Assignment 9: GBDT

1. GBDT (xgboost/lightgbm)

In [1]:

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

import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/

import pickle
from tqdm import tqdm
import os

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

1.1 Loading Data

In [2]:

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

In [3]:

```
data.head(3)
```

Out[3]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	c
0	ca	mrs	grades_prek_2	53	1	
1	ut	ms	grades_3_5	4	1	
2	ca	mrs	grades_prek_2	10	1	lit

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	c

In [4]:

```
#storing the project_is_approved(class label) seperately and removing it from the dataset
y = data['project_is_approved'].values
x = data.drop(columns='project_is_approved', axis=1)
x.head(3)
```

Out[4]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clea
0	ca	mrs	grades_prek_2	53	math_science	h
1	ut	ms	grades_3_5	4	specialneeds	
2	ca	mrs	grades_prek_2	10	literacy_language	

In [5]:

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

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

Out[5]:

```
True
```

In [6]:

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

import numpy as np
def sentiment_score(X,feature):
  neg=[]
  neu=[]
  pos=[]
  compound=[]
  sid = SentimentIntensityAnalyzer()
  for i in range(len(X)):
    for_sentiment = X[feature].iloc[i]
    ss = sid.polarity_scores(for_sentiment)
    neg.append(ss['neg'])
    neu.append(ss['neu'])
    pos.append(ss['pos'])
    compound.append(ss['compound'])
  return np.asarray(neg).reshape(-1,1),np.asarray(neu).reshape(-1,1),np.asarray(pos).res
hape(-1,1),np.asarray(compound).reshape(-1,1)
```

In [7]:

```
#adding the 4 new features to the dataset x
```

```
negative,neutral,postive,compound = sentiment_score(x,"essay")
x["sen_neg"]=negative
x["sen_pos"]=postive
x["sen_neu"]=neutral
x["sen_comp"]=compound
```

In [8]:

```
x.head(3)
```

Out[8]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clea
0	ca	mrs	grades_prek_2	53	math_science	h
1	ut	ms	grades_3_5	4	specialneeds	
2	ca	mrs	grades_prek_2	10	literacy_language	

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

In [9]:

```
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)
```

In [10]:

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

```
(33500, 12) (33500,)
(16500, 12) (16500,)
```

1.3 Make Data Model Ready

Encoding the Text Feature using TFIDF : Essay

In [11]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)

train_essay_tfidf = vectorizer.fit_transform(X_train['essay'].values)
test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("Shapes after vectorization")
print(train_essay_tfidf.shape,y_train.shape)
print(test_essay_tfidf.shape,y_test.shape)
```

```
Shapes after vectorization
(33500, 5000) (33500,)
(16500, 5000) (16500,)
```

(16500, 5000) (16500,)

Encoding the Text Feature using TFIDF W2V : Essay

In [12]:

```
#please use below code to load glove vectors
with open('/content/sample_data/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [13]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
#print(dictionary)
#print(tfidf_words)
```

In [15]:

```
def compute_tfidf_w2v(data1):
    # compute average word2vec for each review.
    tfidf_w2v_vectors = [] # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(data1): # 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((s
entence.count(word)/len(sentence.split())))
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) #
getting the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)
    return tfidf_w2v_vectors
```

In [16]:

```
X_train_tfidf_w2v = compute_tfidf_w2v(X_train['essay'].values)
X_test_tfidf_w2v = compute_tfidf_w2v(X_test['essay'].values)
```

```
100%|██████████| 33500/33500 [01:15<00:00, 443.21it/s]
100%|██████████| 16500/16500 [00:36<00:00, 453.71it/s]
```

In [17]:

```
train_tfidf_w2v = np.array(X_train_tfidf_w2v)
test_tfidf_w2v = np.array(X_test_tfidf_w2v)
print("Shape after vectorizzation")
print(train_tfidf_w2v.shape,y_train.shape)
print(test_tfidf_w2v.shape,y_test.shape)
```

```
Shape after vectorizzation
(33500, 300) (33500,)
(16500, 300) (16500,)
```

In [18]:

```
from scipy.sparse import coo_matrix
#code for converting a dense tfidf_w2v matrix into sparse matrix
train_tfidf_w2v = coo_matrix(train_tfidf_w2v)
test_tfidf_w2v = coo_matrix(test_tfidf_w2v)
print(train_tfidf_w2v.shape,y_train.shape)
```

```
print(test_tfidf_w2v.shape,y_test.shape)
```

```
(33500, 300) (33500,)
(16500, 300) (16500,)
```

Encoding Numerical Features : price

In [19]:

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

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

print("Shape after vectorizations")
print(train_price_norm.shape, y_train.shape)
print(test_price_norm.shape, y_test.shape)
```

```
Shape after vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

Encoding Numerical Features : teacher_number_of_previously_posted_projects

In [20]:

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

train_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X_train['t
eacher_number_of_previously_posted_projects'].values.reshape(-1,1))
test_teacher_number_of_previously_posted_projects_norm = normalizer.transform(X_test['tea
cher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("Shape after vectorizations")
print(train_teacher_number_of_previously_posted_projects_norm.shape, y_train.shape)
print(test_teacher_number_of_previously_posted_projects_norm.shape, y_test.shape)
```

```
Shape after vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

Encoding Categorical Features : project_grade_category using Response Coding

In [21]:

```
#https://stackoverflow.com/questions/66122577/response-coding-for-categorical-data
def compute_response_coding(xtrain,ytrain,feature):
    dict_ = dict()
    unique_cat_labels = xtrain[feature].unique()

    for i in tqdm(range(len(unique_cat_labels))):
        total_count = xtrain.loc[:, feature][(xtrain[feature] == unique_cat_labels[i])].
count()
        p_0 = xtrain.loc[:, feature][((xtrain[feature] == unique_cat_labels[i]) & (ytrai
n==0))].count()
        p_1 = xtrain.loc[:, feature][((xtrain[feature] == unique_cat_labels[i]) & (ytrai
n==1))].count()

        dict_[unique_cat_labels[i]] = [p_1/total_count, p_0/total_count]

    return dict_
```

In [22]:

```
column_names_list = list(x.columns)
print(column_names_list)
project_grade_category = column_names_list[2]
dict_ = compute_response_coding(X_train,y_train,project_grade_category)
print(dict_)
```

```
100%|██████████| 4/4 [00:00<00:00, 97.85it/s]
```

```
['school_state', 'teacher_prefix', 'project_grade_category', 'teacher_number_of_previously_posted_projects', 'clean_categories', 'clean_subcategories', 'essay', 'price', 'sen_neg', 'sen_pos', 'sen_neu', 'sen_comp']
{'grades_3_5': [0.8450912250217203, 0.15490877497827976], 'grades_6_8': [0.8289003572846367, 0.17109964271536324], 'grades_prek_2': [0.8401882693700218, 0.15981173062997828], 'grades_9_12': [0.8367281985996181, 0.16327180140038192]}
```

In [23]:

```
'''encoding X_train data
train_lst = []
lst =  list(X_train['project_grade_category'])
for val in lst:
  #checking if category is present as key the dictionary created
   if val in dict_.keys():
       #appending the dict value to the list based on the dict key
       train_lst.append(dict_.get(val))
#converting list to an array
train_project_grade_category_res = np.array(train_lst)'''
```

Out[23]:

```
"encoding X_train data\ntrain_lst = []\nlst =  list(X_train['project_grade_category'])\nfor val in lst:\n  #checking if category is present as key the dictionary created\n   if val in dict_.keys():\n       #appending the dict value to the list based on the dict key\n       train_lst.append(dict_.get(val))\n#converting list to an array\ntrain_project_grade_category_res = np.array(train_lst)"
```

In [ ]:

```
'''encoding X_test data
default = [0.5,0.5]
test_lst = []
lst = list(X_test['project_grade_category'])
for val in lst:
    if val in dict_.keys():
        test_lst.append(dict_.get(val))
    else:
        test_lst.append(default)
test_project_grade_category_res = np.array(test_lst)'''
```

Out[ ]:

```
"encoding X_test data\ndefault = [0.5,0.5]\ntest_lst = []\nlst = list(X_test['project_grade_category'])\nfor val in lst:\n    if val in dict_.keys():\n        test_lst.append(dict_.get(val))\n    else:\n        test_lst.append(default)\ntest_project_grade_category_res = np.array(test_lst)"
```

In [24]:

```
def compute_vec(X,feature,d):
    default_val = [0.5,0.5]
    vec_lst = []
    lst =  list(X[feature])
    for val in lst:
    #checking if category is present as key the dictionary created
        if val in d.keys():
           #appending the dict value to the list based on the dict key
           vec_lst.append(d.get(val))
        else:
           #appending the default value[0.5,0.5] if the key is not present in the dict
            vec_lst.append(default_val)
```

```
    #converting list to an array
    vec_arr = np.array(vec_lst)
    return vec_arr
```

In [25]:

```
#encoding X_train data
train_project_grade_category_res = compute_vec(X_train,project_grade_category,dict_)
#encoding X_test data
test_project_grade_category_res = compute_vec(X_test,project_grade_category,dict_)
```

In [ ]:

```
#print(dict_)
#print('*'*50)
#print(test_project_grade_category_res[0:10,:])
#print('*'*50)
#print(X_test['project_grade_category'].head(10))
#print(test_project_grade_category_res.shape)
```

In [26]:

```
print("Shape after vectorization")
print(train_project_grade_category_res.shape,y_train.shape)
print(test_project_grade_category_res.shape,y_test.shape)
```

```
Shape after vectorization
(33500, 2) (33500,)
(16500, 2) (16500,)
```

Encoding Categorical Features : teacher_prefix using Response Coding

In [27]:

```
column_names_list = list(x.columns)
#print(column_names_list)
teacher_prefix = column_names_list[1]
dict_ = compute_response_coding(X_train,y_train,teacher_prefix)
print(dict_)

#encoding X_train data
train_teacher_prefix_res = compute_vec(X_train,teacher_prefix,dict_)
#encoding X_test data
test_teacher_prefix_res = compute_vec(X_test,teacher_prefix,dict_)

print("Shape after vectorization")
print(train_teacher_prefix_res.shape,y_train.shape)
print(test_teacher_prefix_res.shape,y_test.shape)

#print(test_teacher_prefix_res[0:10,:])
#print('*'*50)
#print(X_test['teacher_prefix'].head(10))
```

```
100%|██████████| 5/5 [00:00<00:00, 102.90it/s]
```

```
{'ms': [0.8334600118233257, 0.16653998817667426], 'mrs': [0.8474472468125382, 0.152552753
18746173], 'mr': [0.8336062888961677, 0.1663937111038323], 'teacher': [0.7753510140405616
, 0.22464898595943839], 'dr': [0.75, 0.25]}
Shape after vectorization
(33500, 2) (33500,)
(16500, 2) (16500,)
```

Encoding Categorical Features : school_state using Response Coding

In [28]:

```
column_names_list = list(x.columns)
```

```python
#print(column_names_list)
school_state = column_names_list[0]
dict_ = compute_response_coding(X_train,y_train,school_state)
print(dict_)

#encoding X_train data
train_school_state_res = compute_vec(X_train,school_state,dict_)
#encoding X_test data
test_school_state_res = compute_vec(X_test,school_state,dict_)

print("Shape after vectorization")
print(train_school_state_res.shape,y_train.shape)
print(test_school_state_res.shape,y_test.shape)

#print(test_school_state_res[0:10,:])
#print('*'*50)
#print(X_test['school_state'].head(10))
```

```
100%|██████████| 51/51 [00:00<00:00, 120.01it/s]
```

```
{'pa': [0.841831425598335, 0.15816857440166493], 'mi': [0.8177874186550976, 0.18221258134
49024], 'hi': [0.875, 0.125], 'ms': [0.8144578313253013, 0.1855421686746988], 'de': [0.86
92307692307693, 0.13076923076923078], 'ga': [0.8303078137332282, 0.1696921862667719], 'ca
': [0.84751269035533, 0.15248730964467005], 'fl': [0.8314328210213188, 0.1685671789786812
], 'or': [0.8198614318706697, 0.18013856812933027], 'il': [0.855084067253803, 0.144915932
74619696], 'sc': [0.8479582971329279, 0.1520417028670721], 'ny': [0.8657407407407407, 0.1
342592592592592 7], 'ok': [0.8259109311740891, 0.17408906882591094], 'ut': [0.834365325077
3994, 0.16563467492260062], 'nj': [0.8212996389891697, 0.17870036101083034], 'la': [0.809
8958333333334, 0.19010416666666666], 'tx': [0.7969798657718121, 0.20302013422818793], 'wa
': [0.8810198300283286, 0.11898016997167139], 'mn': [0.8657142857142858, 0.13428571428571
429], 'nv': [0.8160377358490566, 0.18396226415094338], 'in': [0.8578431372549019, 0.14215
686274509803], 'oh': [0.8755020080321285, 0.12449799196787148], 'ky': [0.8608923884514436
, 0.13910761154855644], 'va': [0.8464052287581699, 0.15359477124183007], 'dc': [0.8106508
875739645, 0.1893491124260355], 'nc': [0.8358570563294972, 0.16414294367050272], 'id': [0
.8382352941176471, 0.16176470588235295], 'vt': [0.8076923076923077, 0.19230769230769232],
'ct': [0.8452914798206278, 0.1547085201793722], 'al': [0.8526315789473684, 0.147368421052
63157], 'mo': [0.8657465495608532, 0.1342534504391468], 'wi': [0.848816029143898, 0.15118
3970856102], 'ar': [0.8136094674556213, 0.1863905325443787], 'az': [0.8295454545454546, 0
.17045454545454544], 'nm': [0.8136645962732919, 0.18633540372670807], 'tn': [0.8486486486
486486, 0.15135135135135136], 'ma': [0.841726618705036, 0.15827338129496402], 'sd': [0.83
69565217391305, 0.16304347826086957], 'mt': [0.8032786885245902, 0.19672131147540983], 'a
k': [0.8529411764705882, 0.14705882352941177], 'ia': [0.8497652582159625, 0.1502347417840
3756], 'co': [0.8349514563106796, 0.1650485436893204], 'md': [0.8260869565217391, 0.17391
304347826086], 'me': [0.875968992248062, 0.12403100775193798], 'ks': [0.8352272727272727,
0.16477272727272727], 'wv': [0.8076923076923077, 0.19230769230769232], 'ri': [0.887323943
6619719, 0.11267605633802817], 'nd': [0.9473684210526315, 0.05263157894736842], 'ne': [0.
8222222222222222, 0.17777777777777778], 'nh': [0.8913043478260869, 0.10869565217391304],
'wy': [0.9230769230769231, 0.07692307692307693]}
Shape after vectorization
(33500, 2) (33500,)
(16500, 2) (16500,)
```

Encoding Categorical Features : clean_categories using Response Coding

In [29]:

```python
column_names_list = list(x.columns)
#print(column_names_list)
clean_categories = column_names_list[4]
dict_ = compute_response_coding(X_train,y_train,clean_categories)
print(dict_)

#encoding X_train data
train_clean_categories_res = compute_vec(X_train,clean_categories,dict_)
#encoding X_test data
test_clean_categories_res = compute_vec(X_test,clean_categories,dict_)

print("Shape after vectorization")
```

```
print(train_clean_categories_res.shape,y_train.shape)
print(test_clean_categories_res.shape,y_test.shape)

#print(test_clean_categories_res[0:10,:])
#print('*'*50)
#print(X_test['clean_categories'].head(10))
```

```
100%|██████████| 45/45 [00:00<00:00, 130.76it/s]
```

```
{'literacy_language math_science': [0.8551276309896999, 0.14487236901030004], 'math_scien
ce': [0.8016064257028113, 0.19839357429718876], 'health_sports': [0.8554721977052074, 0.1
445278022947926], 'history_civics specialneeds': [0.7547169811320755, 0.24528301886792453
], 'math_science music_arts': [0.8323529411764706, 0.16764705882352941], 'literacy_languag
e specialneeds': [0.8574007220216606, 0.14259927797833935], 'history_civics': [0.84427767
35459663, 0.15572232645403378], 'literacy_language music_arts': [0.8076923076923077, 0.19
230769230769232], 'literacy_language': [0.8526968032671455, 0.14730319673285452], 'math_s
cience history_civics': [0.8362831858407079, 0.16371681415929204], 'appliedlearning music
_arts': [0.8258706467661692, 0.17412935323383086], 'appliedlearning': [0.8070175438596491
, 0.19298245614035087], 'music_arts': [0.8639218422889045, 0.1360781577110956], 'specialn
eeds': [0.7964757709251101, 0.20352422907488987], 'math_science specialneeds': [0.8405511
811023622, 0.1594488188976378], 'appliedlearning literacy_language': [0.8658008658008658,
0.1341991341991342], 'literacy_language appliedlearning': [0.8742857142857143, 0.12571428
571428572], 'history_civics literacy_language': [0.8932714617169374, 0.10672853828306264]
, 'appliedlearning specialneeds': [0.794392523364486, 0.205607476635514], 'health_sports
specialneeds': [0.8718487394957983, 0.12815126050420167], 'math_science health_sports': [
0.7806451612903226, 0.21935483870967742], 'appliedlearning health_sports': [0.83482142857
14286, 0.16517857142857142], 'math_science literacy_language': [0.8480845442536328, 0.151
91545574636725], 'health_sports literacy_language': [0.8676470588235294, 0.13235294117647
06], 'appliedlearning math_science': [0.8032786885245902, 0.19672131147540983], 'appliedl
earning history_civics': [0.8409090909090909, 0.1590909090909091], 'math_science appliedl
earning': [0.8078175895765473, 0.19218241042345277], 'health_sports warmth care_hunger':
[1.0, 0.0], 'literacy_language history_civics': [0.8522012578616353, 0.14779874213836477]
, 'health_sports math_science': [0.8317757009345794, 0.16822429906542055], 'health_sports
music_arts': [0.7931034482758621, 0.20689655172413793], 'history_civics math_science': [0
.8076923076923077, 0.19230769230769232], 'specialneeds music_arts': [0.8028169014084507,
0.19718309859154928], 'health_sports history_civics': [0.8095238095238095, 0.190476190476
19047], 'history_civics music_arts': [0.8924731182795699, 0.10752688172043011], 'health_s
ports appliedlearning': [0.7910447761194029, 0.208955223880597], 'music_arts specialneeds
': [0.8918918918918919, 0.10810810810810811], 'history_civics appliedlearning': [0.857142
8571428571, 0.14285714285714285], 'literacy_language health_sports': [0.7241379310344828,
0.27586206896551724], 'music_arts appliedlearning': [0.3333333333333333, 0.66666666666666
66], 'music_arts health_sports': [0.625, 0.375], 'specialneeds health_sports': [0.6, 0.4]
, 'history_civics health_sports': [1.0, 0.0], 'music_arts history_civics': [1.0, 0.0], 'h
istory_civics warmth care_hunger': [0.0, 1.0]}
Shape after vectorization
(33500, 2) (33500,)
(16500, 2) (16500,)
```

Encoding Categorical Features : clean_subcategories using Response Coding

In [30]:

```
column_names_list = list(x.columns)
#print(column_names_list)
clean_subcategories = column_names_list[5]
dict_ = compute_response_coding(X_train,y_train,clean_subcategories)
print(dict_)

#encoding X_train data
train_clean_subcategories_res = compute_vec(X_train,clean_subcategories,dict_)
#encoding X_test data
test_clean_subcategories_res = compute_vec(X_test,clean_subcategories,dict_)

print("Shape after vectorization")
print(train_clean_subcategories_res.shape,y_train.shape)
print(test_clean_subcategories_res.shape,y_test.shape)

#print(test_clean_subcategories_res[0:10,:])
```

```
#print('*'*50)
#print(X_test['clean_subcategories'].head(10))
```

```
100%|██████████| 339/339 [00:02<00:00, 135.48it/s]
```


```
{'literature_writing mathematics': [0.8588110403397028, 0.14118895966029724], 'appliedsci
ences mathematics': [0.8295566502463054, 0.17044334975369457], 'environmentalscience heal
th_lifescience': [0.8075601374570447, 0.19243986254295534], 'mathematics': [0.79654510556
62188, 0.2034548944337812], 'gym_fitness health_wellness': [0.8806509945750453, 0.1193490
0542495479], 'health_wellness nutritioneducation': [0.8398169336384439, 0.16018306636155
608], 'history_geography specialneeds': [0.7878787878787878, 0.21212121212121213], 'applie
dsciences performingarts': [0.3333333333333333, 0.6666666666666666], 'literacy specialnee
ds': [0.8740849194729137, 0.1259150805270864], 'history_geography socialsciences': [0.853
2110091743119, 0.14678899082568808], 'literature_writing visualarts': [0.7864583333333334
, 0.21354166666666666], 'esl literacy': [0.8548644338118022, 0.14513556618819776], 'envir
onmentalscience history_geography': [0.875, 0.125], 'literacy': [0.8725882781215872, 0.12
741172187841282], 'extracurricular music': [1.0, 0.0], 'appliedsciences': [0.797752808988
764, 0.20224719101123595], 'earlydevelopment other': [0.7796610169491526, 0.2203389830508
4745], 'literacy mathematics': [0.8525976641159887, 0.14740233588401128], 'music': [0.883
8268792710706, 0.11617312072892938], 'literacy literature_writing': [0.8428899082568807,
0.15711009174311927], 'music performingarts': [0.896551724137931, 0.10344827586206896], '
specialneeds': [0.7964757709251101, 0.20352422907488987], 'gym_fitness teamsports': [0.75
26881720430108, 0.24731182795698925], 'appliedsciences specialneeds': [0.9191919191919192
, 0.08080808080808081], 'college_careerprep literacy': [0.9117647058823529, 0.08823529411
764706], 'esl literature_writing': [0.815668202764977, 0.18433179723502305], 'literacy pa
rentinvolvement': [0.92, 0.08], 'health_wellness': [0.8694412770809579, 0.13055872291904
2], 'literature_writing': [0.8455571227080395, 0.15444287729196052], 'literature_writing
performingarts': [0.8157894736842105, 0.18421052631578946], 'history_geography literature
_writing': [0.8839779005524862, 0.11602209944751381], 'earlydevelopment specialneeds': [0
.775, 0.225], 'health_wellness specialneeds': [0.875609756097561, 0.12439024390243902], '
health_lifescience health_wellness': [0.8484848484848485, 0.15151515151515152], 'earlydev
elopment literacy': [0.8371040723981901, 0.16289592760180996], 'appliedsciences visualart
s': [0.8584070796460177, 0.1415929203539823], 'health_lifescience mathematics': [0.766666
6666666667, 0.23333333333333334], 'earlydevelopment gym_fitness': [0.8571428571428571, 0.
14285714285714285], 'health_lifescience history_geography': [0.8888888888888888, 0.111111
1111111111], 'literature_writing specialneeds': [0.8287292817679558, 0.1712707182320442],
'environmentalscience literacy': [0.8354430379746836, 0.16455696202531644], 'health_welln
ess literacy': [0.8505154639175257, 0.14948453608247422], 'mathematics visualarts': [0.78
18181818181819, 0.21818181818181817], 'visualarts': [0.8421052631578947, 0.15789473684210
525], 'mathematics specialneeds': [0.8192771084337349, 0.18072289156626506], 'health_life
science visualarts': [0.9, 0.1], 'health_lifescience literacy': [0.8879310344827587, 0.11
206896551724138], 'teamsports': [0.8148148148148148, 0.18518518518518517], 'gym_fitness':
[0.853763440860215, 0.14623655913978495], 'appliedsciences health_lifescience': [0.784210
5263157895, 0.21578947368421053], 'earlydevelopment health_wellness': [0.8677685950413223
, 0.1322314049586777], 'environmentalscience mathematics': [0.7401574803149606, 0.2598425
1968503935], 'health_wellness literature_writing': [0.9047619047619048, 0.095238095238095
23], 'communityservice mathematics': [1.0, 0.0], 'appliedsciences literature_writing': [0
.8775510204081632, 0.12244897959183673], 'appliedsciences environmentalscience': [0.77985
07462686567, 0.22014925373134328], 'college_careerprep socialsciences': [0.8, 0.2], 'appl
iedsciences other': [0.8108108108108109, 0.1891891891891892], 'health_wellness teamsports
': [0.8394160583941606, 0.16058394160583941], 'appliedsciences literacy': [0.819277108433
7349, 0.18072289156626506], 'college_careerprep literature_writing': [0.9080459770114943,
0.09195402298850575], 'communityservice other': [0.0, 1.0], 'health_lifescience': [0.8273
092369477911, 0.1726907630522088], 'foreignlanguages': [0.7798165137614679, 0.2201834862
3853212], 'esl music': [0.8333333333333334, 0.16666666666666666], 'environmentalscience':
[0.8217054263565892, 0.17829457364341086], 'civics_government history_geography': [0.8441
558441558441, 0.15584415584415584], 'earlydevelopment': [0.8095238095238095, 0.1904761904
7619047], 'other specialneeds': [0.8522727272727273, 0.14772727272727273], 'earlydevelopm
ent visualarts': [0.7, 0.3], 'environmentalscience literature_writing': [0.78888888888888
89, 0.2111111111111111], 'appliedsciences college_careerprep': [0.7627118644067796, 0.237
28813559322035], 'esl': [0.8174603174603174, 0.18253968253968253], 'history_geography': [
0.8375, 0.1625], 'environmentalscience visualarts': [0.8125, 0.1875], 'literacy visualart
s': [0.7784090909090909, 0.2215909090909091], 'extracurricular history_geography': [1.0,
0.0], 'extracurricular literacy': [0.875, 0.125], 'nutritioneducation': [0.77622377622377
63, 0.22377622377622378], 'appliedsciences earlydevelopment': [0.8421052631578947, 0.1578
9473684210525], 'literacy music': [0.9365079365079365, 0.06349206349206349], 'health_well
ness warmth care_hunger': [1.0, 0.0], 'literacy socialsciences': [0.8717948717948718, 0.1
282051282051282], 'college_careerprep visualarts': [0.90625, 0.09375], 'esl specialneeds'
: [0.8301886792452831, 0.16981132075471697], 'college_careerprep extracurricular': [0.818
1818181818182, 0.18181818181818182], 'gym_fitness mathematics': [0.8571428571428571, 0.14
285714285714285], 'civics_government socialsciences': [0.8484848484848485, 0.151515151515
15152], 'gym_fitness health_lifescience': [1.0, 0.0], 'earlydevelopment literature_writin
```

g': [0.8115942028985508, 0.18840579710144928], 'other visualarts': [0.9333333333333333, 0
.06666666666666667], 'charactereducation visualarts': [0.8823529411764706, 0.117647058823
52941], 'health_wellness music': [0.7647058823529411, 0.23529411764705882], 'environmenta
lscience specialneeds': [0.8421052631578947, 0.15789473684210525], 'mathematics other': [
0.8, 0.2], 'charactereducation health_wellness': [0.7446808510638298, 0.2553191489361702]
, 'foreignlanguages literacy': [0.8115942028985508, 0.18840579710144928], 'earlydevelopme
nt mathematics': [0.7640449438202247, 0.23595505617977527], 'mathematics music': [1.0, 0.
0], 'other': [0.8188976377952756, 0.18110236220472442], 'environmentalscience health_well
ness': [0.7619047619047619, 0.23809523809523808], 'health_lifescience literature_writing'
: [0.8983050847457628, 0.1016949152542373], 'college_careerprep health_lifescience': [0.9
411764705882353, 0.058823529411764705], 'charactereducation environmentalscience': [0.875
, 0.125], 'charactereducation literacy': [0.8854961832061069, 0.11450381679389313], 'extr
acurricular performingarts': [1.0, 0.0], 'socialsciences': [0.8611111111111112, 0.1388888
888888889], 'history_geography mathematics': [0.7948717948717948, 0.20512820512820512], '
esl mathematics': [0.8805970149253731, 0.11940298507462686], 'charactereducation teamspor
ts': [0.8333333333333334, 0.16666666666666666], 'college_careerprep specialneeds': [0.694
4444444444444, 0.3055555555555556], 'earlydevelopment socialsciences': [1.0, 0.0], 'commu
nityservice visualarts': [0.7272727272727273, 0.2727272727272727], 'esl visualarts': [0.9
375, 0.0625], 'financialliteracy': [0.7272727272727273, 0.2727272727272727], 'specialneed
s visualarts': [0.8028169014084507, 0.19718309859154928], 'gym_fitness history_geography'
: [1.0, 0.0], 'literacy other': [0.8771929824561403, 0.12280701754385964], 'charactereduc
ation': [0.8421052631578947, 0.15789473684210525], 'charactereducation communityservice':
[0.8571428571428571, 0.14285714285714285], 'health_wellness mathematics': [0.820224719101
1236, 0.1797752808988764], 'college_careerprep parentinvolvement': [0.8888888888888888, 0
.1111111111111111], 'gym_fitness nutritioneducation': [0.8846153846153846, 0.115384615384
61539], 'gym_fitness specialneeds': [0.8333333333333334, 0.16666666666666666], 'gym_fitne
ss literature_writing': [0.8333333333333334, 0.16666666666666666], 'civics_government': [
0.8, 0.2], 'charactereducation earlydevelopment': [0.7321428571428571, 0.2678571428571428
5], 'history_geography visualarts': [0.9322033898305084, 0.06779661016949153], 'history_g
eography literacy': [0.9019607843137255, 0.09803921568627451], 'esl health_lifescience':
[0.9090909090909091, 0.09090909090909091], 'literature_writing socialsciences': [0.849624
060150376, 0.15037593984962405], 'appliedsciences charactereducation': [0.8, 0.2], 'liter
ature_writing parentinvolvement': [0.9333333333333333, 0.06666666666666667], 'charactered
ucation specialneeds': [0.8431372549019608, 0.1568627450980392], 'charactereducation math
ematics': [0.7837837837837838, 0.21621621621621623], 'financialliteracy specialneeds': [0
.5, 0.5], 'health_wellness other': [0.7777777777777778, 0.2222222222222222], 'foreignlang
uages specialneeds': [0.9, 0.1], 'college_careerprep mathematics': [0.8513513513513513, 0
.14864864864864866], 'environmentalscience nutritioneducation': [0.6, 0.4], 'charactereducation
cation literature_writing': [0.8955223880597015, 0.1044776119402985], 'mathematics social
sciences': [0.7857142857142857, 0.21428571428571427], 'college_careerprep other': [0.6923
076923076923, 0.3076923076923077], 'communityservice': [0.7727272727272727, 0.22727272727
272727], 'esl earlydevelopment': [0.9473684210526315, 0.05263157894736842], 'earlydevelop
ment environmentalscience': [0.8666666666666667, 0.13333333333333333], 'college_careerpre
p': [0.8494623655913979, 0.15053763440860216], 'health_lifescience specialneeds': [0.8205
128205128205, 0.1794871794871795], 'appliedsciences music': [0.8333333333333334, 0.166666
66666666666], 'charactereducation civics_government': [0.8, 0.2], 'communityservice histo
ry_geography': [0.75, 0.25], 'performingarts': [0.8494623655913979, 0.15053763440860216],
'literacy performingarts': [0.875, 0.125], 'charactereducation other': [0.821428571428571
4, 0.17857142857142858], 'civics_government environmentalscience': [0.5714285714285714, 0
.42857142857142855], 'college_careerprep communityservice': [0.75, 0.25], 'esl performing
arts': [0.5, 0.5], 'financialliteracy visualarts': [1.0, 0.0], 'civics_government literat
ure_writing': [0.8648648648648649, 0.13513513513513514], 'civics_government literacy': [0
.8888888888888888, 0.1111111111111111], 'economics mathematics': [1.0, 0.0], 'literature_
writing other': [0.7916666666666666, 0.20833333333333334], 'appliedsciences extracurricul
ar': [0.92, 0.08], 'financialliteracy mathematics': [0.8666666666666667, 0.13333333333333
333], 'extracurricular visualarts': [0.8260869565217391, 0.17391304347826086], 'appliedsc
iences history_geography': [0.76, 0.24], 'extracurricular mathematics': [0.91666666666666
66, 0.08333333333333333], 'music specialneeds': [0.8823529411764706, 0.11764705882352941]
, 'history_geography other': [0.8, 0.2], 'charactereducation college_careerprep': [0.8529
411764705882, 0.14705882352941177], 'charactereducation parentinvolvement': [0.75, 0.25],
'communityservice environmentalscience': [0.9333333333333333, 0.06666666666666667], 'esl
health_wellness': [0.6666666666666666, 0.3333333333333333], 'foreignlanguages literature_
writing': [0.7, 0.3], 'health_lifescience nutritioneducation': [0.7142857142857143, 0.285
7142857142857], 'civics_government economics': [0.8, 0.2], 'charactereducation extracurri
cular': [0.6923076923076923, 0.3076923076923077], 'esl history_geography': [0.90909090909
09091, 0.09090909090909091], 'appliedsciences health_wellness': [0.9285714285714286, 0.07
142857142857142], 'socialsciences visualarts': [0.7222222222222222, 0.2777777777777778],
'music other': [0.3333333333333333, 0.6666666666666666], 'environmentalscience socialscie
nces': [0.8275862068965517, 0.1724137931034483], 'health_wellness visualarts': [0.7647058
823529411, 0.23529411764705882], 'civics_government mathematics': [0.75, 0.25], 'environm
entalscience music': [1.0, 0.0], 'college_careerprep economics': [1.0, 0.0], 'communityse
rvice specialneeds': [0.8571428571428571, 0.14285714285714285], 'literature_writing music
': [0.8333333333333334, 0.16666666666666666], 'history_geography music': [1.0, 0.0], 'hea

': [0.8333333333333334, 0.16666666666666666], 'history_geography music': [1.0, 0.0], 'hea
lth_lifescience socialsciences': [0.84375, 0.15625], 'esl socialsciences': [0.72727272727
27273, 0.2727272727272727], 'economics socialsciences': [1.0, 0.0], 'earlydevelopment per
formingarts': [1.0, 0.0], 'music visualarts': [0.7142857142857143, 0.2857142857142857], '
extracurricular': [0.7619047619047619, 0.23809523809523808], 'foreignlanguages history_ge
ography': [0.6666666666666666, 0.3333333333333333], 'extracurricular literature_writing':
[0.9090909090909091, 0.09090909090909091], 'appliedsciences esl': [0.8888888888888888, 0.
1111111111111111], 'health_wellness history_geography': [0.75, 0.25], 'esl nutritioneduca
tion': [1.0, 0.0], 'esl environmentalscience': [0.6666666666666666, 0.3333333333333333],
'communityservice earlydevelopment': [1.0, 0.0], 'communityservice health_wellness': [0.8
75, 0.125], 'mathematics performingarts': [0.9, 0.1], 'communityservice literacy': [0.857
1428571428571, 0.14285714285714285], 'parentinvolvement': [1.0, 0.0], 'socialsciences spe
cialneeds': [0.7142857142857143, 0.2857142857142857], 'extracurricular teamsports': [1.0,
0.0], 'performingarts visualarts': [0.7777777777777778, 0.2222222222222222], 'health_well
ness performingarts': [0.8333333333333334, 0.16666666666666666], 'gym_fitness performinga
rts': [0.75, 0.25], 'extracurricular other': [0.5714285714285714, 0.42857142857142855], '
civics_government communityservice': [0.8333333333333334, 0.16666666666666666], 'earlydev
elopment health_lifescience': [0.5, 0.5], 'parentinvolvement visualarts': [1.0, 0.0], 'fo
reignlanguages mathematics': [0.7272727272727273, 0.2727272727272727], 'performingarts sp
ecialneeds': [1.0, 0.0], 'college_careerprep environmentalscience': [0.625, 0.375], 'coll
ege_careerprep health_wellness': [0.6666666666666666, 0.3333333333333333], 'earlydevelopm
ent extracurricular': [0.6666666666666666, 0.3333333333333333], 'esl foreignlanguages': [
0.7857142857142857, 0.21428571428571427], 'college_careerprep earlydevelopment': [0.8, 0.
2], 'financialliteracy literacy': [1.0, 0.0], 'economics foreignlanguages': [1.0, 0.0], '
communityservice literature_writing': [0.8181818181818182, 0.18181818181818182], 'esl oth
er': [0.8571428571428571, 0.14285714285714285], 'foreignlanguages gym_fitness': [1.0, 0.0
], 'economics financialliteracy': [0.8333333333333334, 0.16666666666666666], 'gym_fitness
literacy': [0.7857142857142857, 0.21428571428571427], 'economics': [0.9, 0.1], 'civics_go
vernment health_lifescience': [0.8333333333333334, 0.16666666666666666], 'civics_governme
nt esl': [1.0, 0.0], 'environmentalscience other': [1.0, 0.0], 'nutritioneducation specia
lneeds': [0.9166666666666666, 0.08333333333333333], 'nutritioneducation teamsports': [0.6
25, 0.375], 'charactereducation performingarts': [0.7142857142857143, 0.2857142857142857]
, 'earlydevelopment parentinvolvement': [0.7777777777777778, 0.2222222222222222], 'perfor
mingarts teamsports': [0.6, 0.4], 'charactereducation esl': [0.4, 0.6], 'appliedsciences
socialsciences': [0.8571428571428571, 0.14285714285714285], 'mathematics teamsports': [0.
0, 1.0], 'college_careerprep history_geography': [1.0, 0.0], 'esl parentinvolvement': [1.
0, 0.0], 'environmentalscience financialliteracy': [1.0, 0.0], 'health_wellness socialsci
ences': [0.8571428571428571, 0.14285714285714285], 'college_careerprep music': [0.5, 0.5]
, 'charactereducation health_lifescience': [0.7142857142857143, 0.2857142857142857], 'civ
ics_government foreignlanguages': [1.0, 0.0], 'literacy teamsports': [0.6, 0.4], 'extracu
rricular health_lifescience': [0.5, 0.5], 'college_careerprep teamsports': [1.0, 0.0], 'f
oreignlanguages health_wellness': [1.0, 0.0], 'parentinvolvement specialneeds': [0.666666
6666666666, 0.3333333333333333], 'health_lifescience teamsports': [1.0, 0.0], 'health_lif
escience performingarts': [1.0, 0.0], 'earlydevelopment foreignlanguages': [1.0, 0.0], 'o
ther parentinvolvement': [1.0, 0.0], 'mathematics parentinvolvement': [0.8333333333333334
, 0.16666666666666666], 'health_lifescience music': [1.0, 0.0], 'specialneeds teamsports'
: [0.6, 0.4], 'gym_fitness other': [1.0, 0.0], 'appliedsciences foreignlanguages': [0.0,
1.0], 'economics history_geography': [0.9090909090909091, 0.09090909090909091], 'music te
amsports': [0.6666666666666666, 0.3333333333333333], 'communityservice parentinvolvement'
: [1.0, 0.0], 'college_careerprep foreignlanguages': [1.0, 0.0], 'economics literacy': [1
.0, 0.0], 'teamsports visualarts': [1.0, 0.0], 'history_geography performingarts': [1.0,
0.0], 'civics_government health_wellness': [1.0, 0.0], 'appliedsciences civics_government
': [0.6, 0.4], 'health_lifescience parentinvolvement': [0.5, 0.5], 'literacy nutritionedu
cation': [0.8, 0.2], 'communityservice extracurricular': [0.6666666666666666, 0.333333333
3333333], 'appliedsciences economics': [1.0, 0.0], 'appliedsciences teamsports': [0.66666
66666666666, 0.3333333333333333], 'foreignlanguages other': [1.0, 0.0], 'literature_writi
ng teamsports': [0.5, 0.5], 'foreignlanguages health_lifescience': [1.0, 0.0], 'extracurr
icular parentinvolvement': [1.0, 0.0], 'civics_government visualarts': [0.75, 0.25], 'eco
nomics health_lifescience': [1.0, 0.0], 'charactereducation music': [0.8, 0.2], 'nutritio
neducation other': [1.0, 0.0], 'financialliteracy health_wellness': [1.0, 0.0], 'earlydev
elopment nutritioneducation': [0.5, 0.5], 'gym_fitness music': [0.75, 0.25], 'communityse
rvice performingarts': [1.0, 0.0], 'gym_fitness visualarts': [0.8, 0.2], 'appliedsciences
communityservice': [0.6666666666666666, 0.3333333333333333], 'appliedsciences parentinvol
vement': [0.75, 0.25], 'foreignlanguages visualarts': [0.25, 0.75], 'gym_fitness parentin
volvement': [1.0, 0.0], 'appliedsciences gym_fitness': [0.8333333333333334, 0.16666666666
666666], 'foreignlanguages music': [0.6666666666666666, 0.3333333333333333], 'college_car
eerprep gym_fitness': [1.0, 0.0], 'earlydevelopment music': [0.7777777777777778, 0.222222
2222222222], 'health_lifescience other': [1.0, 0.0], 'college_careerprep performingarts':
[0.6666666666666666, 0.3333333333333333], 'mathematics nutritioneducation': [0.6666666666
666666, 0.3333333333333333], 'charactereducation foreignlanguages': [1.0, 0.0], 'applieds
ciences nutritioneducation': [1.0, 0.0], 'environmentalscience parentinvolvement': [1.0,
0.0], 'environmentalscience extracurricular': [1.0, 0.0], 'communityservice health_lifesc
ience': [0.5, 0.5], 'music socialsciences': [1.0, 0.0], 'charactereducation socialsciences

```
ience': [0.5, 0.5], 'music socialsciences': [1.0, 0.0], 'charactereducation socialscience
s': [1.0, 0.0], 'charactereducation history_geography': [1.0, 0.0], 'esl gym_fitness': [0
.0, 1.0], 'civics_government financialliteracy': [1.0, 0.0], 'extracurricular health_well
ness': [0.75, 0.25], 'extracurricular specialneeds': [1.0, 0.0], 'nutritioneducation soci
alsciences': [1.0, 0.0], 'civics_government specialneeds': [1.0, 0.0], 'communityservice
financialliteracy': [1.0, 0.0], 'earlydevelopment history_geography': [1.0, 0.0], 'colleg
e_careerprep financialliteracy': [0.5, 0.5], 'communityservice socialsciences': [0.666666
6666666666, 0.3333333333333333], 'economics literature_writing': [1.0, 0.0], 'college_car
eerprep nutritioneducation': [0.75, 0.25], 'environmentalscience performingarts': [0.5, 0
.5], 'foreignlanguages socialsciences': [0.0, 1.0], 'economics music': [1.0, 0.0], 'civic
s_government college_careerprep': [1.0, 0.0], 'environmentalscience foreignlanguages': [1
.0, 0.0], 'civics_government performingarts': [1.0, 0.0], 'charactereducation nutritioned
ucation': [1.0, 0.0], 'history_geography warmth care_hunger': [0.0, 1.0], 'economics othe
r': [1.0, 0.0], 'college_careerprep esl': [1.0, 0.0], 'environmentalscience gym_fitness':
[1.0, 0.0], 'literature_writing nutritioneducation': [1.0, 0.0], 'other teamsports': [1.0
, 0.0], 'other socialsciences': [0.0, 1.0], 'nutritioneducation visualarts': [1.0, 0.0],
'charactereducation financialliteracy': [1.0, 0.0], 'extracurricular gym_fitness': [1.0,
0.0], 'extracurricular nutritioneducation': [1.0, 0.0], 'earlydevelopment economics': [1.
0, 0.0]}
Shape after vectorization
(33500, 2) (33500,)
(16500, 2) (16500,)
```

Encoding Sentiment Score : sen_neg

In [31]:

```python
train_neg_ss = np.array(X_train['sen_neg']).reshape(-1,1)
test_neg_ss = np.array(X_test['sen_neg']).reshape(-1,1)

print("Shape after vectorization")
print(train_neg_ss.shape,y_train.shape)
print(test_neg_ss.shape,y_test.shape)
```

```
Shape after vectorization
(33500, 1) (33500,)
(16500, 1) (16500,)
```

Encoding Sentiment Score : sen_pos

In [32]:

```python
train_pos_ss = np.array(X_train['sen_pos']).reshape(-1,1)
test_pos_ss = np.array(X_test['sen_pos']).reshape(-1,1)

print("Shape after vectorization")
print(train_pos_ss.shape,y_train.shape)
print(test_pos_ss.shape,y_test.shape)
```

```
Shape after vectorization
(33500, 1) (33500,)
(16500, 1) (16500,)
```

Encoding Sentiment Score : sen_neu

In [33]:

```python
train_neu_ss = np.array(X_train['sen_neu']).reshape(-1,1)
test_neu_ss = np.array(X_test['sen_neu']).reshape(-1,1)

print("Shape after vectorization")
print(train_neu_ss.shape,y_train.shape)
print(test_neu_ss.shape,y_test.shape)
```

```
Shape after vectorization
(33500, 1) (33500,)
(16500, 1) (16500,)
```

Encoding Sentiment Score : sen_comp

In [34]:

```
train_comp_ss = np.array(X_train['sen_comp']).reshape(-1,1)
test_comp_ss = np.array(X_test['sen_comp']).reshape(-1,1)

print("Shape after vectorization")
print(train_comp_ss.shape,y_train.shape)
print(test_comp_ss.shape,y_test.shape)
```

```
Shape after vectorization
(33500, 1) (33500,)
(16500, 1) (16500,)
```

Concatinating all the Features

Set 1: categorical(response coding) + numerical features + preprocessed_eassay(TFIDF) + Sentiment scores(preprocessed_essay)

In [35]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_train_set1 = hstack((train_essay_tfidf,train_teacher_prefix_res,train_project_grade_cat
egory_res,train_school_state_res,train_clean_categories_res,train_clean_subcategories_res
,train_price_norm,train_teacher_number_of_previously_posted_projects_norm,train_neg_ss,tr
ain_pos_ss,train_neu_ss,train_comp_ss)).tocsr()
X_test_set1 = hstack((test_essay_tfidf,test_teacher_prefix_res,test_project_grade_categor
y_res,test_school_state_res,test_clean_categories_res,test_clean_subcategories_res,test_p
rice_norm,test_teacher_number_of_previously_posted_projects_norm,test_neg_ss,test_pos_ss,
test_neu_ss,test_comp_ss)).tocsr()

print("Final Data Matrix")
print(X_train_set1.shape,y_train.shape)
print(X_test_set1.shape,y_test.shape)
```

```
Final Data Matrix
(33500, 5016) (33500,)
(16500, 5016) (16500,)
```

In [36]:

```
print(train_tfidf_w2v.shape)
print(test_tfidf_w2v.shape)
```

```
(33500, 300)
(16500, 300)
```

Set 2: categorical(response coding) + numerical features + preprocessed_eassay (TFIDF W2V)

In [37]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_train_set2 = hstack((train_tfidf_w2v,train_teacher_prefix_res,train_project_grade_categ
ory_res,train_school_state_res,train_clean_categories_res,train_clean_subcategories_res,t
rain_price_norm,train_teacher_number_of_previously_posted_projects_norm)).tocsr()
X_test_set2 = hstack((test_tfidf_w2v,test_teacher_prefix_res,test_project_grade_category_
res,test_school_state_res,test_clean_categories_res,test_clean_subcategories_res,test_pri
ce_norm,test_teacher_number_of_previously_posted_projects_norm)).tocsr()

print("Final Data Matrix")
print(X_train_set2.shape,y_train.shape)
print(X_test_set2.shape,y_test.shape)
```

```
Final Data Matrix
```

```
Final Data Matrix
(33500, 312) (33500,)
(16500, 312) (16500,)
```

In [ ]:

```
pip install lightgbm
```

```
Requirement already satisfied: lightgbm in /usr/local/lib/python3.7/dist-packages (2.2.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from ligh
tgbm) (1.19.5)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (fr
om lightgbm) (0.22.2.post1)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from ligh
tgbm) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (fr
om scikit-learn->lightgbm) (1.0.1)
```

Hyper Parameter Tuning Using GridSearchCV on Set-1 [categorical(response coding) + numerical features + preprocessed_eassay(TFIDF) + Sentiment scores(preprocessed_essay)]

In [ ]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.h
tml
#https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClass
ifier.html
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
import lightgbm as lgb

clf_lgb = lgb.LGBMClassifier()
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
min_samples_split =[5, 10, 100, 500]
param_dict = {'learning_rate':learning_rate,'min_samples_split':min_samples_split}

grid = GridSearchCV(clf_lgb,param_grid=param_dict,scoring='roc_auc',return_train_score=T
rue)
grid.fit(X_train_set1,y_train)

#Printing the best hyper parameter values and the best auc score on cv data
print("The Best AUC Score on cross validation data : {:.3f} ".format(grid.best_score_))
print("The Best Hyper Parameters :",grid.best_params_)

best_params = grid.best_params_
best_learning_rate = best_params.get('learning_rate')
best_min_sample_split = best_params.get('min_samples_split')

results = pd.DataFrame.from_dict(grid.cv_results_)
results = results.sort_values(['param_learning_rate','param_min_samples_split'],ascendin
g=(True,True))

train_auc = results['mean_train_score'].tolist()
cv_auc = results['mean_test_score'].tolist()
#train_auc_std = results['std_train_score']
#cv_auc_std = results['std_test_score']
learning_rate = results['param_learning_rate'].tolist()
min_samples_split = results['param_min_samples_split'].tolist()

print(best_learning_rate)
print(best_min_sample_split)
#results.head(3)
```

```
The Best AUC Score on cross validation data : 0.704
The Best Hyper Parameters : {'learning_rate': 0.1, 'min_samples_split': 5}
0.1
5
```

In [41]:

```
from sklearn.externals import joblib
grid = joblib.load('model_joblib')
```

In [46]:

```
#Printing the best hyper parameter values and the best auc score on cv data
print("The Best AUC Score on cross validation data : {:.3f} ".format(grid.best_score_))
print("The Best Hyper Parameters :",grid.best_params_)

best_params = grid.best_params_
best_learning_rate = best_params.get('learning_rate')
best_min_sample_split = best_params.get('min_samples_split')

results = pd.DataFrame.from_dict(grid.cv_results_)
results = results.sort_values(['param_learning_rate','param_min_samples_split'],ascendin
g=(True,True))

train_auc = results['mean_train_score'].tolist()
cv_auc = results['mean_test_score'].tolist()
#train_auc_std = results['std_train_score']
#cv_auc_std = results['std_test_score']
learning_rate = results['param_learning_rate'].tolist()
min_samples_split = results['param_min_samples_split'].tolist()

print(best_learning_rate)
print(best_min_sample_split)
#results.head(3)
```

```
The Best AUC Score on cross validation data : 0.704
The Best Hyper Parameters : {'learning_rate': 0.1, 'min_samples_split': 5}
0.1
5
```

3-D plot representing the performance for train and cross valdation data for each value of hyper parameters

In [43]:

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

In [44]:

```
def configure_plotly_browser_state():
      import IPython
      display(IPython.core.display.HTML('''
                 <script src="/static/components/requirejs/require.js"></script>
                 <script>
                      requirejs.config({
                          paths: {
                             base: '/static/base',
                            plotly: 'https://cdn.plot.ly/plotly-latest.min.js?noext',
                          },
                       });
                  </script>
                  '''))
```

In [47]:

```
# https://plot.ly/python/3d-axes/

x1=learning_rate
y1=min_samples_split
```

```python
z1=train_auc
x2=learning_rate
y2=min_samples_split
z2=cv_auc

configure_plotly_browser_state()
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'Train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='learning_rate'),
        yaxis = dict(title='min_samples_split'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')

#fig.show()
plt.show()
```

Testing the performance of the model on test data, plotting ROC Curves

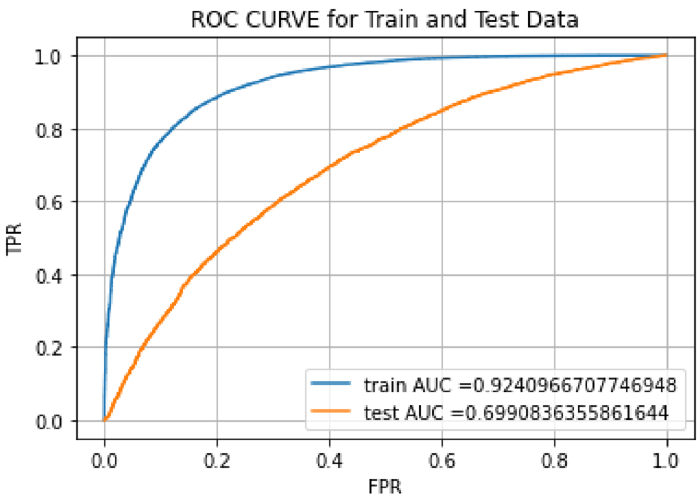
In [48]:

```python
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
the positive class
    # 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 = 49
000
    # 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
```

In [49]:

```
from sklearn.metrics import roc_curve, auc
import lightgbm as lgb

clf_tfidf = lgb.LGBMClassifier(min_samples_split=best_min_sample_split,learning_rate=best
_learning_rate)
clf_tfidf.fit(X_train_set1,y_train)

y_train_pred = batch_predict(clf_tfidf, X_train_set1)
y_test_pred = batch_predict(clf_tfidf, X_test_set1)

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("ROC CURVE for Train and Test Data")
plt.grid()
plt.show()
```



Plotting the Confusion Matrix for Test Data Point

In [50]:

```
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round
(t,3))
    return t

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

In [51]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
#print("Train confusion matrix")
#print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
#print("Test confusion matrix")
#print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))

#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
matrix = confusion_matrix(y_test,predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 0.7152081521475392 for threshold 0.806
```

In [52]:

```
print(matrix)
```

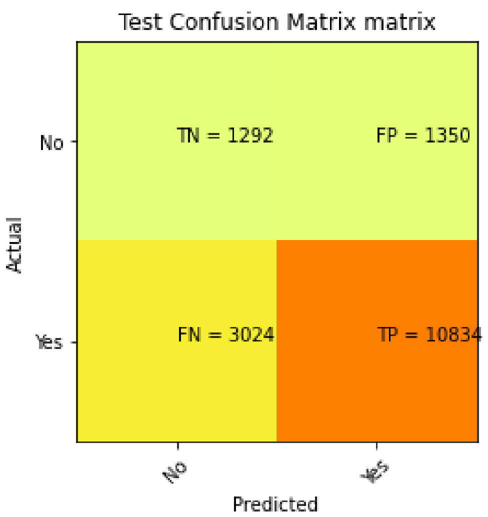
```
[[ 1292  1350]
 [ 3024 10834]]
```

In [53]:

```
def myplot_matrix1(data):
    plt.clf()
    plt.imshow(data,interpolation='nearest',cmap=plt.cm.Wistia)
    classNames = ['No','Yes']
    plt.title("Test Confusion Matrix matrix")
    tick_marks = np.arange(len(classNames))

    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.xticks(tick_marks,classNames,rotation=45)
    plt.yticks(tick_marks,classNames)
    s=[['TN','FP'],['FN','TP']]
    for i in range(2):
        for j in range(2):
            plt.text(j,i,str(s[i][j])+" = "+str(data[i][j]))
    plt.show()
```

In [54]:

```
myplot_matrix1(matrix)
```



Hyper Parameter Tuning Using GridSearchCV on Set-2 [categorical(response coding) + numerical features + preprocessed_eassay (TFIDF W2V)]

In [55]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.h
```

```
tml
#https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClass
ifier.html
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import GradientBoostingClassifier
import lightgbm as lgb

clf_lgb = lgb.LGBMClassifier()
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
min_samples_split =[5, 10, 100, 500]
param_dict = {'learning_rate':learning_rate,'min_samples_split':min_samples_split}

grid = GridSearchCV(clf_lgb,param_grid=param_dict,scoring='roc_auc',return_train_score=T
rue)
grid.fit(X_train_set2,y_train)

#Printing the best hyper parameter values and the best auc score on cv data
print("The Best AUC Score on cross validation data : {:.3f} ".format(grid.best_score_))
print("The Best Hyper Parameters :",grid.best_params_)

best_params = grid.best_params_
best_learning_rate = best_params.get('learning_rate')
best_min_sample_split = best_params.get('min_samples_split')

results = pd.DataFrame.from_dict(grid.cv_results_)
results = results.sort_values(['param_learning_rate','param_min_samples_split'],ascendin
g=(True,True))

train_auc = results['mean_train_score'].tolist()
cv_auc = results['mean_test_score'].tolist()
#train_auc_std = results['std_train_score']
#cv_auc_std = results['std_test_score']
learning_rate = results['param_learning_rate'].tolist()
min_samples_split = results['param_min_samples_split'].tolist()

print(best_learning_rate)
print(best_min_sample_split)
#results.head(3)
```

```
The Best AUC Score on cross validation data : 0.691
The Best Hyper Parameters : {'learning_rate': 0.1, 'min_samples_split': 5}
0.1
5
```

In [57]:

```
joblib.dump(grid,'model2_joblib')
```

Out[57]:

```
['model2_joblib']
```

In [62]:

```
grid = joblib.load('model2_joblib')
```

3-D plot representing the performance for train and cross valdation data for each value of hyper parameters

In [63]:

```
# https://plot.ly/python/3d-axes/

x1=learning_rate
y1=min_samples_split
z1=train_auc
x2=learning_rate
y2=min_samples_split
z2=cv_auc
```

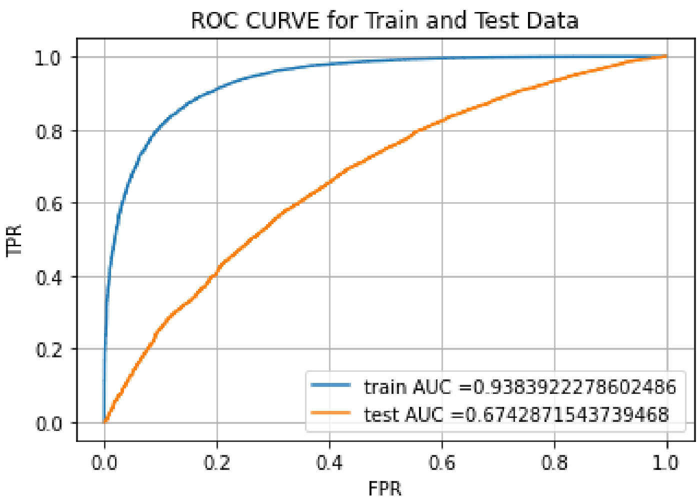
```
configure_plotly_browser_state()
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'Train')
trace2 = go.Scatter3d(x=x2,y=y2,z=z2, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='learning_rate'),
        yaxis = dict(title='min_samples_split'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')

#fig.show()
plt.show()
```

Testing the performance of the model on test data, plotting ROC Curves

In [66]:

```
from sklearn.metrics import roc_curve, auc
import lightgbm as lgb

clf_tfidf_w2v = lgb.LGBMClassifier(min_samples_split=best_min_sample_split,learning_rate=
best_learning_rate)
clf_tfidf_w2v.fit(X_train_set2,y_train)

y_train_pred = batch_predict(clf_tfidf_w2v, X_train_set2)
y_test_pred = batch_predict(clf_tfidf_w2v, X_test_set2)

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("ROC CURVE for Train and Test Data")
plt.grid()
plt.show()
```



Plotting the Confusion Matrix for Test Data Point

In [67]:

```
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round
(t,3))
    return t

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

In [68]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
#print("Train confusion matrix")
#print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
#print("Test confusion matrix")
#print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))

#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
matrix = confusion_matrix(y_test,predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 0.7421995700068783 for threshold 0.8
```

In [69]:

```
print(matrix)
```

```
[[ 1187  1455]
 [ 2915 10943]]
```

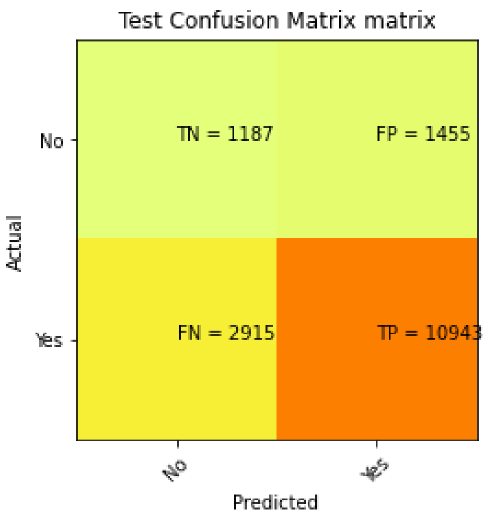
In [70]:

```
def myplot_matrix1(data):
    plt.clf()
    plt.imshow(data,interpolation='nearest',cmap=plt.cm.Wistia)
```

```
    classNames = ['No','Yes']
    plt.title("Test Confusion Matrix matrix")
    tick_marks = np.arange(len(classNames))

    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.xticks(tick_marks,classNames,rotation=45)
    plt.yticks(tick_marks,classNames)
    s=[['TN','FP'],['FN','TP']]
    for i in range(2):
        for j in range(2):
            plt.text(j,i,str(s[i][j])+" = "+str(data[i][j]))
    plt.show()
```

In [71]:

```
myplot_matrix1(matrix)
```



Summary :

In [73]:

```
#https://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer","Model","Best HyperParameter","Train AUC","Test AUC"]
x.add_row(["TFIDF","GBDT","learning_rate=0.1 & min_samples_split=5","0.924","0.699"])
x.add_row(["TFIDF W2V","GBDT","learning_rate=0.1 & min_samples_split=5","0.938","0.674"]
)
```

In [74]:

```
print(x)
```

```
+------------+-------+-----------------------------------------+-----------+----------+
| Vectorizer | Model |           Best HyperParameter           | Train AUC | Test AUC |
+------------+-------+-----------------------------------------+-----------+----------+
|   TFIDF    |  GBDT | learning_rate=0.1 & min_samples_split=5 |   0.924   |  0.699   |
| TFIDF W2V  |  GBDT | learning_rate=0.1 & min_samples_split=5 |   0.938   |  0.674   |
+------------+-------+-----------------------------------------+-----------+----------+
```