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**

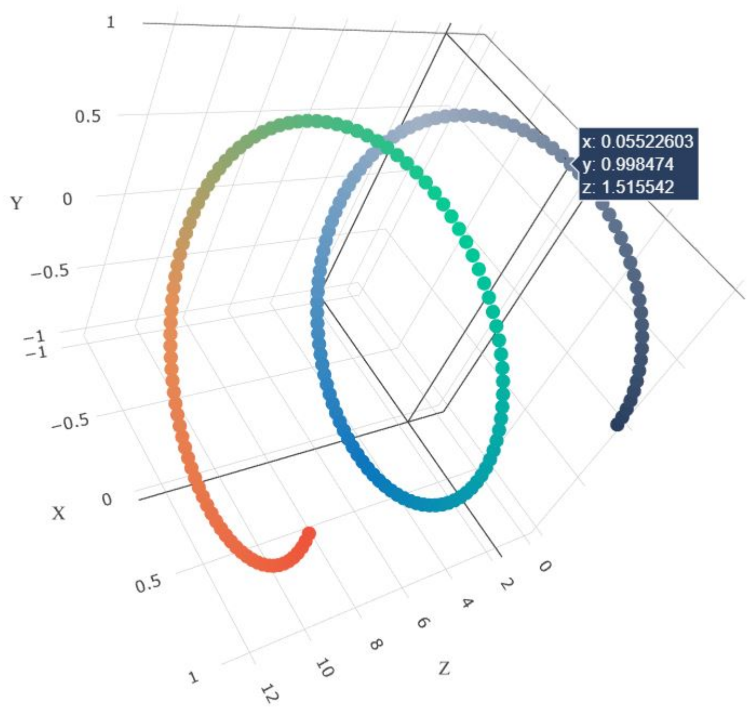
    - Set 1: categorical(instead of one hot encoding, try response coding (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check the bellow example, include all 4 values as 4 features)
    - Set 2: categorical(instead of one hot encoding, try response coding (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) 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**

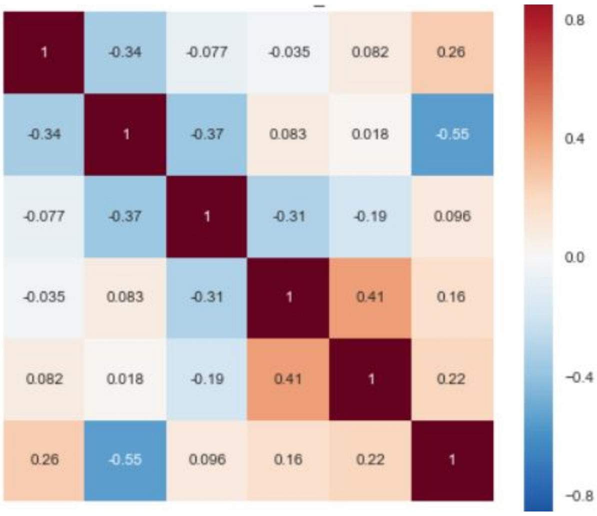
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



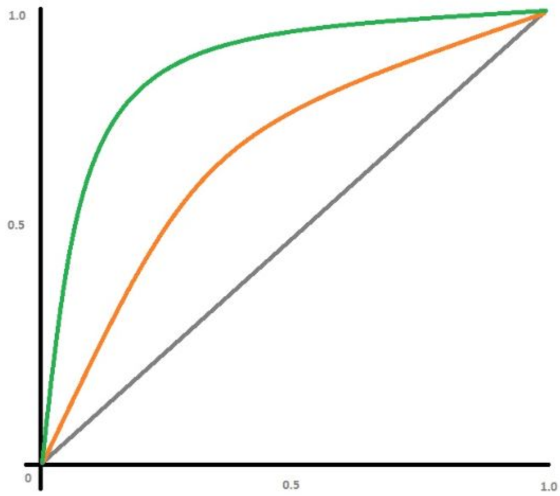
with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

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 (https://seaborn.pydata.org/generated/seaborn.heatmap.html) 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

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

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

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

In [93]:

```
import pandas as pd
import numpy as np
from tqdm import tqdm
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```

```

1. GBDT (xgboost/lightgbm)

1.1 Loading Data

In [94]:

```
data = pd.read_csv(r'../input/donorschoosewithglovevectors/preprocessed_data.cs
v')
```

In [95]:

```
data.head(1)
```

Out[95]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_
0	ca	mrs	grades_prek_2	53

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

In [96]:

```
# please write all the code with proper documentation, and proper titles for each
 subsection
# 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 code
# 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
y=data['project_is_approved'].values
x=data.drop(['project_is_approved'],axis=1)
x.head(1)
```

Out[96]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_
0	ca	mrs	grades_prek_2	53

In [97]:

```
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,random_state
=42,stratify=y)
```

In [98]:

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(73196, 8)
(73196,)
(36052, 8)
(36052,)
```

1.3 Make Data Model Ready: encoding essay, and project_title

Encoding Essay with TF-IDF

In [99]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

TFIDF = TfidfVectorizer(min_df=10,ngram_range=(1,1),stop_words='english')
TFIDF.fit(x_train['essay'].values)

x_train_tfidf = TFIDF.transform(x_train['essay'].values)
x_test_tfidf = TFIDF.transform(x_test['essay'].values)
```

In [100]:

```
print(x_train_tfidf.shape, y_train.shape)
print(x_test_tfidf.shape,y_test.shape)
```

```
(73196, 13985) (73196,)
(36052, 13985) (36052,)
```

1.4 Make Data Model Ready: encoding numerical, categorical features

Encoding Numerical Features

In [101]:

```
from sklearn.preprocessing import Normalizer

#price
normalizer = Normalizer()
normalizer.fit(x_train['price'].values.reshape(1,-1))

x_train_price_normalized = normalizer.transform(x_train['price'].values.reshape(
1,-1)).reshape(-1,1)
x_test_price_normalized = normalizer.transform(x_test['price'].values.reshape(1,
-1)).reshape(-1,1)

print(x_train_price_normalized.shape)
print(x_test_price_normalized.shape)

#teacher_number_of_previously_posted_projects
normalizer.fit(x_train['teacher_number_of_previously_posted_projects'].values.re
shape(1,-1))

x_train_teacher_number_of_previously_posted_projects_normalized = normalizer.tra
nsform(x_train['teacher_number_of_previously_posted_projects'].values.reshape(1,
-1)).reshape(-1,1)
x_test_teacher_number_of_previously_posted_projects_normalized = normalizer.tran
sform(x_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1
)).reshape(-1,1)

print(x_train_teacher_number_of_previously_posted_projects_normalized.shape)
print(x_test_teacher_number_of_previously_posted_projects_normalized.shape)
```

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

Encoding Categorical Features

In [102]:

```
x_train.insert(loc=len(x_train.columns),column='y_dummy',value=y_train)
```

In [103]:

```
x_test.insert(loc=len(x_test.columns),column='y_dummy',value=y_test)
```

In [104]:

```
def calculate_prob_score(xtrain,feature):
    positive_prob_score={}
    negative_prob_score={}
    unique_values = xtrain[feature].unique()
    for value in unique_values:
        positive_count = len(xtrain.loc[(xtrain[feature]==value) & (xtrain['y_du
mmy']==1)])
        negative_count = len(xtrain.loc[(xtrain[feature]==value) & (xtrain['y_du
mmy']==0)])
        positive_prob_score[value]=(positive_count/(positive_count+negative_coun
t))
        negative_prob_score[value]=(negative_count/(positive_count+negative_coun
t))
    return positive_prob_score, negative_prob_score

def response_coding(xtrain,xtest,feature):
    no_of_values = x_train['y_dummy'].nunique()
    for i in range(0,no_of_values):
        index = x_train.columns.get_loc(feature)
        x_train.insert(loc=index,column=feature+'_'+str(i),value=x_train[feature
]) #creating two columns for each categorical features as _0 and _1
        x_test.insert(loc=index,column=feature+'_'+str(i),value=x_test[feature])
    positive,negative = calculate_prob_score(xtrain,feature)
    unique_test_values = x_test[feature].unique()
    for value in unique_test_values:
        if value not in positive:
            positive[value]=0.5
            negative[value]=0.5

    for key,value in positive.items():
        x_train[feature+'_'+str(i)] = np.where((x_train[feature]==key),value,x_t
rain[feature+'_'+str(i)])
        x_test[feature+'_'+str(i)] = np.where((x_test[feature]==key),value,x_tes
t[feature+'_'+str(i)])
    arr1 = x_train[feature+'_'+str(i)].values
    arr2 = x_test[feature+'_'+str(i)].values
    i=i-1
    for key,value in negative.items():
        x_train[feature+'_'+str(i)] = np.where((x_train[feature]==key),value,x_t
rain[feature+'_'+str(i)])
        x_test[feature+'_'+str(i)] = np.where((x_test[feature]==key),value,x_tes
t[feature+'_'+str(i)])
    arr3 = x_train[feature+'_'+str(i)].values
```

```
    arr4 = x_test[feature+'_'+str(i)].values

    return arr1,arr2,arr3,arr4
```

In [105]:

```
#school_state
xtrain_school_state_0, xtest_school_state_0, xtrain_school_state_1, xtest_school
_state_1 = response_coding(x_train,x_test,'school_state')
xtrain_school_state_0 = xtrain_school_state_0.reshape(-1,1).astype(float)
xtrain_school_state_1 = xtrain_school_state_1.reshape(-1,1).astype(float)
xtest_school_state_0 = xtest_school_state_0.reshape(-1,1).astype(float)
xtest_school_state_1 = xtest_school_state_1.reshape(-1,1).astype(float)
```

In [106]:

```
#teacher_prefix
xtrain_teacher_prefix_0, xtest_teacher_prefix_0, xtrain_teacher_prefix_1, xtest_
teacher_prefix_1 = response_coding(x_train,x_test,'teacher_prefix')
xtrain_teacher_prefix_0 = xtrain_teacher_prefix_0.reshape(-1,1).astype(float)
xtrain_teacher_prefix_1 = xtrain_teacher_prefix_1.reshape(-1,1).astype(float)
xtest_teacher_prefix_0 = xtest_teacher_prefix_0.reshape(-1,1).astype(float)
xtest_teacher_prefix_1 = xtest_teacher_prefix_1.reshape(-1,1).astype(float)
```

In [107]:

```
#project_grade_category
xtrain_project_grade_category_0, xtest_project_grade_category_0, xtrain_project_grade_category_1, xtest_project_grade_category_1 = response_coding(x_train,x_test,'project_grade_category')
xtrain_project_grade_category_0 = xtrain_project_grade_category_0.reshape(-1,1).astype(float)
xtrain_project_grade_category_1 = xtrain_project_grade_category_1.reshape(-1,1).astype(float)
xtest_project_grade_category_0 = xtest_project_grade_category_0.reshape(-1,1).astype(float)
xtest_project_grade_category_1 = xtest_project_grade_category_1.reshape(-1,1).astype(float)
```

In [108]:

```
#clean_categories
xtrain_clean_categories_0, xtest_clean_categories_0, xtrain_clean_categories_1,
xtest_clean_categories_1 = response_coding(x_train,x_test,'clean_categories')
xtrain_clean_categories_0 = xtrain_clean_categories_0.reshape(-1,1).astype(float)
xtrain_clean_categories_1 = xtrain_clean_categories_1.reshape(-1,1).astype(float)
xtest_clean_categories_0 = xtest_clean_categories_0.reshape(-1,1).astype(float)
xtest_clean_categories_1 = xtest_clean_categories_1.reshape(-1,1).astype(float)
```

In [109]:

```
#clean_subcategories
xtrain_clean_subcategories_0, xtest_clean_subcategories_0, xtrain_clean_subcategories_1, xtest_clean_subcategories_1 = response_coding(x_train,x_test,'clean_subcategories')
xtrain_clean_subcategories_0 = xtrain_clean_subcategories_0.reshape(-1,1).astype(float)
xtrain_clean_subcategories_1 = xtrain_clean_subcategories_1.reshape(-1,1).astype(float)
xtest_clean_subcategories_0 = xtest_clean_subcategories_0.reshape(-1,1).astype(float)
xtest_clean_subcategories_1 = xtest_clean_subcategories_1.reshape(-1,1).astype(float)
```

In [64]:

```

```

Out[64]:

```
array([[0.00161251],
       [0.00049616],
       [0.        ],
       ...,
       [0.00012404],
       [0.00012404],
       [0.        ]])
```

1.5 Taking sentiment values of essay from sentimentAnalyzer()

In [110]:

```
x_train_essay = x_train['essay'].values
x_test_essay = x_test['essay'].values
```

In [111]:

```
print(x_train_essay.shape,x_test_essay.shape)
```

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

In [112]:

```
x_train_essay[0]
```

Out[112]:

```
'our school title i elementary school located socio economically di
sadvantaged area city we diverse group largest student population p
rimarily african american 90 receive free reduced lunch we avid sch
ool want students become life long learners college bond my student
s amazing they young fun ready learn the students classroom love le
arn thirst knowledge curiosity world having chromebooks room allow
students access self paced learning curriculum allowing successful
learning hands approach we limited technology building computers i
classroom pulled state assessments weeks time leaving students no c
omputers classroom we avid school chromebooks classroom help become
college career ready 21st technology skills chromebooks used litera
cy math centers students develop skills class regular basis helping
remain class within class setting this provide highly engaging nece
ssary tool academic skills nannan'
```

In [113]:

```
sid = SentimentIntensityAnalyzer()
for_sentiment = x_train_essay[0]
ss=sid.polarity_scores(for_sentiment)
print(ss['neg'])
```

```
0.045
```

In [114]:

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

sid = SentimentIntensityAnalyzer()
neg_train=[]
neu_train=[]
pos_train=[]
compound_train=[]

neg_test=[]
neu_test=[]
pos_test=[]
compound_test=[]

for description in tqdm(x_train_essay):
    for_sentiment = description
    ss = sid.polarity_scores(for_sentiment)
    neg_train.append(ss['neg'])
    neu_train.append(ss['neu'])
    pos_train.append(ss['pos'])
    compound_train.append(ss['compound'])

for description in tqdm(x_test_essay):
    for_sentiment = description
    ss = sid.polarity_scores(for_sentiment)
    neg_test.append(ss['neg'])
    neu_test.append(ss['neu'])
    pos_test.append(ss['pos'])
    compound_test.append(ss['compound'])



# 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
```

```
100%|██████████| 73196/73196 [02:52<00:00, 423.86it/s]
100%|██████████| 36052/36052 [01:26<00:00, 417.54it/s]
```

In [115]:

```
#reshaping train sentiment values
neg_train_sentiment = np.array(neg_train).reshape(-1,1).astype('float')
neu_train_sentiment = np.array(neu_train).reshape(-1,1).astype('float')
pos_train_sentiment = np.array(pos_train).reshape(-1,1).astype('float')
compound_train_sentiment = np.array(compound_train).reshape(-1,1).astype('float'
)

#reshaping test sentiment values
neg_test_sentiment = np.array(neg_test).reshape(-1,1).astype('float')
neu_test_sentiment = np.array(neu_test).reshape(-1,1).astype('float')
pos_test_sentiment = np.array(pos_test).reshape(-1,1).astype('float')
compound_test_sentiment = np.array(compound_test).reshape(-1,1).astype('float')
```

1.6 Creating total dataset with tf-idf values and converted values

In [116]:

```
x_train.head(1)
```

Out[116]:

	school_state_0	school_state_1	school_state	teacher_prefix_0	teacher_prefix_1
76564	0.143868	0.856132	ks	0.1454	0.8546

In [117]:

```
x_train['price']=x_train_price_normalized
x_train['teacher_number_of_previously_posted_projects']=x_train_teacher_number_o
f_previously_posted_projects_normalized
```

In [118]:

```
from scipy.sparse import hstack

X_Train = hstack((x_train_tfidf,
                  xtrain_school_state_0,
                  xtrain_school_state_1,
                  xtrain_teacher_prefix_0,
                  xtrain_teacher_prefix_1,
                  xtrain_project_grade_category_0,
                  xtrain_project_grade_category_1,
                  xtrain_clean_categories_0,
                  xtrain_clean_categories_1,
                  xtrain_clean_subcategories_0,
                  xtrain_clean_subcategories_1,
                  x_train_price_normalized,
                  x_train_teacher_number_of_previously_posted_projects_normalize
d,
                  neg_train_sentiment,
                  neu_train_sentiment,
                  pos_train_sentiment,
                  compound_train_sentiment)).tocsr()

X_Test = hstack((x_test_tfidf,
                  xtest_school_state_0,
                  xtest_school_state_1,
                  xtest_teacher_prefix_0,
                  xtest_teacher_prefix_1,
                  xtest_project_grade_category_0,
                  xtest_project_grade_category_1,
                  xtest_clean_categories_0,
                  xtest_clean_categories_1,
                  xtest_clean_subcategories_0,
                  xtest_clean_subcategories_1,
                  x_test_price_normalized,
                  x_test_teacher_number_of_previously_posted_projects_normalized
,
                  neg_test_sentiment,
                  neu_test_sentiment,
                  pos_test_sentiment,
                  compound_test_sentiment)).tocsr()
```

In [119]:

```
print(X_Train.shape,y_train.shape)
```

```
(73196, 14001) (73196,)
```

In [120]:

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

```
(36052, 14001) (36052,)
```

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

In [121]:

```
#feature set 1
import matplotlib.pyplot as plt
from lightgbm import LGBMClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import learning_curve, GridSearchCV




lgb = LGBMClassifier(class_weight='balanced', n_jobs=-1)
parameters = {'learning_rate':[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3], 'n_estimator
s': [5,10,50, 75, 100]}
clf=GridSearchCV(lgb, parameters, cv=3, scoring='roc_auc', verbose = 1,n_jobs=-1
, return_train_score=True)
clf.fit(X_Train, y_train)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed: 11.1min
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed: 23.1min finis
hed
```

Out[121]:

```
GridSearchCV(cv=3, estimator=LGBMClassifier(class_weight='balance
d'), n_jobs=-1,
             param_grid={'learning_rate': [0.0001, 0.001, 0.01, 0.
1, 0.2, 0.3],
                         'n_estimators': [5, 10, 50, 75, 100]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

In [122]:

```
clf.cv_results_
```

Out[122]:

```
{'mean_fit_time': array([31.49472706, 35.51734122, 62.91791582, 81.
34962781, 97.99702438,
        31.26217413, 34.70084953, 64.38943442, 81.24695683, 98.6822
4136,
        31.14912279, 34.600945  , 63.00851838, 80.64237309, 96.7752
858 ,
        31.21366278, 34.78375697, 62.02573816, 78.30426367, 92.7182
7141,
        31.07651464, 35.41402054, 62.33488067, 75.30314   , 88.4176
0055,
        32.04503123, 35.99542721, 60.97578406, 72.76324487, 71.6980
838 ]),
 'std_fit_time': array([ 0.38536574,  0.24428837,  0.49751653,  0.9
3254469,  0.52108269,
         0.19120069,  0.38140135,  0.52850363,  0.67983967,  0.9640
7813,
         0.41969185,  0.30008321,  0.40169958,  0.60220644,  0.6119
2403,
         0.32014768,  0.63039351,  0.53012353,  0.02393099,  0.2715
16  ,
         0.44311018,  0.66673047,  0.19387484,  0.83497976,  1.7004
2735,
         0.26685123,  0.5802893 ,  0.48280306,  0.7616759 , 10.1325
8237]),
 'mean_score_time': array([0.08122595, 0.06375567, 0.11568292, 0.14
975444, 0.17488456,
        0.05302779, 0.06266904, 0.12451196, 0.18308655, 0.20468775,
        0.05484716, 0.06441895, 0.15768393, 0.25566268, 0.29737655,
        0.05861799, 0.07411949, 0.18421714, 0.27388326, 0.39226031,
        0.05872536, 0.08097736, 0.2082804 , 0.24720097, 0.32736786,
        0.06056722, 0.07801755, 0.18672379, 0.25038592, 0.2441012
9]),
 'std_score_time': array([0.02097317, 0.00214332, 0.00504974, 0.001
49237, 0.00762106,
        0.00140672, 0.00318115, 0.00866583, 0.02495983, 0.00417266,
        0.00178139, 0.00365309, 0.00269842, 0.05249413, 0.00580359,
        0.00158465, 0.0012801 , 0.00297188, 0.01396307, 0.04640804,
        0.00193697, 0.00432793, 0.02020572, 0.01262349, 0.02330691,
        0.00153282, 0.00238112, 0.00219442, 0.00270617, 0.0503192
8]),
 'param_learning_rate': masked_array(data=[0.0001, 0.0001, 0.0001,
0.0001, 0.0001, 0.001, 0.001,
                    0.001, 0.001, 0.001, 0.01, 0.01, 0.01, 0.01, 0.
```

```
01, 0.1,
                    0.1, 0.1, 0.1, 0.1, 0.2, 0.2, 0.2, 0.2, 0.2, 0.
3, 0.3,
                    0.3, 0.3, 0.3],
              mask=[False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[5, 10, 50, 75, 100, 5, 1
0, 50, 75, 100, 5, 10, 50, 75,
                    100, 5, 10, 50, 75, 100, 5, 10, 50, 75, 100, 5,
10, 50,
                    75, 100],
              mask=[False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'learning_rate': 0.0001, 'n_estimators': 5},
  {'learning_rate': 0.0001, 'n_estimators': 10},
  {'learning_rate': 0.0001, 'n_estimators': 50},
  {'learning_rate': 0.0001, 'n_estimators': 75},
  {'learning_rate': 0.0001, 'n_estimators': 100},
  {'learning_rate': 0.001, 'n_estimators': 5},
  {'learning_rate': 0.001, 'n_estimators': 10},
  {'learning_rate': 0.001, 'n_estimators': 50},
  {'learning_rate': 0.001, 'n_estimators': 75},
  {'learning_rate': 0.001, 'n_estimators': 100},
  {'learning_rate': 0.01, 'n_estimators': 5},
  {'learning_rate': 0.01, 'n_estimators': 10},
  {'learning_rate': 0.01, 'n_estimators': 50},
  {'learning_rate': 0.01, 'n_estimators': 75},
  {'learning_rate': 0.01, 'n_estimators': 100},
  {'learning_rate': 0.1, 'n_estimators': 5},
  {'learning_rate': 0.1, 'n_estimators': 10},
  {'learning_rate': 0.1, 'n_estimators': 50},
```

```
  {'learning_rate': 0.1, 'n_estimators': 75},
  {'learning_rate': 0.1, 'n_estimators': 100},
  {'learning_rate': 0.2, 'n_estimators': 5},
  {'learning_rate': 0.2, 'n_estimators': 10},
  {'learning_rate': 0.2, 'n_estimators': 50},
  {'learning_rate': 0.2, 'n_estimators': 75},
  {'learning_rate': 0.2, 'n_estimators': 100},
  {'learning_rate': 0.3, 'n_estimators': 5},
  {'learning_rate': 0.3, 'n_estimators': 10},
  {'learning_rate': 0.3, 'n_estimators': 50},
  {'learning_rate': 0.3, 'n_estimators': 75},
  {'learning_rate': 0.3, 'n_estimators': 100}],
 'split0_test_score': array([0.65105788, 0.65186095, 0.65253923, 0.
65304922, 0.65317521,
        0.65239659, 0.65328536, 0.66221387, 0.6670235 , 0.6693398 ,
        0.66667282, 0.67223942, 0.68757225, 0.69621648, 0.70261909,
        0.68888198, 0.70009127, 0.73166798, 0.73421907, 0.73487489,
        0.69360857, 0.71134727, 0.73085804, 0.7303893 , 0.72682497,
        0.69749164, 0.71483685, 0.7238132 , 0.72409613, 0.7189143
2]),
 'split1_test_score': array([0.65831514, 0.65831514, 0.66115319, 0.
66206856, 0.66199622,
        0.66039514, 0.66181314, 0.66729485, 0.66841009, 0.66963204,
        0.66619447, 0.66995555, 0.69198598, 0.69711834, 0.7024618 ,
        0.68618739, 0.69530176, 0.73268002, 0.73673888, 0.73722858,
        0.69368867, 0.71187028, 0.73229591, 0.73082639, 0.73020135,
        0.695278  , 0.71241958, 0.72125224, 0.71605403, 0.7120542
]),
 'split2_test_score': array([0.65678587, 0.65678587, 0.65847651, 0.
65851284, 0.65854363,
        0.65678587, 0.65843812, 0.66176741, 0.66268828, 0.66513632,
        0.66136978, 0.66507263, 0.6861325 , 0.69283303, 0.69830382,
        0.682543  , 0.69552678, 0.73020065, 0.73371855, 0.73638826,
        0.68973608, 0.70395579, 0.72911539, 0.72798015, 0.72650304,
        0.69405868, 0.71213216, 0.7199888 , 0.71708364, 0.7147167
9]),
 'mean_test_score': array([0.6553863 , 0.65565399, 0.65738964, 0.65
787687, 0.65790502,
        0.65652587, 0.65784554, 0.66375871, 0.66604062, 0.66803605,
        0.66474569, 0.6690892 , 0.68856358, 0.69538928, 0.70112823,
        0.68587079, 0.69697327, 0.73151621, 0.73489216, 0.73616391,
        0.69234444, 0.70905778, 0.73075645, 0.72973194, 0.72784312,
        0.69560944, 0.71312953, 0.72168475, 0.71907794, 0.7152284
4]),
 'std_test_score': array([0.00312368, 0.00275379, 0.00359963, 0.003
```

```
70949, 0.00362936,
       0.00327057, 0.00350658, 0.00250706, 0.00243712, 0.00205389,
       0.00239511, 0.00298927, 0.00249037, 0.00184466, 0.0019982 ,
       0.00259754, 0.00220667, 0.00101787, 0.00132171, 0.0009739 ,
       0.00184468, 0.00361397, 0.00130043, 0.00125149, 0.00167269,
       0.00142096, 0.00121294, 0.00159098, 0.00357321, 0.0028239
]),
 'rank_test_score': array([30, 29, 27, 25, 24, 28, 26, 23, 21, 20,
22, 19, 17, 15, 12, 18, 13,
        3,  2,  1, 16, 11,  4,  5,  6, 14, 10,  7,  8,  9], dtype=
int32),
 'split0_train_score': array([0.67172544, 0.6722066 , 0.67281705,
0.67317553, 0.67340751,
       0.67251768, 0.67344105, 0.68599513, 0.69131505, 0.69379032,
       0.68903213, 0.69609735, 0.71568426, 0.72730797, 0.7370073 ,
       0.71437993, 0.7346373 , 0.82462624, 0.85817429, 0.8827051 ,
       0.72476881, 0.75767769, 0.87432228, 0.91035958, 0.93692539,
       0.73708893, 0.77540379, 0.90241248, 0.93727818, 0.9600777
2]),
 'split1_train_score': array([0.67737897, 0.67737897, 0.68026755,
0.68118904, 0.68144655,
       0.679652  , 0.68112883, 0.68847206, 0.69059509, 0.69239848,
       0.68762483, 0.69259765, 0.72118855, 0.73014458, 0.73950841,
       0.71450137, 0.73281256, 0.82258926, 0.85450461, 0.87845019,
       0.7283855 , 0.75923095, 0.87184423, 0.90903381, 0.93389598,
       0.73957059, 0.77524531, 0.90049827, 0.93601094, 0.9593671
8]),
 'split2_train_score': array([0.6769817 , 0.6769817 , 0.6792997 ,
0.68104911, 0.68204332,
       0.6769817 , 0.68154216, 0.68869623, 0.69079554, 0.69395027,
       0.68798532, 0.69330018, 0.71861297, 0.72801311, 0.73628122,
       0.71436667, 0.73414769, 0.82416872, 0.85696647, 0.88188746,
       0.72674002, 0.75653894, 0.87270717, 0.90925176, 0.93487184,
       0.73696559, 0.77430497, 0.90178624, 0.93819975, 0.9605587
4]),
 'mean_train_score': array([0.67536204, 0.67552243, 0.67746143, 0.6
7847123, 0.6789658 ,
       0.6763838 , 0.67870401, 0.68772114, 0.6909019 , 0.69337969,
       0.68821409, 0.6939984 , 0.71849526, 0.72848855, 0.73759898,
       0.71441599, 0.73386585, 0.82379474, 0.85654846, 0.88101425,
       0.72663144, 0.75781586, 0.87295789, 0.90954839, 0.93523107,
       0.73787504, 0.77498469, 0.90156566, 0.93716296, 0.9600012
1]),
 'std_train_score': array([2.57657344e-03, 2.35024407e-03, 3.307760
89e-03, 3.74506125e-03,
```

```
        3.93784333e-03, 2.94310027e-03, 3.72530080e-03, 1.22390140e
-03,
        3.03391083e-04, 6.96888852e-04, 5.96867873e-04, 1.51164528e
-03,
        2.24865774e-03, 1.20585424e-03, 1.38233145e-03, 6.06163674e
-05,
        7.71140272e-04, 8.72630051e-04, 1.52702195e-03, 1.84353716e
-03,
        1.47850344e-03, 1.10334409e-03, 1.02707730e-03, 5.80463356e
-04,
        1.26256606e-03, 1.19999338e-03, 4.84967857e-04, 7.96884257e
-04,
        8.97282344e-04, 4.89452343e-04])}
```

In [123]:

```
df_clf = pd.DataFrame(clf.cv_results_)
df_clf.head()
```

Out[123]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	pa
0	31.494727	0.385366	0.081226	0.020973	0.0001	5
1	35.517341	0.244288	0.063756	0.002143	0.0001	1(
2	62.917916	0.497517	0.115683	0.005050	0.0001	5(
3	81.349628	0.932545	0.149754	0.001492	0.0001	7!
4	97.997024	0.521083	0.174885	0.007621	0.0001	1(

In [124]:

```
scores_clf = df_clf.groupby(['param_learning_rate','param_n_estimators']).max().
unstack()[['mean_train_score','mean_test_score']]
```

In [125]:

```
scores_clf
```

Out[125]:

	mean_train_score					mean_test_
param_n_estimators	5	10	50	75	100	5
param_learning_rate						
0.0001	0.675362	0.675522	0.677461	0.678471	0.678966	0.655386
0.0010	0.676384	0.678704	0.687721	0.690902	0.693380	0.656526
0.0100	0.688214	0.693998	0.718495	0.728489	0.737599	0.664746
0.1000	0.714416	0.733866	0.823795	0.856548	0.881014	0.685871
0.2000	0.726631	0.757816	0.872958	0.909548	0.935231	0.692344
0.3000	0.737875	0.774985	0.901566	0.937163	0.960001	0.695609

In [126]:

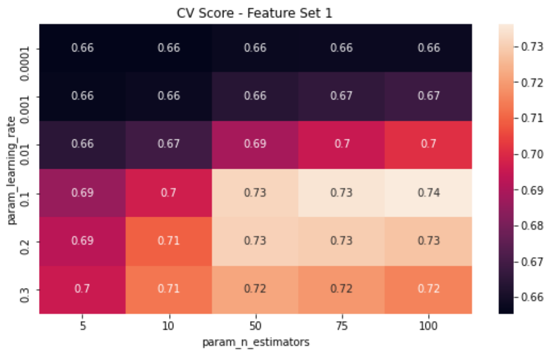
```
train_score_1 = scores_clf['mean_train_score']
test_score_1 = scores_clf['mean_test_score']
```

In [127]:

```
import seaborn as sns
fig,ax = plt.subplots(ncols=2,figsize=(20,5))
sns.heatmap(train_score_1,annot=True,ax=ax[0])
sns.heatmap(test_score_1,annot=True,ax=ax[1])
ax[0].set_title("Train Score - Feature Set 1")
ax[1].set_title("CV Score - Feature Set 1")
```

Out[127]:

```
Text(0.5, 1.0, 'CV Score - Feature Set 1')
```



In [128]:

```
clf.best_params_
```

Out[128]:

```
{'learning_rate': 0.1, 'n_estimators': 100}
```

In [129]:

```
best_model_1 = clf.best_estimator_
best_model_1.fit(X_Train,y_train)
learning_rate_1 = clf.best_params_['learning_rate']
n_estimator_1 = clf.best_params_['n_estimators']
```

In [130]:

```
print(learning_rate_1,n_estimator_1)
```

```
0.1 100
```

In [131]:

```
y_pred_prob_train_1 = best_model_1.predict_proba(X_Train)
y_pred_prob_test_1 = best_model_1.predict_proba(X_Test)
```

In [132]:

```
y_pred_prob_train_1
```

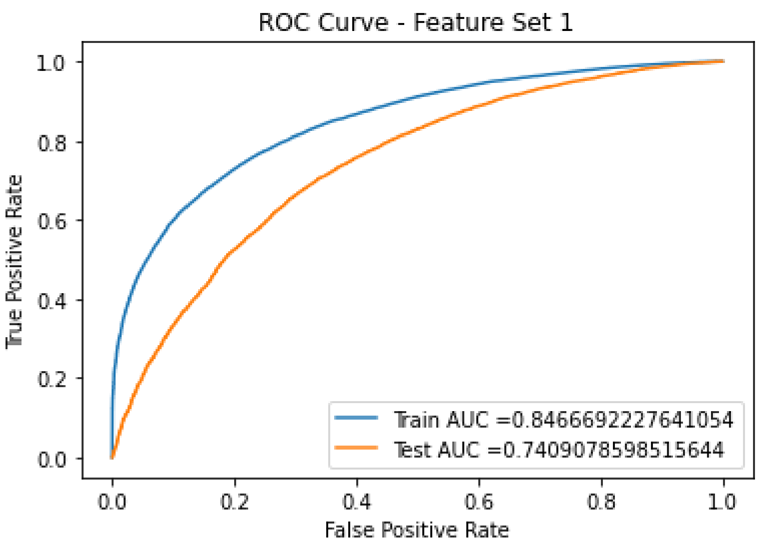
Out[132]:

```
array([[0.37732048, 0.62267952],
       [0.78931092, 0.21068908],
       [0.46393047, 0.53606953],
       ...,
       [0.32749983, 0.67250017],
       [0.46434529, 0.53565471],
       [0.30213778, 0.69786222]])
```

In [133]:

```
from sklearn.metrics import roc_curve,auc

train_fpr,train_tpr,tr_thresholds = roc_curve(y_train,y_pred_prob_train_1[:,1])
test_fpr,test_tpr,te_thresholds = roc_curve(y_test,y_pred_prob_test_1[:,1])
train_auc_1 = auc(train_fpr,train_tpr)
test_auc_1 = auc(test_fpr,test_tpr)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(train_auc_1))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(test_auc_1))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Feature Set 1")
plt.legend()
plt.show()
```



In [134]:

```
from sklearn.metrics import confusion_matrix
y_pred_1 = best_model_1.predict(X_Test)
cm_1 = confusion_matrix(y_test,y_pred_1)
cm_1
```

Out[134]:

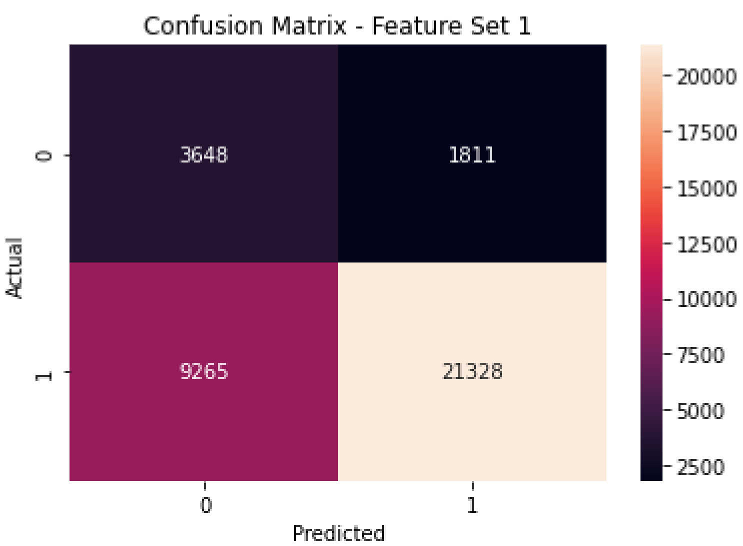
```
array([[ 3648,  1811],
       [ 9265, 21328]])
```

In [135]:

```
sns.heatmap(cm_1,annot=True,fmt="g")
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Feature Set 1')
```

Out[135]:

```
Text(0.5, 1.0, 'Confusion Matrix - Feature Set 1')
```



In [136]:

```
#feature set 2

#loading glove vectors
import pickle
with open('../input/donorschoosewithglovevectors/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [137]:

```
#tfidf w2v

tfidf_model = TfidfVectorizer()
tfidf_model.fit(x_train['essay'])
# 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())

x_train_essays_tfidf_w2v_vectors = []
for essay in tqdm(x_train['essay']):
    vector = np.zeros(300)
    tf_idf_weight =0
    for word in essay.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word]*(essay.count(word)/len(essay.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_train_essays_tfidf_w2v_vectors.append(vector)

x_test_essays_tfidf_w2v_vectors = []
for essay in tqdm(x_test['essay']):
    vector = np.zeros(300)
    tf_idf_weight =0
    for word in essay.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word]*(essay.count(word)/len(essay.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_test_essays_tfidf_w2v_vectors.append(vector)
```

```
100%|██████████| 73196/73196 [03:28<00:00, 350.62it/s]
100%|██████████| 36052/36052 [01:43<00:00, 348.62it/s]
```

In [142]:

```
X_Train_1 = np.hstack((x_train_essays_tfidf_w2v_vectors,
                  xtrain_school_state_0,
                  xtrain_school_state_1,
                  xtrain_teacher_prefix_0,
                  xtrain_teacher_prefix_1,
                  xtrain_project_grade_category_0,
                  xtrain_project_grade_category_1,
                  xtrain_clean_categories_0,
                  xtrain_clean_categories_1,
                  xtrain_clean_subcategories_0,
                  xtrain_clean_subcategories_1,
                  x_train_price_normalized,
                  x_train_teacher_number_of_previously_posted_projects_normalize
d))

X_Test_1 = np.hstack((x_test_essays_tfidf_w2v_vectors,
                  xtest_school_state_0,
                  xtest_school_state_1,
                  xtest_teacher_prefix_0,
                  xtest_teacher_prefix_1,
                  xtest_project_grade_category_0,
                  xtest_project_grade_category_1,
                  xtest_clean_categories_0,
                  xtest_clean_categories_1,
                  xtest_clean_subcategories_0,
                  xtest_clean_subcategories_1,
                  x_test_price_normalized,
                  x_test_teacher_number_of_previously_posted_projects_normalized
))
```

In [143]:

```
print(X_Train_1.shape,y_train.shape)
print(X_Test_1.shape,y_test.shape)
```

```
(73196, 312) (73196,)
(36052, 312) (36052,)
```

In [144]:

```
lgb = LGBMClassifier(class_weight='balanced', n_jobs=-1)
parameters = {'learning_rate':[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3], 'n_estimator
s': [5,10,50, 75, 100]}
clf_1=GridSearchCV(lgb, parameters, cv=3, scoring='roc_auc', verbose = 1,n_jobs=
-1, return_train_score=True)
clf_1.fit(X_Train_1, y_train)
```

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed:  6.8min finis
hed
```

Out[144]:

```
GridSearchCV(cv=3, estimator=LGBMClassifier(class_weight='balance
d'), n_jobs=-1,
             param_grid={'learning_rate': [0.0001, 0.001, 0.01, 0.
1, 0.2, 0.3],
                         'n_estimators': [5, 10, 50, 75, 100]},
             return_train_score=True, scoring='roc_auc', verbose=1)
```

In [146]:

```
df_clf_1 = pd.DataFrame(clf_1.cv_results_)
df_clf_1.head()
```

Out[146]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_learning_rate	paran
0	6.918662	0.028487	0.080974	0.002578	0.0001	5
1	7.925557	0.218153	0.085687	0.001754	0.0001	10
2	19.393532	0.439579	0.145654	0.005544	0.0001	50
3	25.388169	0.425673	0.160092	0.000747	0.0001	75
4	32.175142	0.427131	0.192213	0.006035	0.0001	100

In [147]:

```
scores_clf_1 = df_clf_1.groupby(['param_learning_rate','param_n_estimators']).ma
x().unstack()[['mean_train_score','mean_test_score']]
```

In [148]:

```
scores_clf_1
```

Out[148]:

	mean_train_score					mean_test_sco	
param_n_estimators	5	10	50	75	100	5	1
param_learning_rate							
0.0001	0.669709	0.669709	0.672104	0.673484	0.675828	0.633774	0
0.0010	0.671981	0.675509	0.696093	0.701480	0.706275	0.636060	0
0.0100	0.695838	0.705813	0.734404	0.744621	0.752708	0.655550	0
0.1000	0.721062	0.744280	0.829126	0.865157	0.893657	0.675474	0
0.2000	0.732199	0.762469	0.881112	0.922313	0.950231	0.676731	0
0.3000	0.738821	0.774430	0.910547	0.949863	0.973074	0.681299	0

In [149]:

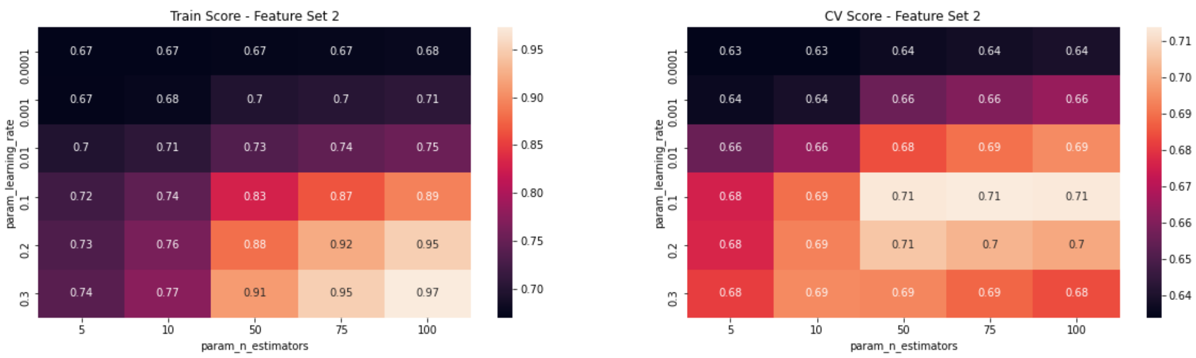
```
train_score_2 = scores_clf_1['mean_train_score']
test_score_2 = scores_clf_1['mean_test_score']
```

In [150]:

```
import seaborn as sns
fig,ax = plt.subplots(ncols=2,figsize=(20,5))
sns.heatmap(train_score_2,annot=True,ax=ax[0])
sns.heatmap(test_score_2,annot=True,ax=ax[1])
ax[0].set_title("Train Score - Feature Set 2")
ax[1].set_title("CV Score - Feature Set 2")
```

Out[150]:

```
Text(0.5, 1.0, 'CV Score - Feature Set 2')
```



In [151]:

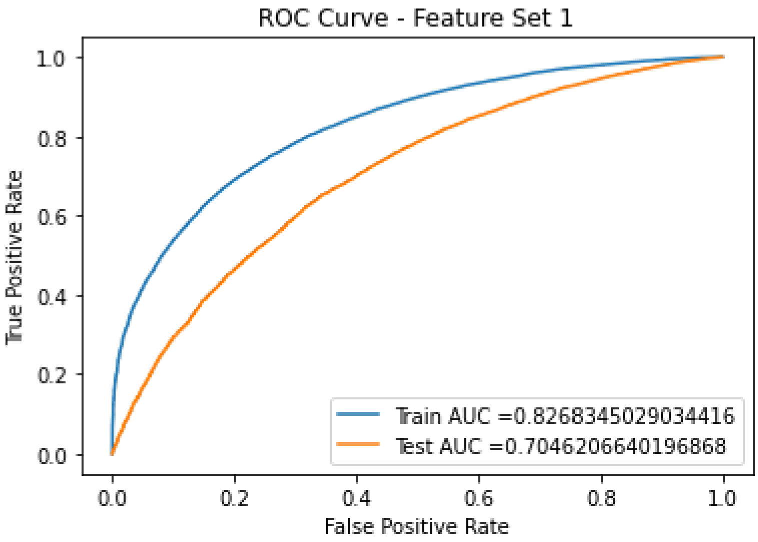
```
best_model_2 = clf_1.best_estimator_
best_model_2.fit(X_Train_1,y_train)
learning_rate_2 = clf.best_params_['learning_rate']
n_estimator_2 = clf.best_params_['n_estimators']
```

In [152]:

```
y_pred_prob_train_2 = best_model_2.predict_proba(X_Train_1)
y_pred_prob_test_2 = best_model_2.predict_proba(X_Test_1)
```

In [153]:

```
train_fpr,train_tpr,tr_thresholds = roc_curve(y_train,y_pred_prob_train_2[:,1])
test_fpr,test_tpr,te_thresholds = roc_curve(y_test,y_pred_prob_test_2[:,1])
train_auc_2 = auc(train_fpr,train_tpr)
test_auc_2 = auc(test_fpr,test_tpr)
plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(train_auc_2))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(test_auc_2))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Feature Set 1")
plt.legend()
plt.show()
```



In [155]:

```
y_pred_2 = best_model_2.predict(X_Test_1)
cm_2 = confusion_matrix(y_test,y_pred_2)
cm_2
```

Out[155]:

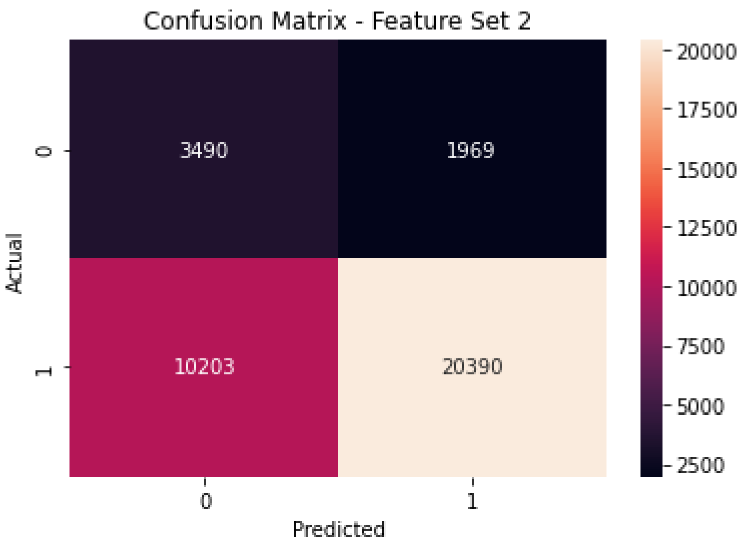
```
array([[ 3490,  1969],
       [10203, 20390]])
```

In [156]:

```
sns.heatmap(cm_2,annot=True,fmt="g")
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Feature Set 2')
```

Out[156]:

```
Text(0.5, 1.0, 'Confusion Matrix - Feature Set 2')
```



In [157]:

```
# please write all the code with proper documentation, and proper titles for each
 subsection
# 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 code
# 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
```

3. Summary

as mentioned in the step 4 of instructions

In [158]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer","Model","learning rate","n estimators","Train AUC"
,"Test AUC"]

x.add_row(["TF-IDF","LightGBM", learning_rate_1,n_estimator_1,train_auc_1,test_a
uc_1])
x.add_row(["TF-IDF W2V","LightGBM", learning_rate_2,n_estimator_2,train_auc_2,te
st_auc_2])



print(x)
```

```
+------------+----------+---------------+--------------+-----------
---------+--------------------+
| Vectorizer |  Model   | learning rate | n estimators |     Train
AUC        |      Test AUC      |
+------------+----------+---------------+--------------+-----------
---------+--------------------+
|   TF-IDF   | LightGBM |      0.1      |     100      | 0.84666922
27641054 | 0.7409078598515644 |
| TF-IDF W2V | LightGBM |      0.1      |     100      | 0.82683450
29034416 | 0.7046206640196868 |
+------------+----------+---------------+--------------+-----------
---------+--------------------+
```

In [ ]:

In [ ]: