Assignment 9: GBDT

Libraries

In [ ]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from wordcloud import WordCloud, STOPWORDS

import pickle
import numpy as np
import pandas as pd
from tqdm import tqdm
import seaborn as sns
import xgboost as xgb
import matplotlib.pyplot as plt
from  scipy.sparse  import hstack
from  scipy.sparse  import vstack

from sklearn.metrics import roc_curve, auc
from sklearn.tree import DecisionTreeClassifier as DTC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix as c_m
from sklearn.model_selection import RandomizedSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import tensorflow as tf
```

Custom Functions

In [ ]:

```
# code to load glove vectors
import pickle
with open('/content/drive/My Drive/Colab Notebooks/08. NB on Donor Choose/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())

# TFIDF- Word2Vec
def TFIDF_w2v(data, IDF):
    with tf.device('/device:GPU:0'):
        tfidf_w2v_vectors= []

        for sentence in tqdm(data):
            vector = np.zeros(300)
            tf_idf_weight =0;

            for word in sentence.split():
                if (word in glove_words) and (word in IDF.keys()):
                    tf_idf = IDF[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf va
                    vector += (model[word] * 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 np.array(tfidf_w2v_vectors)
```

In [ ]:

```
def sent_score(data):
    sid, lst = SentimentIntensityAnalyzer(), []
    for essay in data:
        ss = sid.polarity_scores(essay)
        sent = list(ss.values())
        lst.append(sent)
    return lst
```

In [ ]:

```
def Vectorise(feature, Data, Y_train):
    with tf.device('/device:GPU:0'):
        ful_lst = X_train[feature].values
        lst = np.unique(ful_lst)
        Pos = {} ; Encod = {}
        for ele in lst:
            Pos[ele]=[0,0]
        for i in range(len(Data)):
            for ele in lst:
                if ele == ful_lst[i] :
                    Pos[ele][1] +=1
                    if Y_train[i] ==1: Pos[ele][0] +=1
        for ele in lst:
            Encod[ele] = Pos[ele][0]/Pos[ele][1]
        return Encod, feature

def Fit_resp(Data, resp):
    feature_ = []
    for ele in Data[resp[1]].values:
        try: feature_.append([resp[0][ele], 1-resp[0][ele]])
        except: feature_.append([1/2, 1/2])
    return feature_
```

In [ ]:

```
def Concatenate(X_train_essay, X_test_essay):
    X_tr = hstack((X_train_essay,Tr_sent, X_train_state_rp, X_train_tchr_rp, X_train_pgc_rp,
                   X_train_clc_rp, X_train_clsc_rp, X_tr_Num)).tocsr()
    X_te = hstack((X_test_essay, Te_sent, X_test_state_rp, X_test_tchr_rp, X_test_pgc_rp,
                   X_test_clc_rp, X_test_clsc_rp, X_tst_Num)).tocsr()

    print("Final Data matrix")
    print(X_tr.shape, Y_train.shape)
    print(X_te.shape, Y_test.shape)
    return X_tr, X_te
```

In [ ]:

```
def Results_Random_Search(X_tr, Y_train):
    with tf.device('/device:GPU:0'):
        XGB = xgb.XGBClassifier()
        Hyp_param = {'max_depth': [1, 5, 10], 'learning_rate': [10**x for x in range(-2, 1)]}
        fit_params = {'eval_metric': 'auc', 'early_stopping_rounds': 10, 'eval_set': [(X_tr, Y_train)]}
        num_round = 10

        clf = RandomizedSearchCV(XGB, Hyp_param, n_iter=20, n_jobs=-1, cv=2, verbose=2,
                                 scoring='roc_auc',return_train_score=True, refit=False, random_state=42)
        clf.fit(X_tr, Y_train)

        return clf
```

In [ ]:

```python
import seaborn as sns

def plot_Hyperparam_vs_AUC(clf):
    Hyp_param = {'max_depth': [1, 5, 10], 'learning_rate': [10**x for x in range(-2, 1)]}

    cv_results = pd.DataFrame.from_dict(clf.cv_results_)
    mxid = np.argmax(cv_results["mean_test_score"])
    Opt_depth =cv_results["param_max_depth"][mxid]
    Opt_learning_rate = cv_results["param_learning_rate"][mxid]
    print('Opt_depth : ', Opt_depth,'and   Opt_learning_rate : ', Opt_learning_rate)

    train_auc= cv_results['mean_train_score'].astype(float)
    train_auc_std= cv_results['std_train_score'].astype(float)
    cv_auc = cv_results['mean_test_score'].astype(float)
    cv_auc_std= cv_results['std_test_score'].astype(float)
    max_depth =  cv_results["param_max_depth"].astype(float)
    learning_rate =  cv_results["param_learning_rate"].astype(float)

    sns.heatmap(np.array(cv_auc).reshape(3,3), cmap='Blues', xticklabels= Hyp_param['max_depth'],
                yticklabels= Hyp_param['learning_rate'], annot=True)
    plt.xlabel('max_depth') ; plt.ylabel('learning_rate')
    plt.title('AUC wrt Hyp_param')

    return Opt_depth, Opt_learning_rate
```

In [ ]:

```python
def plot_ROC_curve(X_tr,X_te,Y_train,Y_test, Opt_depth, Opt_learning_rate):
    with tf.device('/device:GPU:0'):
        clf = XGB = xgb.XGBClassifier(Opt_depth, Opt_learning_rate)
        clf.fit(X_tr, Y_train)

        Y_train_pred = clf.predict_proba(X_tr)[:,1]
        Y_test_pred = clf.predict_proba(X_te)[:,1]

        tr_fpr, tr_tpr, tr_thr = roc_curve(Y_train, Y_train_pred)
        tst_fpr, tst_tpr, te_thr = roc_curve(Y_test, Y_test_pred)

        plt.plot(tr_fpr, tr_tpr, label="Train AUC ="+str(auc(tr_fpr, tr_tpr)))
        plt.plot(tst_fpr, tst_tpr, label="Test AUC ="+str(auc(tst_fpr, tst_tpr)))
        plt.grid() ; plt.legend()
        plt.xlabel("fpr") ;plt.ylabel("tpr") ;plt.title("ROC Curve")
        plt.show()
        return Y_train_pred, Y_test_pred, tr_fpr, tr_tpr, tr_thr
```

In [ ]:

```python
def find_best_threshold(threshould, fpr, tpr):     # finds the Optimum t
    t = threshould[np.argmax((tpr*(1-fpr)))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

def Predict(proba, thr, Y):  # Predicts using the Optimum t
    pred = []
    FP_ind = []
    for i in range(len(proba)):
        if proba[i]>=thr: pred.append(1)
        else:             pred.append(0)
        if pred[i] == 0 and Y[i] ==1:
            FP_ind.append(i)
    return pred, FP_ind
```

In [ ]:

```python
def plot_conf_mat(Train, Test):
    plt.figure(figsize = (16,8))
    plt.subplot(221) ;  plt.title('Train : Confusion Matrix ')
    sns.heatmap(Train, annot=True, fmt="d", cmap="YlGnBu", square=True,xticklabels=['Negative', 'Positive'
    plt.xlabel("Actual") ;plt.ylabel('Predcted')
    plt.subplot(222) ; plt.title('Test : Confusion Matrix ') ;
    sns.heatmap(Test, annot=True, fmt="d", cmap="YlGnBu", square=True, xticklabels=['Negative', 'Positive'
    plt.xlabel("Actual") ;plt.ylabel('Predcted')
```

1. GBDT (xgboost/lightgbm)

1.1 Loading Data

In [ ]:

```
data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/11. DT on Donor Choose/preprocessed_data.csv')
X = data.drop(['project_is_approved'], axis=1)
Y = data['project_is_approved'].values
```

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

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, stratify=Y) # Train 70% , test 30
```

1.3 Make Data Model Ready: encoding eassay, and project_title

In [ ]:

```
# extracting unique words from the Train dataset.
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)
X_train_essay_tfidf = vectorizer.fit_transform(X_train['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values) # we use the fitted CountVectorizer to c

print(X_train_essay_tfidf.shape, X_test_essay_tfidf.shape)
```

```
(76473, 10000) (32775, 10000)
```

In [ ]:

```
nltk.download('vader_lexicon')  # use if error
Tr_sent = sent_score(X_train['essay'])
Te_sent = sent_score(X_test['essay'])
```

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

1.4 Make Data Model Ready: encoding numerical, categorical features

Encoding Categorical by Responce Coding

In [ ]:

```
sst = Vectorise('school_state', X_train, Y_train)
X_train_state_rp = Fit_resp(X_train, sst)
X_test_state_rp = Fit_resp(X_test, sst)

tchr = Vectorise('teacher_prefix', X_train, Y_train)
X_train_tchr_rp = Fit_resp(X_train, tchr)
X_test_tchr_rp = Fit_resp(X_test, tchr)

pgc = Vectorise('project_grade_category', X_train, Y_train)
X_train_pgc_rp = Fit_resp(X_train, pgc)
X_test_pgc_rp = Fit_resp(X_test, pgc)

clc = Vectorise('clean_categories', X_train, Y_train)
X_train_clc_rp = Fit_resp(X_train, clc)
X_test_clc_rp = Fit_resp(X_test, clc)

clsc = Vectorise('clean_subcategories', X_train, Y_train)
X_train_clsc_rp = Fit_resp(X_train, clsc)
X_test_clsc_rp = Fit_resp(X_test, clsc)
```

Encoding numerical

```
In [ ]: # Price: Normalising the vectros between 0 and 1
        Mx, Mn = X_train['price'].max(), X_train['price'].min()
        X_train_price_norm = ((X_train['price'] - Mn)/(Mx - Mn))
        X_test_price_norm = ((X_test['price'] - Mn)/(Mx - Mn))

        # No of projects posted by teachers: Normalising the vectros between 0 and 1
        mx, mn = X_train['teacher_number_of_previously_posted_projects'].max(), X_train['teacher_number_of_previou
        X_train_tnpp_norm = ((X_train['teacher_number_of_previously_posted_projects'] - mn)/(mx - mn))
        X_test_tnpp_norm = ((X_test['teacher_number_of_previously_posted_projects'] - mn)/(mx - mn))

        # Combining the Numerical features
        X_tr_Num = np.array([X_train_tnpp_norm, X_train_price_norm]).T
        X_tst_Num = np.array([X_test_tnpp_norm, X_test_price_norm]).T

        print(X_train_price_norm.shape, X_test_price_norm.shape)
        print(X_train_tnpp_norm.shape, X_test_tnpp_norm.shape)
        print(X_tr_Num.shape, X_tst_Num.shape)
```

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

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

```
In [ ]: X_tr, X_te = Concatenate(X_train_essay_tfidf, X_test_essay_tfidf)
```

```
Final Data matrix
(76473, 10016) (76473,)
(32775, 10016) (32775,)
```

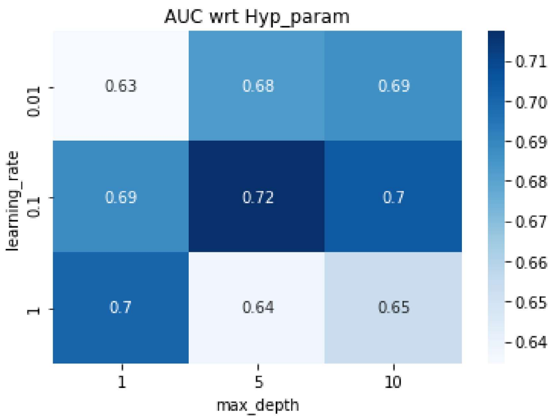
```
In [ ]: clf = Results_Random_Search(X_tr, Y_train)
```

```
Fitting 2 folds for each of 9 candidates, totalling 18 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed: 40.3min finished
```

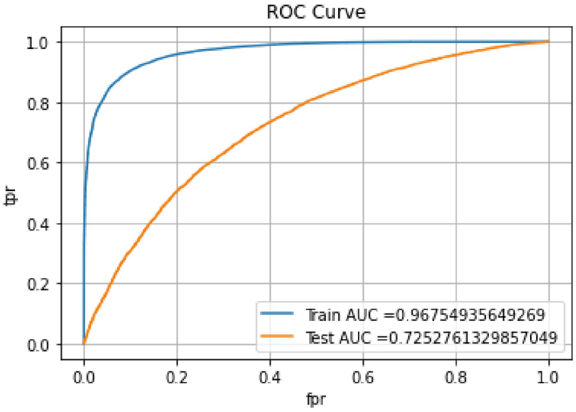
```
In [ ]: Opt_depth, Opt_learning_rate = plot_Hyperparam_vs_AUC(clf)
```

```
Opt_depth :  5 and   Opt_learning_rate :  0.1
```



Performance Testing

```
In [ ]: R = plot_ROC_curve(X_tr,X_te,Y_train,Y_test, Opt_depth, Opt_learning_rate)
```



Confusion Matrix

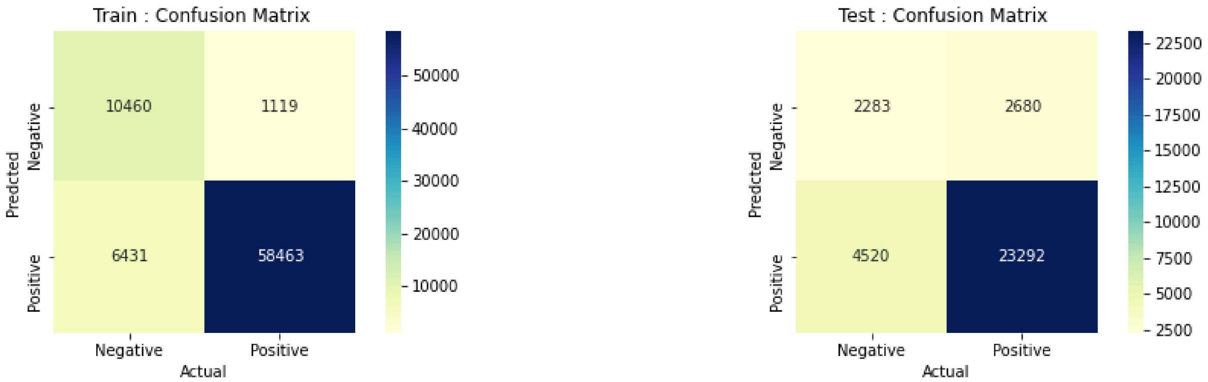
```
In [ ]: Y_train_pred, Y_test_pred, tr_fpr, tr_tpr, tr_thr  = R[0], R[1], R[2], R[3], R[4]
        t = find_best_threshold(tr_thr, tr_fpr, tr_tpr)

        Y_tr_pre, tr_FP_ind = Predict(Y_train_pred, t, Y_train) # Returns Y_ored and indices of False Positive dat
        Y_te_pre, te_FP_ind = Predict(Y_test_pred, t, Y_test)   # Returns Y_ored and indices of False Positive dat

        cfm_train = c_m(Y_train, Y_tr_pre)
        cfm_tst = c_m(Y_test, Y_te_pre)

        plot_conf_mat(cfm_train, cfm_tst)
```

```
the maximum value of tpr*(1-fpr) 0.8138365367082393 for threshold 0.796
```



2. TFIDF weighted w2v

```
In [ ]: from scipy import sparse as sp
        # extracting unique words from the Train dataset.
        TFIDF = TfidfVectorizer(min_df=10)
        TFIDF.fit(X_train['essay'].values)
        IDF = dict(zip(TFIDF.get_feature_names(), list(TFIDF.idf_)))

        tfidf_w2v_vectors_tr = sp.csr_matrix(TFIDF_w2v(X_train['essay'], IDF))
        tfidf_w2v_vectors_te = sp.csr_matrix(TFIDF_w2v(X_test['essay'], IDF))

        print(tfidf_w2v_vectors_tr.shape, tfidf_w2v_vectors_te.shape)
```

```
100%|██████████| 76473/76473 [02:42<00:00, 471.17it/s]
100%|██████████| 32775/32775 [01:08<00:00, 475.54it/s]

(76473, 300) (32775, 300)
```

Applying Model

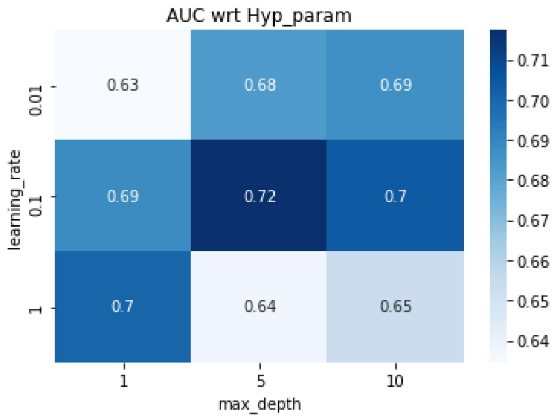
```
In [ ]: print(tfidf_w2v_vectors_tr.shape, tfidf_w2v_vectors_te.shape)
        print(type(tfidf_w2v_vectors_tr))
        X_tr_2, X_te_2 = Concatenate(tfidf_w2v_vectors_tr, tfidf_w2v_vectors_te)
```

```
(76473, 300) (32775, 300)
<class 'scipy.sparse.csr.csr_matrix'>
Final Data matrix
(76473, 316) (76473,)
(32775, 316) (32775,)
```

```
In [ ]: clf = Results_Random_Search(X_tr_2, Y_train)
```

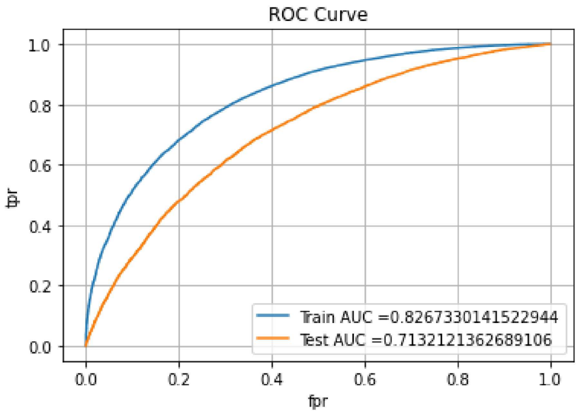
```
In [ ]: Opt_depth, Opt_learning_rate = plot_Hyperparam_vs_AUC(clf)
```

```
Opt_depth :  5 and   Opt_learning_rate :  0.1
```



Performance Testing

```
In [ ]: R = plot_ROC_curve(X_tr_2,X_te_2,Y_train,Y_test, Opt_depth, Opt_learning_rate)
```



Confusion Matrix

In [ ]:

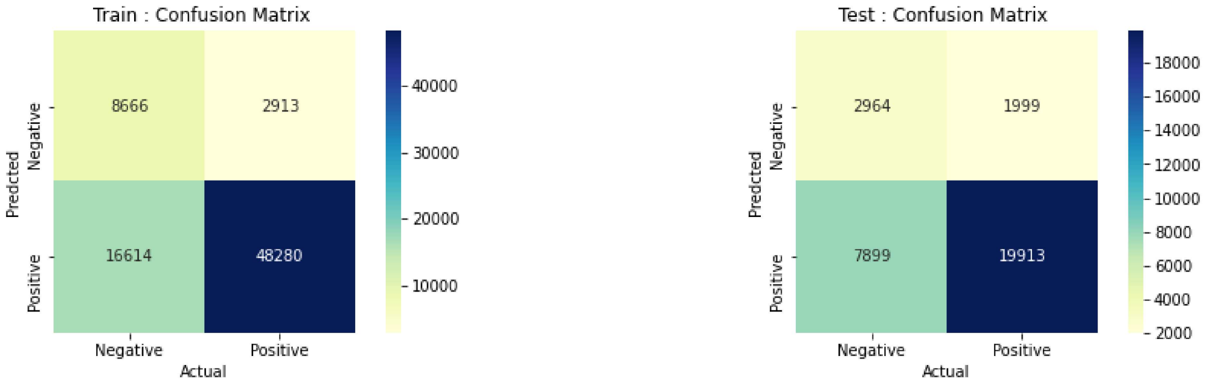
```
Y_train_pred, Y_test_pred, tr_fpr, tr_tpr, tr_thr  = R[0], R[1], R[2], R[3], R[4]
t = find_best_threshold(tr_thr, tr_fpr, tr_tpr)

Y_tr_pre, tr_FP_ind = Predict(Y_train_pred, t, Y_train) # Returns Y_ored and indices of False Positive dat
Y_te_pre, te_FP_ind = Predict(Y_test_pred, t, Y_test)   # Returns Y_ored and indices of False Positive dat

cfm_train = c_m(Y_train, Y_tr_pre)
cfm_tst = c_m(Y_test, Y_te_pre)

plot_conf_mat(cfm_train, cfm_tst)
```

```
the maximum value of tpr*(1-fpr) 0.5568142583636754 for threshold 0.833
```



3. Summary

```
*   Total Datapoints Used              : 109248
*   size of Vector Studied in TFIDF-W2v : 300
*   Features Studied in TFIDF          : 10000


Vectorizer   |     Model       | Hyp (Opt_depth) | Hyp (Learning rate ) |  AUC  |
---------------------------------------------------------------------------------
Tfidf        |  GBDT(XgBoost)  |        5        |        0.1           |  0.73  |
Tfidf-W2v    |  GBDT(XgBoost)  |        5        |        0.1           |  0.71  |
```