature_counter_0, feature_counter_1


  """ Now Function to tranform (generate proba array) for response-encoded categorical fea

args:
  x_feature_train => X_train['feature_name']
  feature counter 0 and feature counter 1 => These are Counter / dict variable returned fr
```

```
  returns:
     List of Probabilities => Of form =>
     [[0.04761905]
      [0.16981132]
      [0.16981132]]
"""

def response_encoding_transform(x_feature_train, feature_counter_total, feature_counter_0,
  feature_proba_arr_0 = []
  feature_proba_arr_1 = []

  for i in x_feature_train:
    # Now loop over each feature-name e.g. 'ca', 'fl' etc for school_state
    if i in feature_counter_total.keys(): # if the specific unique feature-names exist in
      # .get(i) will give me the value of the key, i.e. the number count for each key (whi
      proba_0 = feature_counter_0.get(i)/feature_counter_total.get(i)
      proba_1 = feature_counter_1.get(i)/feature_counter_total.get(i)

      feature_proba_arr_0.append(proba_0)
      feature_proba_arr_1.append(proba_1)
    else:
      feature_proba_arr_0.append(0.5)
      feature_proba_arr_1.append(0.5)
  # Have to convert to array so I can invoke reshape() on these
  feature_proba_arr_0 = np.array(feature_proba_arr_0)
  feature_proba_arr_1 = np.array(feature_proba_arr_1)

  return feature_proba_arr_0.reshape(-1, 1), feature_proba_arr_1.reshape(-1, 1)


# Now make a new dataframe for all the categorical feature from only the train dataset
# And then I will response-encode these dataset.
# Categorial Featues are => school_state, teacher_prefix, project_grade_category, clean_ca

df_cat_train_before_response_coding = pd.DataFrame(y_train, columns=['project_is_approved'
df_cat_train_before_response_coding['school_state'] = X_train['school_state'].values
df_cat_train_before_response_coding['teacher_prefix'] = X_train['teacher_prefix'].values
df_cat_train_before_response_coding['project_grade_category'] = X_train['project_grade_cat
df_cat_train_before_response_coding['clean_categories'] = X_train['clean_categories'].valu
df_cat_train_before_response_coding['clean_subcategories'] = X_train['clean_subcategories'
df_cat_train_before_response_coding.head(3)
```

	project_is_approved	school_state	teacher_prefix	project_grade_category	clean_
0	1	il	ms	grades_9_12	h
1	1	nc	mrs	grades_3_5	litera

Encoding Categorical Variable using Response Coding: "school_state"

```
x_train_feature_total = df_cat_train_before_response_coding['school_state']
x_train_feature_0 = df_cat_train_before_response_coding.loc[df_cat_train_before_response_c
x_train_feature_1 = df_cat_train_before_response_coding.loc[df_cat_train_before_response_c

school_state_counter_total, school_state_counter_0, school_state_counter_1 = response_enco

X_train_school_state_response_proba_0, X_train_school_state_response_proba_1 = response_en

X_test_school_state_response_proba_0, X_test_school_state_response_proba_1 = response_enco

print(np.mean(X_train_school_state_response_proba_0, axis=0))
print(X_train_school_state_response_proba_0.shape, y_train.shape)
print(X_test_school_state_response_proba_0.shape, y_test.shape)
```

```
[0.15141538]
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Encoding Categorical Variable using Response Coding:"project_grade_category""

```
x_train_feature_total = df_cat_train_before_response_coding['project_grade_category']
x_train_feature_0 = df_cat_train_before_response_coding.loc[df_cat_train_before_response_c
x_train_feature_1 = df_cat_train_before_response_coding.loc[df_cat_train_before_response_c

project_grade_category_counter_total, project_grade_category_counter_0, project_grade_cate

X_train_project_grade_category_response_proba_0, X_train_project_grade_category_response_p

X_test_project_grade_category_response_proba_0, X_test_project_grade_category_response_pro

print(np.mean(X_train_project_grade_category_response_proba_0, axis=0))
print(X_train_project_grade_category_response_proba_0.shape, y_train.shape)
print(X_test_project_grade_category_response_proba_0.shape, y_test.shape)
```

```
[0.15141538]
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Encoding Categorical Variable using Response Coding:"clean_subcategories"

```
x_train_feature_total = df_cat_train_before_response_coding['clean_subcategories']
x_train_feature_0 = df_cat_train_before_response_coding.loc[df_cat_train_before_response_c
x_train_feature_1·=·df_cat_train_before_response_coding.loc[df_cat_train_before_response_c

clean_subcategories_counter_total, clean_subcategories_counter_0, clean_subcategories_coun

X_train_clean_subcategories_response_proba_0, X_train_clean_subcategories_response_proba_1

X_test_clean_subcategories_response_proba_0, X_test_clean_subcategories_response_proba_1 =

print(np.mean(X_train_clean_subcategories_response_proba_0, axis=0))
print(X_train_clean_subcategories_response_proba_0.shape, y_train.shape)
print(X_test_clean_subcategories_response_proba_0.shape, y_test.shape)
```

```
[0.15141538]
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Encoding Categorical Variable using Response Coding:"clean_categories"

```
x_train_feature_total = df_cat_train_before_response_coding['clean_categories']
x_train_feature_0 = df_cat_train_before_response_coding.loc[df_cat_train_before_response_c
x_train_feature_1 = df_cat_train_before_response_coding.loc[df_cat_train_before_response_c

clean_categories_counter_total, clean_categories_counter_0, clean_categories_counter_1 = r

X_train_clean_categories_response_proba_0, X_train_clean_categories_response_proba_1 = res

X_test_clean_categories_response_proba_0, X_test_clean_categories_response_proba_1 = respo

print(np.mean(X_train_clean_categories_response_proba_0, axis=0))
print(X_train_clean_categories_response_proba_0.shape, y_train.shape)
print(X_test_clean_categories_response_proba_0.shape, y_test.shape)
```

```
[0.15141538]
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Encoding Categorical Variable using Response Coding:"project_grade_category"

```
x_train_feature_total = df_cat_train_before_response_coding['project_grade_category']
```

```
x_train_feature_0 = df_cat_train_before_response_coding.loc[df_cat_train_before_response_c
x_train_feature_1 = df_cat_train_before_response_coding.loc[df_cat_train_before_response_c

project_grade_category_counter_total, project_grade_category_counter_0, project_grade_cate

X_train_project_grade_category_response_proba_0, X_train_project_grade_category_response_p

X_test_project_grade_category_response_proba_0, X_test_project_grade_category_response_pro

print(np.mean(X_train_project_grade_category_response_proba_0, axis=0))
print(X_train_project_grade_category_response_proba_0.shape, y_train.shape)
print(X_test_project_grade_category_response_proba_0.shape, y_test.shape)
```

```
[0.15141538]
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Encoding Categorical Variable using Response Coding:"teacher_prefix"

```
x_train_feature_total = df_cat_train_before_response_coding['teacher_prefix']
x_train_feature_0 = df_cat_train_before_response_coding.loc[df_cat_train_before_response_c
x_train_feature_1 = df_cat_train_before_response_coding.loc[df_cat_train_before_response_c

teacher_prefix_counter_total, teacher_prefix_counter_0, teacher_prefix_counter_1 = respons

X_train_teacher_prefix_response_proba_0, X_train_teacher_prefix_response_proba_1 = respons

X_test_teacher_prefix_response_proba_0, X_test_teacher_prefix_response_proba_1 = response_

print(np.mean(X_train_teacher_prefix_response_proba_0, axis=0))
print(X_train_teacher_prefix_response_proba_0.shape, y_train.shape)
print(X_test_teacher_prefix_response_proba_0.shape, y_test.shape)
```

```
[0.15141538]
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Encoding Numerical features using tfidf: "teacher_number_of_previously_posted_projects"

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
```

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

X_train_project_teachers_norm = normalizer.transform(X_train['teacher_number_of_previously
X_test_project_teachers_norm = normalizer.transform(X_test['teacher_number_of_previously_p

print("After Normalization")
print(X_train_project_teachers_norm.shape, y_train.shape)
print(X_test_project_teachers_norm.shape, y_test.shape)
```

```
After Normalization
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Encoding Numerical features using tfidf: "price"

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

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

print("After Normalization")
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
```

```
After Normalization
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Standardize and then .fit() and .transform() all the Sentiments related Columns

```
from sklearn.preprocessing import StandardScaler, Normalizer
sentiments_standardizer = StandardScaler()

# First applying the .fit() on the train data to find Mean and SD
sentiments_standardizer.fit(X_train['negative_sent'].values.reshape(-1,1))

# Now applying .transform() to train, test and cv data
X_train_negative_sent_standardized = sentiments_standardizer.transform(X_train['negative_s
X_test_negative_sent_standardized = sentiments_standardizer.transform(X_test['negative_sen

print('After Standardizing on negative_sent column checking the shapes ')
print(X_train_negative_sent_standardized.shape, y_train.shape)
print(X_test_negative_sent_standardized.shape, y_test.shape)
```

```
After Standardizing on negative_sent column checking the shapes
```

```
(73196, 1) (73196,)
(36052, 1) (36052,)
```

```
# First applying the .fit() on the train data to find Mean and SD
sentiments_standardizer.fit(X_train['positive_sent'].values.reshape(-1,1))

# Now applying .transform() to train, test and cv data
X_train_positive_sent_standardized = sentiments_standardizer.transform(X_train['positive_s
X_test_positive_sent_standardized = sentiments_standardizer.transform(X_test['positive_sen

print('After Standardizing on positive_sent column checking the shapes ')
print(X_train_positive_sent_standardized.shape, y_train.shape)
print(X_test_positive_sent_standardized.shape, y_test.shape)
```

```
After Standardizing on positive_sent column checking the shapes
(73196, 1) (73196,)
(36052, 1) (36052,)
```

```
# First applying the .fit() on the train data to find Mean and SD
sentiments_standardizer.fit(X_train['neutral_sent'].values.reshape(-1,1))

# Now applying .transform() to train, test and cv data
X_train_neutral_sent_standardized = sentiments_standardizer.transform(X_train['neutral_sen
X_test_neutral_sent_standardized = sentiments_standardizer.transform(X_test['neutral_sent'

print('After Standardizing on neutral_sent column checking the shapes ')
# print('X_train_neutral_sent_standardized ', X_train_neutral_sent_standardized)
print(X_train_neutral_sent_standardized.shape, y_train.shape)
print(X_test_neutral_sent_standardized.shape, y_test.shape)
```

```
After Standardizing on neutral_sent column checking the shapes
(73196, 1) (73196,)
(36052, 1) (36052,)
```

```
# First applying the .fit() on the train data to find Mean and SD
sentiments_standardizer.fit(X_train['compound_sent'].values.reshape(-1,1))

# Now applying .transform() to train, test and cv data
X_train_compound_sent_standardized = sentiments_standardizer.transform(X_train['compound_s
X_test_compound_sent_standardized = sentiments_standardizer.transform(X_test['compound_sen

print('After Standardizing on compound_sent column checking the shapes ')
# print('X_train_compound_sent_standardized ', X_train_compound_sent_standardized)
print(X_train_compound_sent_standardized.shape, y_train.shape)
print(X_test_compound_sent_standardized.shape, y_test.shape)
```

```
After Standardizing on compound_sent column checking the shapes
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Concatenating all the Features:(Tfidf)

```
from scipy.sparse import hstack

X_train1 = hstack((X_train_essay_Tfidf, X_train_school_state_response_proba_0, X_train_tea
X_test1 = hstack((X_test_essay_Tfidf, X_test_school_state_response_proba_0, X_test_teacher

print("Final Data matrix")
print(X_train1.shape, y_train.shape)
print(X_test1.shape, y_test.shape)
```

```
Final Data matrix
(73196, 259021) (73196,)
(36052, 259021) (36052,)
```

Apply GBDT on these feature sets:

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

- 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)

AUC for Set S1:

```
import math as mt
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from scipy.stats import expon
from collections import Counter
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score

# xgb_clf_s1 = XGBClassifier(booster='gblinear', reg_alpha=0, reg_lambda=0, eval_metric='m
xgb_clf_s1 = XGBClassifier(booster='gblinear', reg_alpha=0, reg_lambda=0, tree_method='gpu

#xgb_clf_s1 = XGBClassifier(eval_metric='mlogloss')

params = {'eta': [0.0001, 0.001, 0.01, 0.1, 0.2],'n_estimators': [30, 40, 50, 60],'tree_me
```

```
# params = {
# 'eta': [0.0001, 0.001, 0.01, 0.1, 0.2],
# 'n_estimators': [30, 40, 50, 60]
# }

grid_search_s1 = GridSearchCV(xgb_clf_s1, params, cv=3, scoring='roc_auc', return_train_sc

grid_search_s1.fit(X_train1, y_train)

results = pd.DataFrame.from_dict(grid_search_s1.cv_results_)

best_params_gridsearch_xgb_s1 = grid_search_s1.best_params_

print("Best Params from GridSearchCV with XGB for Set s1 ", best_params_gridsearch_xgb_s1)
```

```
Best Params from GridSearchCV with XGB for Set s1  {'eta': 0.0001, 'n_estimators': 6(
```

```
train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
#max_depth= results['param_max_depth']
#min_samples_split=results['param_min_samples_split']


print("Train AUC= ",train_auc)
print(50*'-')
print("Std Train Score= ",train_auc_std)
print(50*'-')
print("CV AUC= ",cv_auc)
print(50*'-')
print("CV AUC STD=",cv_auc_std)
print(50*'-')
#print("Maximum Depth of the Tree=",max_depth)
print(50*'-')
#print("Minimum Samples Split= ",min_samples_split)
```

```
8     9.146594e-08
9     1.450134e-07
10    1.580999e-07
11    1.652372e-07
12    9.146594e-08
13    1.450134e-07
14    1.580999e-07
15    1.652372e-07
16    9.146594e-08
17    1.450134e-07
18    1.580999e-07
19    1.652372e-07
Name: std_train_score, dtype: float64
--------------------------------------------------
CV AUC=  0     0.635477
1     0.636883
2     0.637689
3     0.637982
4     0.635477
```

```
5     0.636883
6     0.637689
7     0.637982
8     0.635477
9     0.636883
10    0.637689
11    0.637982
12    0.635477
13    0.636883
14    0.637689
15    0.637982
16    0.635477
17    0.636883
18    0.637689
19    0.637982
Name: mean_test_score, dtype: float64
--------------------------------------------------
CV AUC STD= 0     0.004707
1     0.004828
2     0.004749
3     0.004814
4     0.004707
5     0.004828
6     0.004749
7     0.004814
8     0.004707
9     0.004828
10    0.004749
11    0.004814
12    0.004707
13    0.004828
14    0.004749
15    0.004814
16    0.004707
17    0.004828
18    0.004749
19    0.004814
Name: std_test_score, dtype: float64
--------------------------------------------------
--------------------------------------------------
```

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

```
#from mpl_toolkits import mplot3d
#%matplotlib inline
#import matplotlib.pyplot as plt

#fig = plt.figure()
#ax = plt.axes(projection='3d')
#ax.scatter3D(min_samples_split, max_depth, train_auc, cmap="Black")
#ax.plot3D(min_samples_split, max_depth, train_auc, 'gray')
#ax.set_xlabel('min_samples_split')
#ax.set_ylabel('max_depth')
```

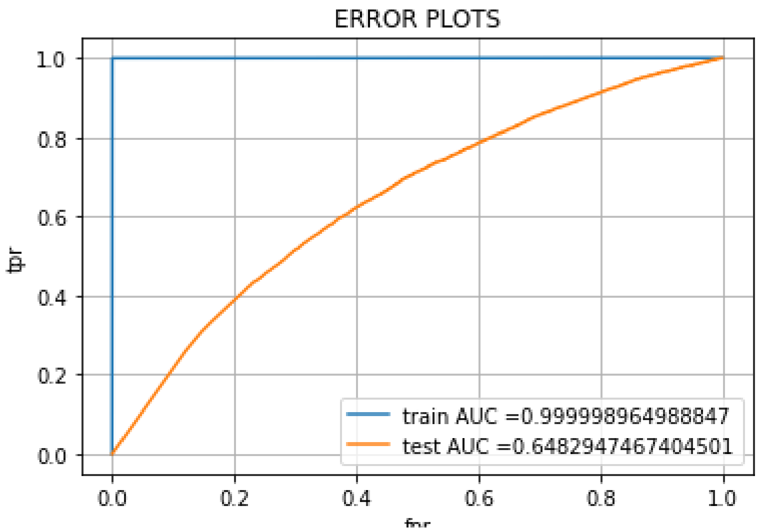
```
#ax.set_zlabel('AUC');
#ax.scatter3D(min_samples_split, max_depth, cv_auc, cmap="Green")
#ax.plot3D(min_samples_split, max_depth, cv_auc, 'Red')
#plt.show()
```

```
grid_search_s1.best_estimator_
```

```
XGBClassifier(base_score=0.5, booster='gblinear', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eta=0.0001,
              eval_metric='mlogloss', gamma=0, learning_rate=0.1,
              max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
              n_estimators=60, n_jobs=1, nthread=None,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=0, scale_pos_weight=1, seed=None, silent=None,
              subsample=1, tree_method='gpu_hist', verbosity=1)
```

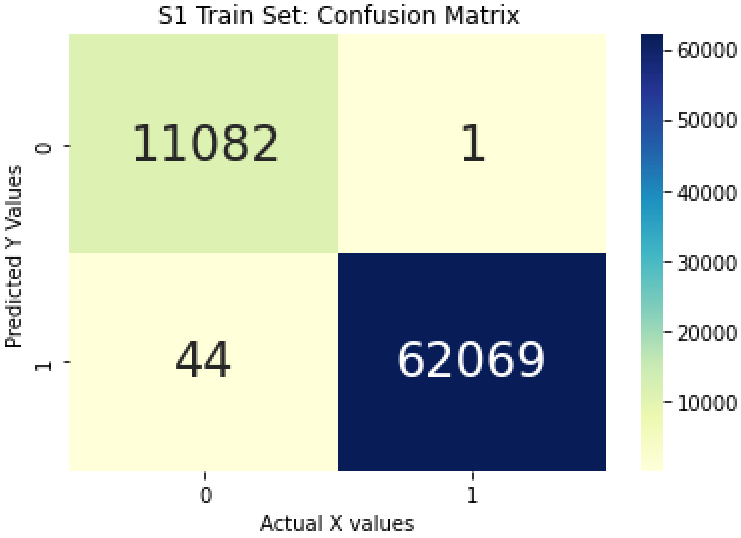
```
def batch_predict(clf, data):
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 73196, then your tr_loop will be 73196 - 73196%1000 = 730
    # in this for loop we will iterate until 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
```

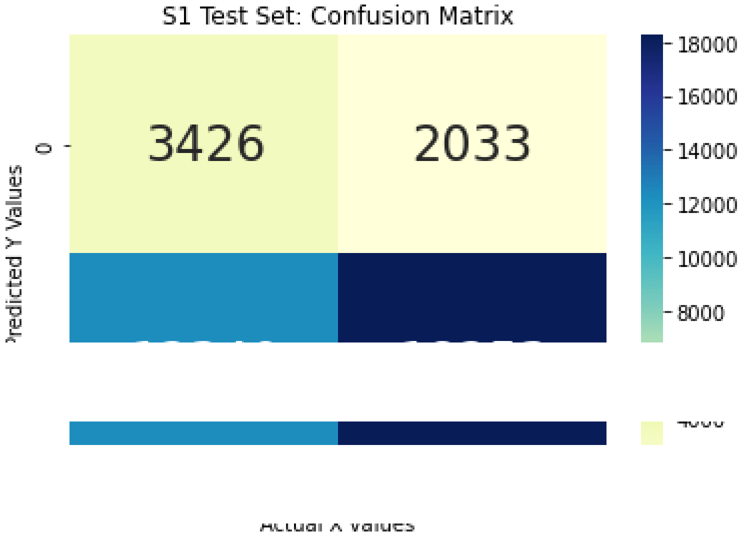
```
#ROC Curve for Set1 :
from sklearn.metrics import roc_curve, auc

Model= XGBClassifier(n_estimators=60, learning_rate=0.1, reg_alpha=0, reg_lambda=0, booste
Model.fit(X_train1, y_train)

y_train_pred = batch_predict(Model, X_train1)
y_test_pred = batch_predict(Model, X_test1)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

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)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```




```
#Confusion Matrix:

def get_predicted_y_vec_from_threshold(proba, threshold, fpr, tpr):
  optimal_threshold = threshold[np.argmax(tpr * (1-fpr))]

  predicted_y_vector = []
  for i in proba:
    if i >= optimal_threshold:
      predicted_y_vector.append(1)
    else:
      predicted_y_vector.append(0)

  return predicted_y_vector

confusion_matrix_s1_train = confusion_matrix(y_train, get_predicted_y_vec_from_threshold(y

confusion_matrix_s1_test = confusion_matrix(y_test, get_predicted_y_vec_from_threshold(y_t

print('confusion_matrix_s1_train ', confusion_matrix_s1_train)
# Heatmap for Confusion Matrix: Train and SET 1
heatmap_confusion_matrix_train_s1 = sns.heatmap(confusion_matrix_s1_train, annot=True, fmt

plt.title('S1 Train Set: Confusion Matrix')
plt.xlabel('Actual X values')
plt.ylabel('Predicted Y Values')
plt.show()

heatmap_confusion_matrix_test_s1 = sns.heatmap(confusion_matrix_s1_test, annot=True, fmt='

plt.title('S1 Test Set: Confusion Matrix')
plt.xlabel('Actual X values')
plt.ylabel('Predicted Y Values')
plt.show()
```

```
confusion_matrix_s1_train  [[11082     1]
 [   44 62069]]
```





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

```
from google.colab import files
files=files.upload()
```

Choose Files glove_vectors

- **glove_vectors**(n/a) - 127506004 bytes, last modified: 9/29/2021 - 100% done

```
Saving glove_vectors to glove_vectors
```

```
#please use below code to load glove vectors

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

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


#Computing tfidf_w2v:

from tqdm import tqdm

tfidf_w2v_vectors = []
for sentence in tqdm(X_train['essay'].values):
  vector = np.zeros(300)
  tf_idf_weight=0;
  for word in sentence.split():
    if (word in glove_words) and (word in tfidf_words):
      vec = model[word]

      tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
      vector += (vec * tf_idf)
      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]))
```

```
100%|██████████| 73196/73196 [02:33<00:00, 476.60it/s]73196
300
```

```
tfidf_w2v_test = [];
for sentence in tqdm(X_test['essay'].values):
  vector = np.zeros(300)
  tf_idf_weight =0;
  for word in sentence.split():
    if (word in glove_words) and (word in tfidf_words):
      vec = model[word]

      tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
      vector += (vec * tf_idf)
      tf_idf_weight += tf_idf
  if tf_idf_weight != 0:
    vector /= tf_idf_weight
  tfidf_w2v_test.append(vector)

print(len(tfidf_w2v_test))
print(len(tfidf_w2v_test[0]))
```

```
100%|██████████| 36052/36052 [01:14<00:00, 482.85it/s]36052
300
```

```
#Convert into Sparse Matrix:
```

```
from scipy import sparse
import numpy as np

X_tr1_w2v= np.hstack((tfidf_w2v_vectors, X_train_school_state_response_proba_0, X_train_te
X_te1_w2v= np.hstack((tfidf_w2v_test, X_test_school_state_response_proba_0, X_test_teacher

print("Final Data matrix")
print(X_tr1_w2v.shape, y_train.shape)
print(X_te1_w2v.shape, y_test.shape)
```

```
Final Data matrix
(73196, 311) (73196,)
(36052, 311) (36052,)
```

```
import math as mt
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.svm import LinearSVC
from scipy.stats import expon
from collections import Counter
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score



# xgb_clf_s2 = XGBClassifier(booster='gblinear', reg_alpha=0, reg_lambda=0, eval_metric='m
# xgb_clf_s2 = XGBClassifier(booster='gblinear', reg_alpha=0, reg_lambda=0, tree_method='g

# xgb_clf_s2 = XGBClassifier(booster='gblinear', reg_alpha=0, reg_lambda=0, eval_metric='m
xgb_clf_s2 = XGBClassifier(booster='gblinear', reg_alpha=0, reg_lambda=0, tree_method='gpu

#xgb_clf_s2 = XGBClassifier(eval_metric='mlogloss')

params = {'eta': [0.0001, 0.001, 0.01, 0.1, 0.2],'n_estimators': [30, 40, 50, 60],'tree_me

# params = {
# 'eta': [0.0001, 0.001, 0.01, 0.1, 0.2],
# 'n_estimators': [30, 40, 50, 60]
# }

grid_search_s2 = GridSearchCV(xgb_clf_s2, params, cv=3, scoring='roc_auc', return_train_sc

grid_search_s2.fit(X_tr1_w2v, y_train)

results1 = pd.DataFrame.from_dict(grid_search_s2.cv_results_)

best_params_gridsearch_xgb_s2 = grid_search_s2.best_params_

print("Best Params from GridSearchCV with XGB for Set s2 ", best_params_gridsearch_xgb_s2)
```

```
Best Params from GridSearchCV with XGB for Set s2  {'eta': 0.0001, 'n_estimators': 6(
```

```
train_auc1= results1['mean_train_score']
train_auc_std1= results1['std_train_score']
cv_auc1 = results1['mean_test_score']
cv_auc_std1= results1['std_test_score']
#max_depth1= results['param_max_depth']
#min_samples_split1=results['param_min_samples_split']


print("Train AUC= ",train_auc1)
print(50*'-')
print("Std Train Score= ",train_auc_std1)
print(50*'-')
print("CV AUC= ",cv_auc1)
print(50*'-')
print("CV AUC STD=",cv_auc_std1)
print(50*'-')
#print("Maximum Depth of the Tree=",max_depth1)
print(50*'-')
#print("Minimum Samples Split= ",min_samples_split1)
```

```
8     0.002608
9     0.002670
10    0.002701
11    0.002724
12    0.002608
13    0.002670
14    0.002701
15    0.002724
16    0.002608
17    0.002670
18    0.002701
19    0.002724
Name: std_train_score, dtype: float64
--------------------------------------------------
CV AUC=  0     0.691190
1     0.693193
2     0.694477
3     0.695377
4     0.691190
5     0.693193
6     0.694477
7     0.695377
8     0.691190
9     0.693193
10    0.694477
11    0.695377
12    0.691190
13    0.693193
14    0.694477
15    0.695377
16    0.691190
17    0.693193
18    0.694477
19    0.695377
Name: mean_test_score, dtype: float64
```

```
--------------------------------------------------
CV AUC STD= 0     0.005000
1     0.005200
2     0.005333
3     0.005438
4     0.005000
5     0.005200
6     0.005333
7     0.005438
8     0.005000
9     0.005200
10    0.005333
11    0.005438
12    0.005000
13    0.005200
14    0.005333
15    0.005438
16    0.005000
17    0.005200
18    0.005333
19    0.005438
Name: std_test_score, dtype: float64
--------------------------------------------------
--------------------------------------------------
```

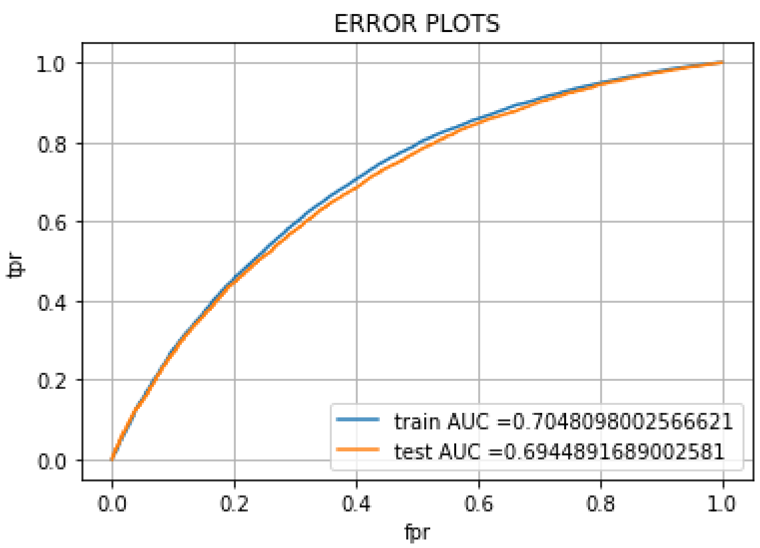
```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

```
#from mpl_toolkits import mplot3d
#%matplotlib inline
#import matplotlib.pyplot as plt

#fig = plt.figure()
#ax = plt.axes(projection='3d')
#ax.scatter3D(min_samples_split, max_depth, train_auc, cmap="Black")
#ax.plot3D(min_samples_split, max_depth, train_auc, 'gray')
#ax.set_xlabel('min_samples_split')
#ax.set_ylabel('max_depth')
#ax.set_zlabel('AUC');
#ax.scatter3D(min_samples_split, max_depth, cv_auc, cmap="Green")
#ax.plot3D(min_samples_split, max_depth, cv_auc, 'Red')
#plt.show()
```

```
grid_search_s2.best_estimator_
```

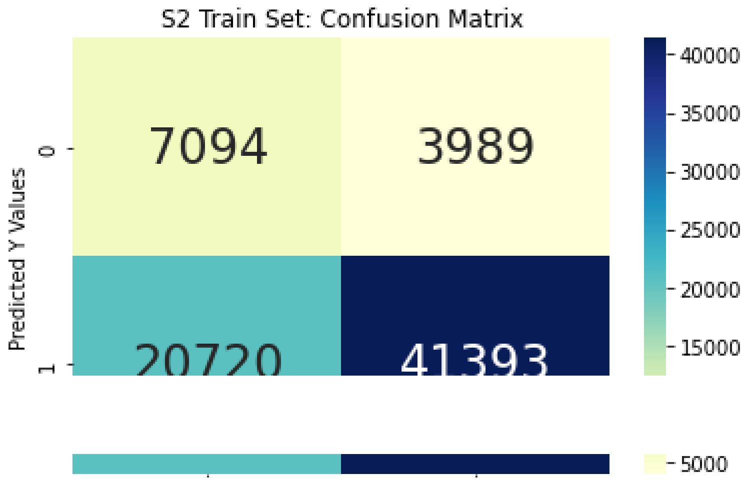
```
XGBClassifier(base_score=0.5, booster='gblinear', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eta=0.0001,
              eval_metric='mlogloss', gamma=0, learning_rate=0.1,
              max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
              n_estimators=60, n_jobs=1, nthread=None,
              objective='binary:logistic', random_state=0, reg_alpha=0,
```

```
                    reg_lambda=0, scale_pos_weight=1, seed=None, silent=None,
                    subsample=1, tree_method='gpu_hist', verbosity=1)
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 73196, then your tr_loop will be 73196 - 73196%1000 = 730
    # in this for loop we will iterate until 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


#ROC Curve for Set1 :
from sklearn.metrics import roc_curve, auc

Model1= XGBClassifier(n_estimators=60, learning_rate=0.1, reg_alpha=0, reg_lambda=0, boost
Model1.fit(X_tr1_w2v, y_train)

y_train_pred1 = batch_predict(Model1, X_tr1_w2v)
y_test_pred1 = batch_predict(Model1, X_te1_w2v)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred1)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred1)

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)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



```
#Confusion Matrix:

def get_predicted_y_vec_from_threshold(proba, threshold, fpr, tpr):
  optimal_threshold = threshold[np.argmax(tpr * (1-fpr))]

  predicted_y_vector1 = []
  for i in proba:
    if i >= optimal_threshold:
      predicted_y_vector1.append(1)
    else:
      predicted_y_vector1.append(0)

  return predicted_y_vector1

confusion_matrix_s2_train = confusion_matrix(y_train, get_predicted_y_vec_from_threshold(y

confusion_matrix_s2_test = confusion_matrix(y_test, get_predicted_y_vec_from_threshold(y_t

print('confusion_matrix_s2_train ', confusion_matrix_s2_train)
# Heatmap for Confusion Matrix: Train and SET 2
heatmap_confusion_matrix_train_s2 = sns.heatmap(confusion_matrix_s2_train, annot=True, fmt

plt.title('S2 Train Set: Confusion Matrix')
plt.xlabel('Actual X values')
plt.ylabel('Predicted Y Values')
plt.show()

heatmap_confusion_matrix_test_s2 = sns.heatmap(confusion_matrix_s2_test, annot=True, fmt='

plt.title('S2 Test Set: Confusion Matrix')
plt.xlabel('Actual X values')
plt.ylabel('Predicted Y Values')
plt.show()
```

```
confusion_matrix_s2_train  [[ 7094  3989]
 [20720 41393]]
```



Lets Build Pretty Table: Summary

```
#Let's Build PrettyTable:
from prettytable import PrettyTable

x=PrettyTable()
x.field_names=["Vectorizer","Model","Train AUC","Test AUC"]
x.add_row(["Set 1: Tfidf","XGBoost","0.9999","0.6482"])
x.add_row(["Set 2: Tfidf_w2v","XGBoost","0.5850","0.5535"])
print(x)
```

```
+------------------+---------+-----------+----------+
|    Vectorizer    |  Model  | Train AUC | Test AUC |
+------------------+---------+-----------+----------+
|   Set 1: Tfidf   | XGBoost |   0.9999  |  0.6482  |
| Set 2: Tfidf_w2v | XGBoost |   0.5850  |  0.5535  |
+------------------+---------+-----------+----------+
```

Conclusion:

- Gradient Boosting for classification:

GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage n_classes_ regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. Binary classification is a special case where only a single regression tree is induced.

- In this, donors choose dataset XGBoost classifier with Tfidf vectorizer works very well rather than Tfidf_w2v.

✓ 0s completed at 9:52 PM