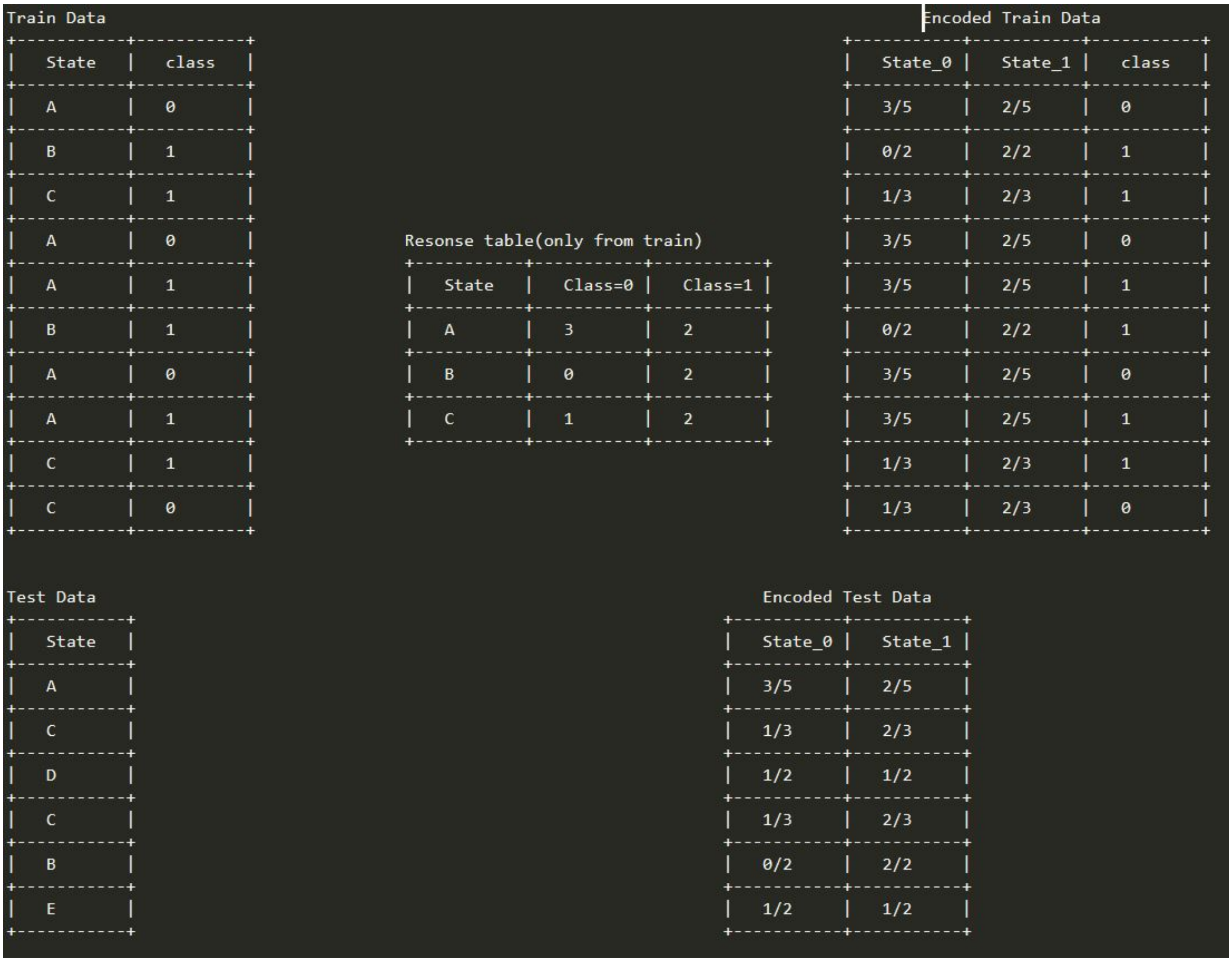
Assignment 9: GBDT

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]**

In [4]:

```
from google.colab import drive
drive.mount('/gdrive',force_remount=True)
```

```
Mounted at /gdrive
```

In [5]:

```
#please use below code to load glove vectors
import pickle
with open('/gdrive/MyDrive/9_Donors_choose_DT/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [6]:

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

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

```
Out[6]:

True
```

1. **Apply GBDT on these feature sets**

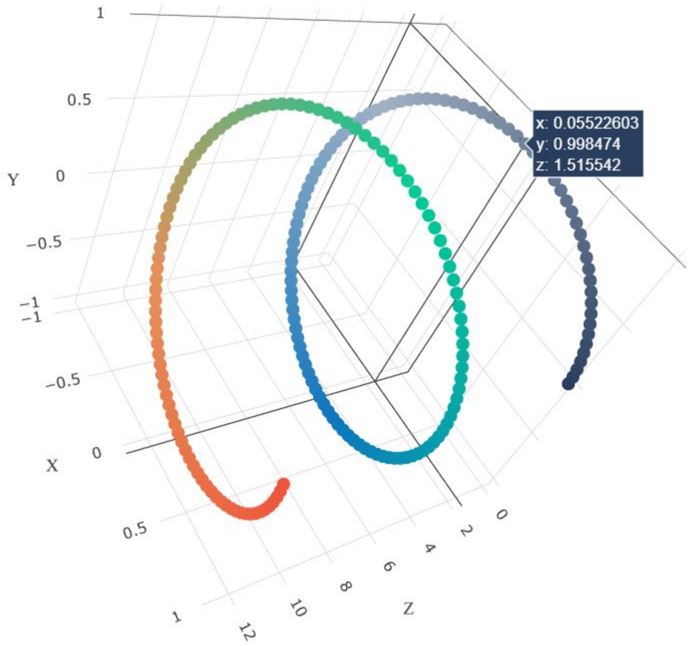
   - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check the bellow example, include all 4 values as 4 features)
   - Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

2. **The hyper paramter tuning (Consider any two hyper parameters)**

   - Find the best hyper parameter which will give the maximum AUC value
   - find the best hyper paramter using k-fold cross validation/simple cross validation data
   - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task

3. **Representation of results**

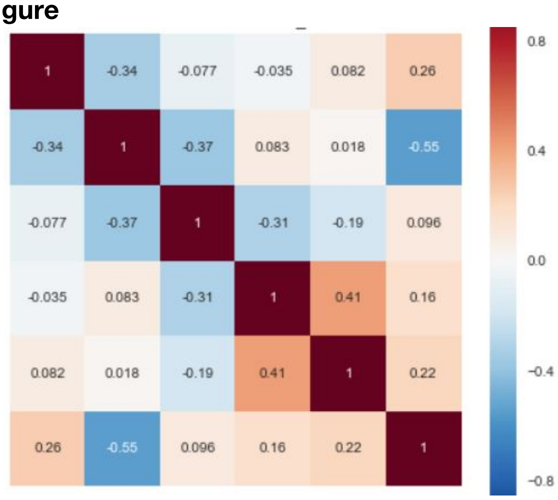
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



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

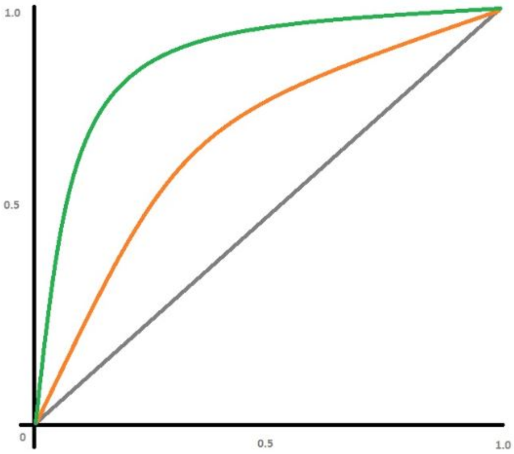
or

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

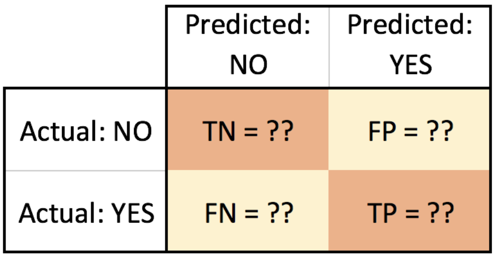


seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**

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

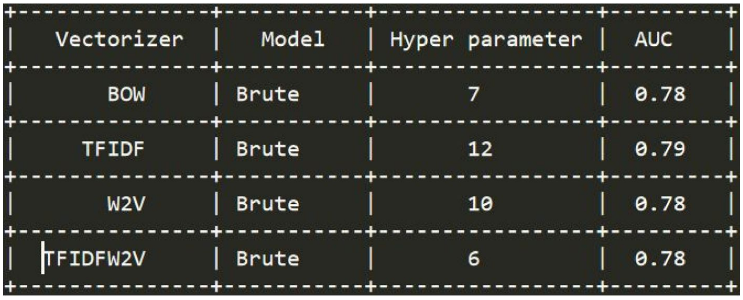


- Along with plotting ROC curve, you need to print the [confusion matrix]() with predicted and original labels of test data points



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

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



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

In [7]:

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

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

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest s
tudents with the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multipl
e intelligences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of
different backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school i
s a caring community of successful \
learners which can be seen through collaborative student project based learning in and ou
t of the classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities to
practice a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect
of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to
role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with real food i will take their idea \
and create common core cooking lessons where we learn important math and writing concepts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cookbooks to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoyment for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

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

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

```
0.01
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

```
/usr/local/lib/python3.7/dist-packages/nltk/twitter/__init__.py:20: UserWarning: The twython library has not been installed. Some functionality from the twitter package will not be available.
  warnings.warn("The twython library has not been installed. "
```

1. GBDT (xgboost/lightgbm)

1.1 Loading Data

In [8]:

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

## To read data
import pandas as pd
import numpy as np
import pickle
from tqdm import tqdm
import os

## Data preprocessing
import nltk
import re

## EDA
import matplotlib.pyplot as plt
import seaborn as sns

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

## Feature Vectorization
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

## Model Performance
```

```
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

data = pd.read_csv('/gdrive/MyDrive/9_Donors_choose_DT/preprocessed_70K.csv',nrows = 5000
0)
```

1.2 Splitting data into Train and test: Stratified Sampling

In [9]:

```
Y = data['project_is_approved'].values
X = data.drop(columns=['project_is_approved','Unnamed: 0','number_in_summary'], axis=1)
print (X.shape)
print (Y.shape)
```

```
(50000, 9)
(50000,)
```

In [10]:

```
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,random
_state = 100)
```

In [11]:

```
print("Train : ",X_train.shape, Y_train.shape)
print("Test  : ",X_test.shape, Y_test.shape)

print("="*100)
```

```
Train :  (33500, 9) (33500,)
Test  :  (16500, 9) (16500,)
==========================================================================================
==========
```

In [15]:

```
np.unique(Y_train, return_counts=True)
```

Out[15]:

```
(array([0, 1]), array([ 5168, 28332]))
```

1.3 Make Data Model Ready: Encoding text features

1) Essay (TFIDF)

In [43]:

```
import time
vectorizer = TfidfVectorizer(min_df = 20,ngram_range=(1,4), max_features=5000)
start = time.time()
vectorizer.fit(X_train['essay'].values)
end = time.time()
print (end-start,'s')
```

```
100.582852602005 s
```

In [44]:

```
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("After TFIDF vectorizations\n")
print("Train Essay : ",X_train_essay_tfidf.shape, Y_train.shape)
print("Test Essay : ",X_test_essay_tfidf.shape, Y_test.shape)
print("="*100)
```

```
After TFIDF vectorizations

Train Essay :  (33500, 5000) (33500,)
Test Essay :  (16500, 5000) (16500,)
=========================================================================================
===========
```

2) Project Title(TFIDF)

In [45]:

```
vectorizer = TfidfVectorizer(min_df = 10,ngram_range=(1,4), max_features=5000)
start = time.time()
vectorizer.fit(X_train['project_title'].values)
end = time.time()
print (end-start,'s')
```

```
1.3070762157440186 s
```

In [46]:

```
X_train_project_title_tfidf = vectorizer.transform(X_train['project_title'].values)
X_test_project_title_tfidf = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations\n")
print("Train Project Title : ",X_train_project_title_tfidf.shape, Y_train.shape)
print("Test Project Title : ",X_test_project_title_tfidf.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations

Train Project Title :  (33500, 2434) (33500,)
Test Project Title :  (16500, 2434) (16500,)
=========================================================================================
===========
```

1.4 Make Data Model Ready: Encoding Categorical features Using Response Coding

2) school_state

In [12]:

```
feature_list = X_train['school_state']
df = pd.DataFrame({'school_state':feature_list,'class':Y_train})
df_temp = df.groupby(["school_state", "class"])['school_state'].count().reset_index(name
="count")
df_temp = df_temp.pivot_table(index = 'school_state',columns = 'class',values = 'count')
df_temp.reset_index(inplace=True)
df_temp = df_temp.replace(np.nan, 0)

df_temp['state_0'] = df_temp[0]/(df_temp[0] + df_temp[1])
df_temp['state_1'] = df_temp[1]/(df_temp[0] + df_temp[1])

df_temp = df_temp[['school_state','state_0','state_1']]

X_train = pd.merge(X_train, df_temp, how="left",on = 'school_state')
X_test  = pd.merge(X_test,  df_temp, how="left",on = 'school_state')

X_test['state_0'] = X_test['state_0'].replace(np.nan,0.5)
X_test['state_1'] = X_test['state_1'].replace(np.nan,0.5)
```

```
print (X_train['state_0'].isnull().sum())
print (X_train['state_1'].isnull().sum())
print (X_test['state_0'].isnull().sum())
print (X_test['state_1'].isnull().sum())

X_train_state_0 = X_train['state_0'].values.reshape(-1,1)
X_train_state_1 = X_train['state_1'].values.reshape(-1,1)
X_test_state_0 = X_test['state_0'].values.reshape(-1,1)
X_test_state_1 = X_test['state_1'].values.reshape(-1,1)

print(" Response Coding for School State done")
print("="*100)
```

```
0
0
0
0
 Response Coding for School State done
===========================================================================================
===========
```

3) teacher_prefix

In [13]:

```
feature_list = X_train['teacher_prefix']
df = pd.DataFrame()
df_temp = pd.DataFrame()
df = pd.DataFrame({'teacher_prefix':feature_list,'class':Y_train})
df_temp = df.groupby(["teacher_prefix", "class"])['teacher_prefix'].count().reset_index(
name="count")
df_temp = df_temp.pivot_table(index = 'teacher_prefix',columns = 'class',values = 'count
')
df_temp.reset_index(inplace=True)
df_temp = df_temp.replace(np.nan, 0)
df_temp['prefix_0'] = df_temp[0]/(df_temp[0] + df_temp[1])
df_temp['prefix_1'] = df_temp[1]/(df_temp[0] + df_temp[1])
df_temp = df_temp[['teacher_prefix','prefix_0','prefix_1']]

X_train = pd.merge(X_train, df_temp, how="left",on = 'teacher_prefix')
X_test  = pd.merge(X_test,  df_temp, how="left",on = 'teacher_prefix')

X_test['prefix_0'] = X_test['prefix_0'].replace(np.nan,0.5)
X_test['prefix_1'] = X_test['prefix_1'].replace(np.nan,0.5)

print (X_train['prefix_0'].isnull().sum())
print (X_train['prefix_1'].isnull().sum())
print (X_test['prefix_0'].isnull().sum())
print (X_test['prefix_1'].isnull().sum())


X_train_prefix_0 = X_train['prefix_0'].values.reshape(-1,1)
X_train_prefix_1 = X_train['prefix_1'].values.reshape(-1,1)
X_test_prefix_0 = X_test['prefix_0'].values.reshape(-1,1)
X_test_prefix_1 = X_test['prefix_1'].values.reshape(-1,1)

print(" Response Coding done")
print("="*100)
```

```
0
0
0
0
 Response Coding done
===========================================================================================
===========
```

4) project_grade_category

In [14]:

```
feature_list = X_train['project_grade_category']
df = pd.DataFrame()
df_temp = pd.DataFrame()
df = pd.DataFrame({'project_grade_category':feature_list,'class':Y_train})
df_temp = df.groupby(["project_grade_category", "class"])['project_grade_category'].count().reset_index(name="count")
df_temp = df_temp.pivot_table(index = 'project_grade_category',columns = 'class',values = 'count')
df_temp.reset_index(inplace=True)
df_temp = df_temp.replace(np.nan, 0)
df_temp['grade_0'] = df_temp[0]/(df_temp[0] + df_temp[1])
df_temp['grade_1'] = df_temp[1]/(df_temp[0] + df_temp[1])
df_temp = df_temp[['project_grade_category','grade_0','grade_1']]

X_train = pd.merge(X_train, df_temp, how="left",on = 'project_grade_category')
X_test  = pd.merge(X_test,  df_temp, how="left",on = 'project_grade_category')

X_test['grade_0'] = X_test['grade_0'].replace(np.nan,0.5)
X_test['grade_1'] = X_test['grade_1'].replace(np.nan,0.5)

print (X_train['grade_0'].isnull().sum())
print (X_train['grade_1'].isnull().sum())
print (X_test['grade_0'].isnull().sum())
print (X_test['grade_1'].isnull().sum())

X_train_grade_0 = X_train['grade_0'].values.reshape(-1,1)
X_train_grade_1 = X_train['grade_1'].values.reshape(-1,1)
X_test_grade_0 = X_test['grade_0'].values.reshape(-1,1)
X_test_grade_1 = X_test['grade_1'].values.reshape(-1,1)

print(" Response Coding done")
print("="*100)
```

```
0
0
0
0
 Response Coding done
====================================================================================================
```

5) clean_categories

In [15]:

```
feature_list = X_train['clean_categories']
df = pd.DataFrame()
df_temp = pd.DataFrame()
df = pd.DataFrame({'clean_categories':feature_list,'class':Y_train})
df_temp = df.groupby(["clean_categories", "class"])['clean_categories'].count().reset_index(name="count")
df_temp = df_temp.pivot_table(index = 'clean_categories',columns = 'class',values = 'count')
df_temp.reset_index(inplace=True)
df_temp = df_temp.replace(np.nan, 0)
df_temp['category_0'] = df_temp[0]/(df_temp[0] + df_temp[1])
df_temp['category_1'] = df_temp[1]/(df_temp[0] + df_temp[1])
df_temp = df_temp[['clean_categories','category_0','category_1']]

X_train = pd.merge(X_train, df_temp, how="left",on = 'clean_categories')
X_test  = pd.merge(X_test,  df_temp, how="left",on = 'clean_categories')

X_test['category_0'] = X_test['category_0'].replace(np.nan,0.5)
X_test['category_1'] = X_test['category_1'].replace(np.nan,0.5)

print (X_train['category_0'].isnull().sum())
print (X_train['category_1'].isnull().sum())
print (X_test['category_0'].isnull().sum())
print (X_test['category_1'].isnull().sum())
```

```
X_train_category_0 = X_train['category_0'].values.reshape(-1,1)
X_train_category_1 = X_train['category_1'].values.reshape(-1,1)
X_test_category_0 = X_test['category_0'].values.reshape(-1,1)
X_test_category_1 = X_test['category_1'].values.reshape(-1,1)

print(" Response Coding done")
print("="*100)
```

```
0
0
0
0
 Response Coding done
=========================================================================================
===========
```

6) clean_subcategories

In [16]:

```
feature_list = X_train['clean_subcategories']

df = pd.DataFrame({'clean_subcategories':feature_list,'class':Y_train})
df_temp = df.groupby(["clean_subcategories", "class"])['clean_subcategories'].count().re
set_index(name="count")
df_temp = df_temp.pivot_table(index = 'clean_subcategories',columns = 'class',values = '
count')
df_temp.reset_index(inplace=True)
df_temp = df_temp.replace(np.nan, 0)
df_temp['subcategory_0'] = df_temp[0]/(df_temp[0] + df_temp[1])
df_temp['subcategory_1'] = df_temp[1]/(df_temp[0] + df_temp[1])
df_temp = df_temp[['clean_subcategories','subcategory_0','subcategory_1']]

X_train = pd.merge(X_train, df_temp, how="left",on = 'clean_subcategories')
X_test  = pd.merge(X_test,  df_temp, how="left",on = 'clean_subcategories')

X_test['subcategory_0'] = X_test['subcategory_0'].replace(np.nan,0.5)
X_test['subcategory_1'] = X_test['subcategory_1'].replace(np.nan,0.5)

print (X_train['subcategory_0'].isnull().sum())
print (X_train['subcategory_1'].isnull().sum())
print (X_test['subcategory_0'].isnull().sum())
print (X_test['subcategory_1'].isnull().sum())

X_train_subcategory_0 = X_train['subcategory_0'].values.reshape(-1,1)
X_train_subcategory_1 = X_train['subcategory_1'].values.reshape(-1,1)
X_test_subcategory_0 = X_test['subcategory_0'].values.reshape(-1,1)
X_test_subcategory_1 = X_test['subcategory_1'].values.reshape(-1,1)

print(" Response Coding done")
print("="*100)
```

```
0
0
0
0
 Response Coding done
=========================================================================================
===========
```

1.5 Make Data Model Ready: Encoding Numerical features

7) price

In [17]:

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

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


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


print("After vectorizations\n")
print("Train Price : ",X_train_price_norm.shape, Y_train.shape)
print("Test Price : ",X_test_price_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations

Train Price :  (33500, 1) (33500,)
Test Price :  (16500, 1) (16500,)
=========================================================================================
===========
```

8) teacher_number_of_previously_posted_projects

In [18]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1
,1))

X_train_previous_norm = normalizer.transform(X_train['teacher_number_of_previously_posted
_projects'].values.reshape(-1,1))
X_test_previous_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_p
rojects'].values.reshape(-1,1))


print("After vectorizations\n")
print("Train Previous Norm : ",X_train_previous_norm.shape, Y_train.shape)
print("Test Previous Norm : ",X_test_previous_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations

Train Previous Norm :  (33500, 1) (33500,)
Test Previous Norm :  (16500, 1) (16500,)
=========================================================================================
===========
```

9) Sentiment scores(preprocessed_essay)

In [19]:

```
sid = SentimentIntensityAnalyzer()

train_essays = X_train['essay'].values
X_train_negative = []
X_train_neutral = []
X_train_positive = []
X_train_compound = []

for i in tqdm(train_essays):
    ss = sid.polarity_scores(i)
    X_train_negative.append(ss['neg'])
    X_train_neutral.append(ss['neu'])
    X_train_positive.append(ss['pos'])
    X_train_compound.append(ss['compound'])

test_essays = X_test['essay'].values
X_test_negative = []
X_test_neutral = []
```

```
X_test_positive = []
X_test_compound = []

for i in tqdm(test_essays):
    ss = sid.polarity_scores(i)
    X_test_negative.append(ss['neg'])
    X_test_neutral.append(ss['neu'])
    X_test_positive.append(ss['pos'])
    X_test_compound.append(ss['compound'])

X_train_negative = np.array(X_train_negative).reshape(-1,1)
X_train_neutral = np.array(X_train_neutral).reshape(-1,1)
X_train_positive = np.array(X_train_positive).reshape(-1,1)
X_train_compound = np.array(X_train_compound).reshape(-1,1)

X_test_negative = np.array(X_test_negative).reshape(-1,1)
X_test_neutral = np.array(X_test_neutral).reshape(-1,1)
X_test_positive = np.array(X_test_positive).reshape(-1,1)
X_test_compound = np.array(X_test_compound).reshape(-1,1)
```

```
100%|██████████| 33500/33500 [01:18<00:00, 429.26it/s]
100%|██████████| 16500/16500 [00:38<00:00, 431.31it/s]
```

Set 1: categorical(response coding), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay

In [47]:

```
from scipy.sparse import hstack

X_tr_1 = hstack(( X_train_essay_tfidf , X_train_project_title_tfidf , X_train_state_0 ,
X_train_state_1 , X_train_prefix_0 , X_train_prefix_1 , X_train_grade_0 , X_train_grade_
1 , X_train_category_0 , X_train_category_1 , X_train_subcategory_0 , X_train_subcategory
_1 , X_train_price_norm , X_train_previous_norm , X_train_negative , X_train_neutral , X
_train_positive , X_train_compound)).tocsr()
X_te_1 = hstack(( X_test_essay_tfidf  , X_test_project_title_tfidf  , X_test_state_0  ,
X_test_state_1  , X_test_prefix_0  , X_test_prefix_1  , X_test_grade_0  , X_test_grade_1
,  X_test_category_0 , X_test_category_1  , X_test_subcategory_0  , X_test_subcategory_1
, X_test_price_norm  , X_test_previous_norm  , X_test_negative  , X_test_neutral  , X_te
st_positive  , X_test_compound )).tocsr()

print("Final Data matrix")
print(X_tr_1.shape, Y_train.shape)
print(X_te_1.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 7450) (33500,)
(16500, 7450) (16500,)
===========================================================================================
===========
```

1.6 Applying GBDT:XGBoost on Set 1 using RandomizedSearch CV

In [20]:

```
# !pip install xgboost
import xgboost
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score
import multiprocessing
import time
```

```
multiprocessing.cpu_count()
from sklearn.ensemble import GradientBoostingClassifier
```

In [48]:

```
# gbdt_model = GradientBoostingClassifier()
# parameters = {'max_depth':[2,4,6,8],'n_estimators' : [50,100,150,200]}
# clf_gbdt = RandomizedSearchCV(gbdt_model, parameters,n_jobs = -1,cv = 5, scoring='roc_a
uc',return_train_score = True)

xgb_model = XGBClassifier(colsample_bytree = 0.5,subsample = 0.5,scale_pos_weight=0.18)
parameters = {'max_depth':[2,4,6,8],'n_estimators' : [100,150,200,250]}
clf_xgb = RandomizedSearchCV(xgb_model, parameters,n_jobs = -1,cv = 5, scoring='roc_auc'
,return_train_score = True)
```

In [49]:

```
start = time.time()
clf_xgb.fit(X_tr_1, Y_train)
end = time.time()
print (end-start,'s')
```

```
1186.462511062622 s
```

In [50]:

```
results1 = pd.DataFrame.from_dict(clf_xgb.cv_results_)[['param_n_estimators', 'param_max
_depth', 'params','mean_test_score','mean_train_score','rank_test_score']].sort_values([
'rank_test_score'])
results1.head()
```

Out[50]:

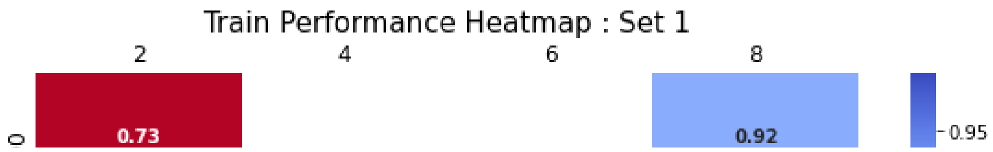
	param_n_estimators	param_max_depth	params	mean_test_score	mean_train_score	rank_test_score
6	250	2	{'n_estimators': 250, 'max_depth': 2}	0.705491	0.782301	1
7	150	4	{'n_estimators': 150, 'max_depth': 4}	0.702880	0.838234	2
4	200	2	{'n_estimators': 200, 'max_depth': 2}	0.702607	0.768865	3
8	150	6	{'n_estimators': 150, 'max_depth': 6}	0.699121	0.903714	4
0	200	6	{'n_estimators': 200, 'max_depth': 6}	0.698681	0.926431	5

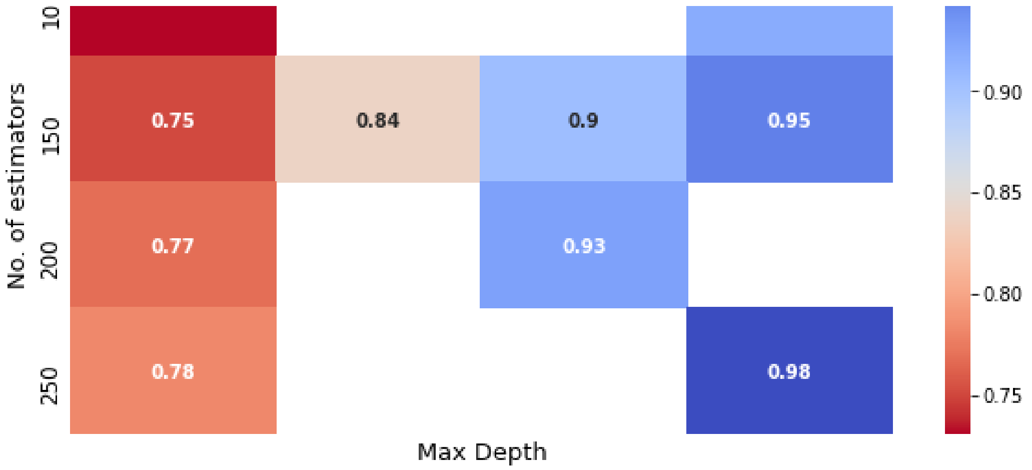
In [52]:

```
plt.figure(figsize=(10, 5))
heat_data = results1.pivot(index='param_n_estimators', columns='param_max_depth', values
='mean_train_score')
vmin = results1['mean_train_score'].min()
vmax = results1['mean_train_score'].max()
heatmap = sns.heatmap(heat_data, vmin, vmax, annot=True,annot_kws={"fontsize":10,"weight
": "bold"}, cmap='coolwarm_r')
heatmap.tick_params(axis='both', which='major', labelsize=12, labelbottom = False, botto
m=False, top = False, labeltop=True,length = 0)
heatmap.set_title('Train Performance Heatmap : Set 1',fontsize = 15)
plt.xlabel('Max Depth',fontsize = 13)
plt.ylabel('No. of estimators',fontsize = 13)
```

Out[52]:

```
Text(69.0, 0.5, 'No. of estimators')
```



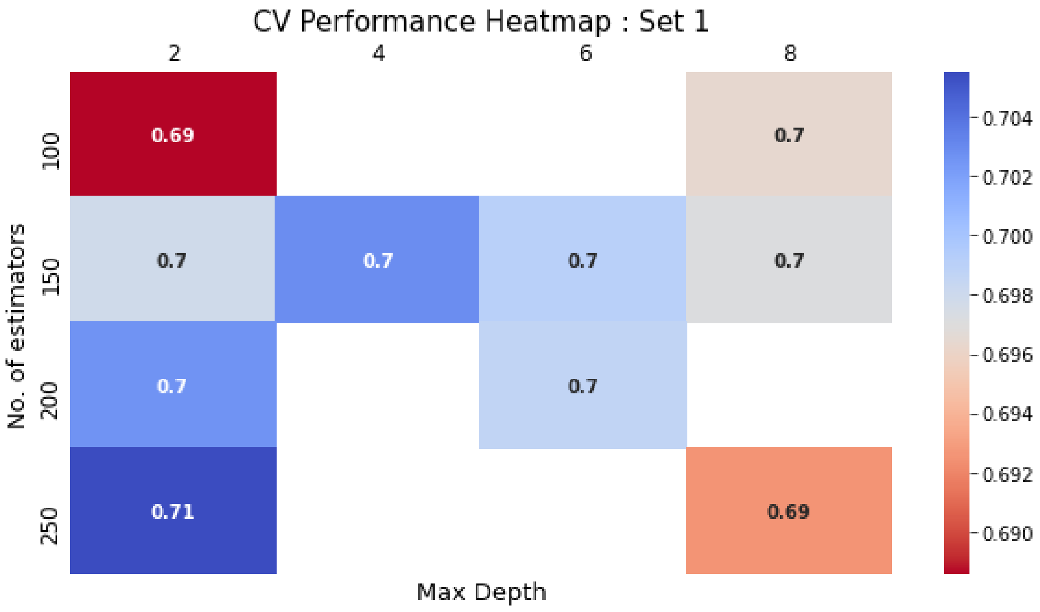


In [53]:

```
plt.figure(figsize=(10, 5))
heat_data = results1.pivot(index='param_n_estimators', columns='param_max_depth', values='mean_test_score')
vmin = results1['mean_test_score'].min()
vmax = results1['mean_test_score'].max()
heatmap = sns.heatmap(heat_data, vmin, vmax, annot=True,annot_kws={"fontsize":10,"weight": "bold"}, cmap='coolwarm_r')
heatmap.tick_params(axis='both', which='major', labelsize=12, labelbottom = False, bottom=False, top = False, labeltop=True,length = 0)
heatmap.set_title('CV Performance Heatmap : Set 1',fontsize = 15)
plt.xlabel('Max Depth',fontsize = 13)
plt.ylabel('No. of estimators',fontsize = 13)
```

Out[53]:

```
Text(69.0, 0.5, 'No. of estimators')
```



In [54]:

```
best_depth1 =  results1[results1['rank_test_score'] == 1]['param_max_depth'].values[0]
best_n_estimators1 = results1[results1['rank_test_score'] == 1]['param_n_estimators'].values[0]
```

In [55]:

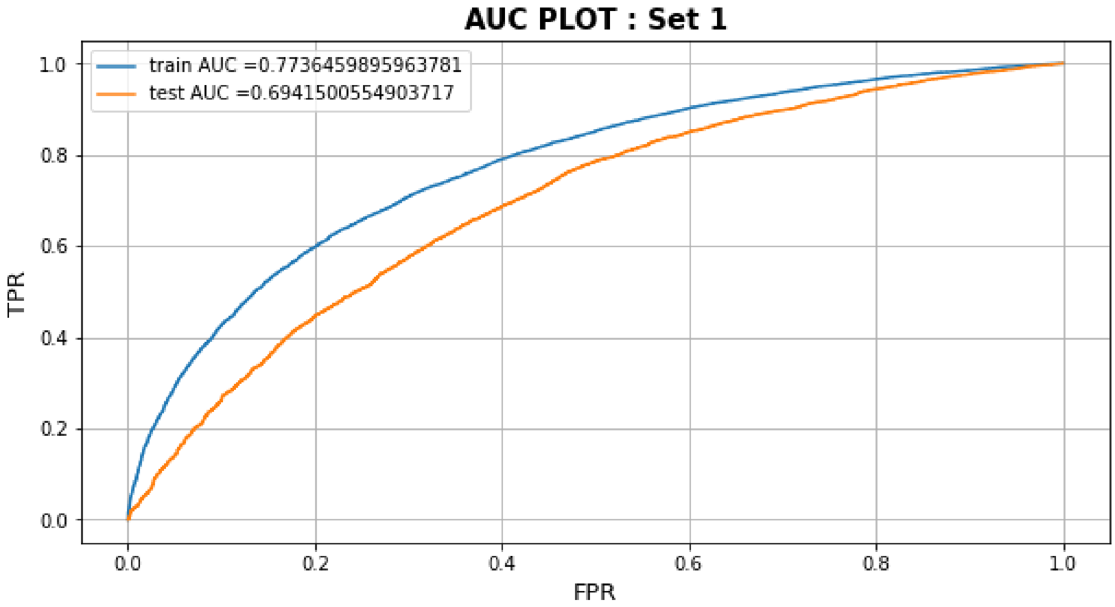
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class not the predicted outputs
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # 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 [56]:

```
from sklearn.metrics import roc_curve, auc

xgb_model_optimal1 = XGBClassifier(colsample_bytree = 0.5,subsample = 0.5,max_depth = bes
t_depth1,n_estimators = best_n_estimators1,scale_pos_weight=0.18)
xgb_model_optimal1.fit(X_tr_1, Y_train)

Y_train_pred1 = batch_predict(xgb_model_optimal1, X_tr_1)
Y_test_pred1 = batch_predict(xgb_model_optimal1, X_te_1)
```

In [57]:

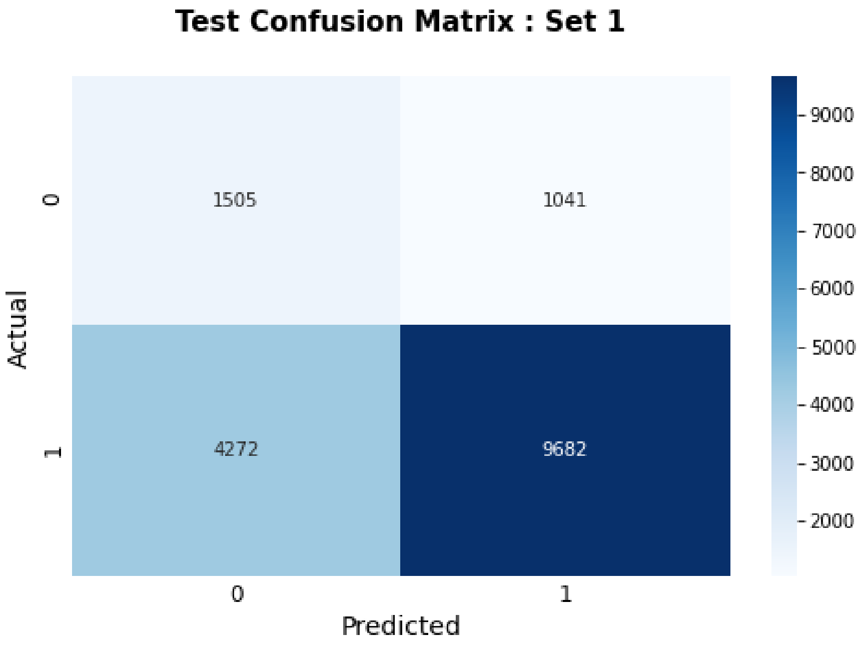
```
plt.figure(figsize=(10, 5))
train_fpr1, train_tpr1,tr_thresholds1 = roc_curve(Y_train, Y_train_pred1)
test_fpr1, test_tpr1,te_thresholds1 = roc_curve(Y_test, Y_test_pred1)

AUC1 = auc(test_fpr1, test_tpr1)
plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))
plt.legend()
plt.xlabel("FPR",fontsize = 13)
plt.ylabel("TPR",fontsize = 13)
plt.title("AUC PLOT : Set 1",fontsize = 15,weight = "bold")
plt.grid()
plt.show()
```



In [58]:

```
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[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))
    print ('\n')
    return t

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

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

best_t1 = find_best_threshold(tr_thresholds1, train_fpr1, train_tpr1)

print("Test confusion matrix : Set 1\n")
print (confusion_matrix(Y_test, predict_with_best_t(Y_test_pred1, best_t1)))
tn1, fp1, fn1, tp1 = confusion_matrix(Y_test, predict_with_best_t(Y_test_pred1, best_t1)
).ravel()
print ("\nTrue Negative  : ",tn1)
print ("False Positive : ",fp1)
print ("False Negative : ",fn1)
print ("True Positive  : ",tp1)
print ("\n")
test_cm = np.array([[tn1,fp1 ],[fn1, tp1 ]])

plt.figure(figsize=(8, 5))
cm = sns.heatmap(test_cm, annot=True,fmt="d",cmap='Blues')
cm.tick_params(axis='both', which='major', labelsize=12, labelbottom = True, bottom=Fals
e, top = False, labeltop=False,length = 0)
cm.set_title('Test Confusion Matrix : Set 1\n',fontsize = 15,weight = 'bold')
plt.xlabel('Predicted',fontsize = 14)
plt.ylabel('Actual',fontsize = 14)
plt.show()
```

```
The maximum value of tpr*(1-fpr) 0.4963436906227749 for threshold 0.484


Test confusion matrix : Set 1

[[1505 1041]
 [4272 9682]]

True Negative  :  1505
False Positive :  1041
False Negative :  4272
True Positive  :  9682
```



Set 2: categorical(response coding), numerical features + project_title(TFIDFW2V)+ preprocessed_eassay (TFIDFW2V)+sentiment Score of eassay

a. Essay(TFIDFW2V)

In [21]:

```
vectorizer = TfidfVectorizer(min_df = 20,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'].values)

dictionary = dict(zip(vectorizer.get_feature_names(), list(vectorizer.idf_)))
tfidf_words = set(vectorizer.get_feature_names())
```

In [22]:

```
#compute average word2vec for each essay.
X_train_essay_tfidf_w2v_vectors = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each essay
    vector = np.zeros(300)
    tf_idf_weight =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word

            # multiplying idf value(dictionary[word]) and the tf value((sentence.count(wo
rd)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # get
ting 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
    X_train_essay_tfidf_w2v_vectors.append(vector)

print(len(X_train_essay_tfidf_w2v_vectors[0]))
print(len(X_train_essay_tfidf_w2v_vectors))
```

```
100%|██████████| 33500/33500 [01:05<00:00, 508.25it/s]
```

```
300
33500
```

In [23]:

```
# calculating tfidf weighted w2v of X_test['essay']
X_test_essay_tfidf_w2v_vectors = [];
for sentence in tqdm(X_test['essay'].values):
    vector = np.zeros(300)
    tf_idf_weight =0;
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_essay_tfidf_w2v_vectors.append(vector)

print(len(X_test_essay_tfidf_w2v_vectors[0]))
print(len(X_test_essay_tfidf_w2v_vectors))
```

```
100%|██████████| 16500/16500 [00:34<00:00, 475.04it/s]
```

```
300
16500
```

b. Project Title(TFIDFW2V)

In [24]:

```
vectorizer = TfidfVectorizer(min_df = 20,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_title'].values)

dictionary = dict(zip(vectorizer.get_feature_names(), list(vectorizer.idf_)))
tfidf_words = set(vectorizer.get_feature_names())
```

In [25]:

```
#compute average word2vec for each essay.
X_train_project_title_tfidf_w2v_vectors = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(X_train['project_title'].values): # for each essay
    vector = np.zeros(300)
    tf_idf_weight =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word

            # multiplying idf value(dictionary[word]) and the tf value((sentence.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
    X_train_project_title_tfidf_w2v_vectors.append(vector)

# X_train_project_title_tfidf_w2v_vectors = np.array(X_train_project_title_tfidf_w2v_vectors)
```

```
100%|██████████| 33500/33500 [00:01<00:00, 28157.27it/s]
```

In [26]:

```
# calculating tfidf weighted w2v of X_test['essay']
X_test_project_title_tfidf_w2v_vectors = [];
for sentence in tqdm(X_test['project_title'].values):
    vector = np.zeros(300)
    tf_idf_weight =0;
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word]
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector += (vec * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_project_title_tfidf_w2v_vectors.append(vector)

# X_test_project_title_tfidf_w2v_vectors = np.array(X_test_project_title_tfidf_w2v_vectors)
```

```
100%|██████████| 16500/16500 [00:00<00:00, 27804.68it/s]
```

In [27]:

```
from scipy.sparse import hstack
from scipy.sparse import csr_matrix

X_tr_2 = hstack(( csr_matrix(X_train_essay_tfidf_w2v_vectors) , X_train_project_title_tfidf_w2v_vectors, X_train_state_0 , X_train_state_1 , X_train_prefix_0 , X_train_prefix_1 , X_train_grade_0 , X_train_grade_1 , X_train_category_0 , X_train_category_1 , X_train_subcategory_0 , X_train_subcategory_1 , X_train_price_norm , X_train_previous_norm , X_train_negative , X_train_neutral , X_train_positive , X_train_compound)).tocsr()
X_te_2 = hstack(( csr_matrix(X_test_essay_tfidf_w2v_vectors)  , X_test_project_title_tfidf_w2v_vectors  , X_test_state_0  , X_test_state_1  , X_test_prefix_0  , X_test_prefix_1 , X_test_grade_0  , X_test_grade_1  ,  X_test_category_0 , X_test_category_1  , X_test_subcategory_0  , X_test_subcategory_1  , X_test_price_norm  , X_test_previous_norm  , X_test_negative  , X_test_neutral  , X_test_positive  , X_test_compound )).tocsr()
```

```
print("Final Data matrix")
print(X_tr_2.shape, Y_train.shape)
print(X_te_2.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 616) (33500,)
(16500, 616) (16500,)
=========================================================================================
===========
```

In [28]:

```
xgb_model2 = XGBClassifier(colsample_bytree = 0.5,subsample = 0.5,scale_pos_weight=0.18)
parameters = {'max_depth':[2,4,6,8],'n_estimators' : [100,150,200,250]}
clf_xgb2 = RandomizedSearchCV(xgb_model2, parameters,n_jobs = -1,cv = 5, scoring='roc_au
c',return_train_score = True)

start = time.time()
clf_xgb2.fit(X_tr_2, Y_train)
end = time.time()
print (end-start,'s')
```

```
4423.194231748581 s
```

In [29]:

```
results2 = pd.DataFrame.from_dict(clf_xgb2.cv_results_)[['param_n_estimators', 'param_max
_depth', 'params','mean_test_score','mean_train_score','rank_test_score']].sort_values([
'rank_test_score'])
results2.head()
```

Out[29]:

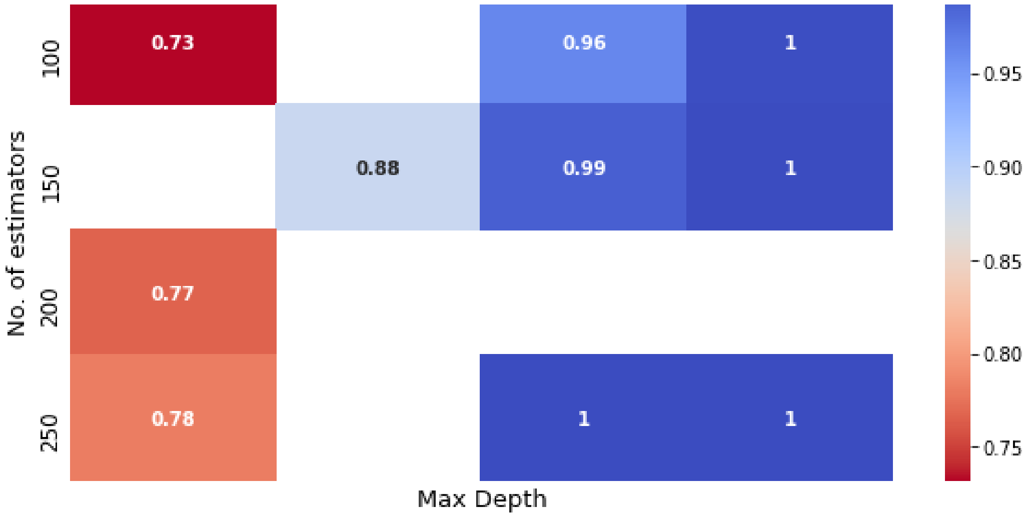
	param_n_estimators	param_max_depth	params	mean_test_score	mean_train_score	rank_test_score
8	200	2	{'n_estimators': 200, 'max_depth': 2}	0.687415	0.767211	1
4	250	2	{'n_estimators': 250, 'max_depth': 2}	0.686783	0.780966	2
0	100	2	{'n_estimators': 100, 'max_depth': 2}	0.680814	0.731638	3
5	150	4	{'n_estimators': 150, 'max_depth': 4}	0.679543	0.884898	4
7	150	6	{'n_estimators': 150, 'max_depth': 6}	0.667086	0.987744	5

In [30]:

```
plt.figure(figsize=(10, 5))
heat_data = results2.pivot(index='param_n_estimators', columns='param_max_depth', values
='mean_train_score')
vmin = results2['mean_train_score'].min()
vmax = results2['mean_train_score'].max()
heatmap = sns.heatmap(heat_data, vmin, vmax, annot=True,annot_kws={"fontsize":10,"weight
": "bold"}, cmap='coolwarm_r')
heatmap.tick_params(axis='both', which='major', labelsize=12, labelbottom = False, botto
m=False, top = False, labeltop=True,length = 0)
heatmap.set_title('Train Performance Heatmap : Set 2',fontsize = 15)
plt.xlabel('Max Depth',fontsize = 13)
plt.ylabel('No. of estimators',fontsize = 13)
```

Out[30]:

```
Text(69.0, 0.5, 'No. of estimators')
```

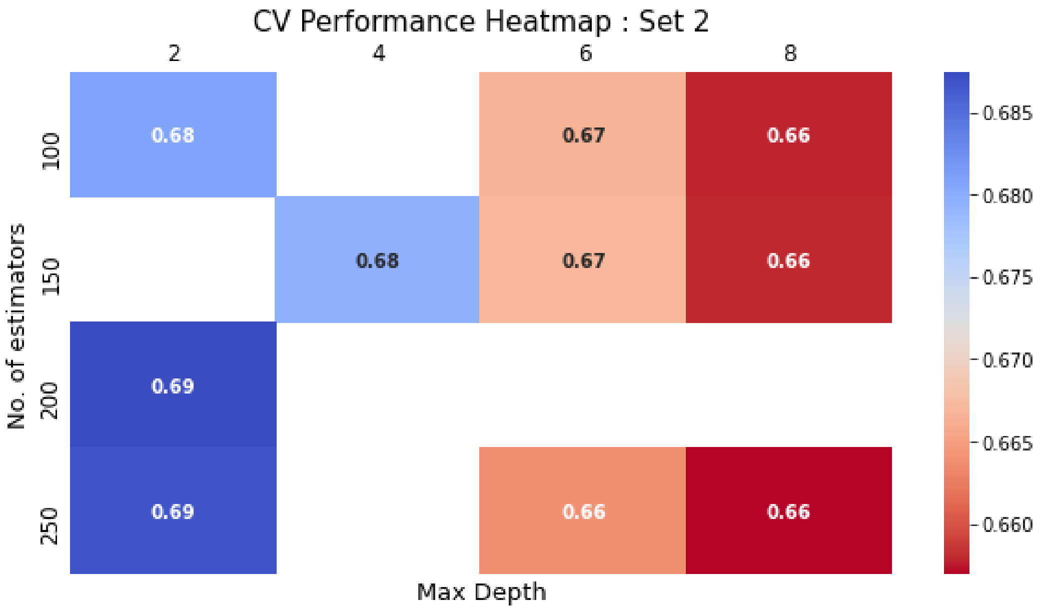


In [31]:

```
plt.figure(figsize=(10, 5))
heat_data = results2.pivot(index='param_n_estimators', columns='param_max_depth', values
='mean_test_score')
vmin = results2['mean_test_score'].min()
vmax = results2['mean_test_score'].max()
heatmap = sns.heatmap(heat_data, vmin, vmax, annot=True,annot_kws={"fontsize":10,"weight
": "bold"}, cmap='coolwarm_r')
heatmap.tick_params(axis='both', which='major', labelsize=12, labelbottom = False, botto
m=False, top = False, labeltop=True,length = 0)
heatmap.set_title('CV Performance Heatmap : Set 2',fontsize = 15)
plt.xlabel('Max Depth',fontsize = 13)
plt.ylabel('No. of estimators',fontsize = 13)
```

Out[31]:

```
Text(69.0, 0.5, 'No. of estimators')
```



In [34]:

```
best_depth2 =  results2[results2['rank_test_score'] == 1]['param_max_depth'].values[0]
best_n_estimators2 = results2[results2['rank_test_score'] == 1]['param_n_estimators'].val
ues[0]
```

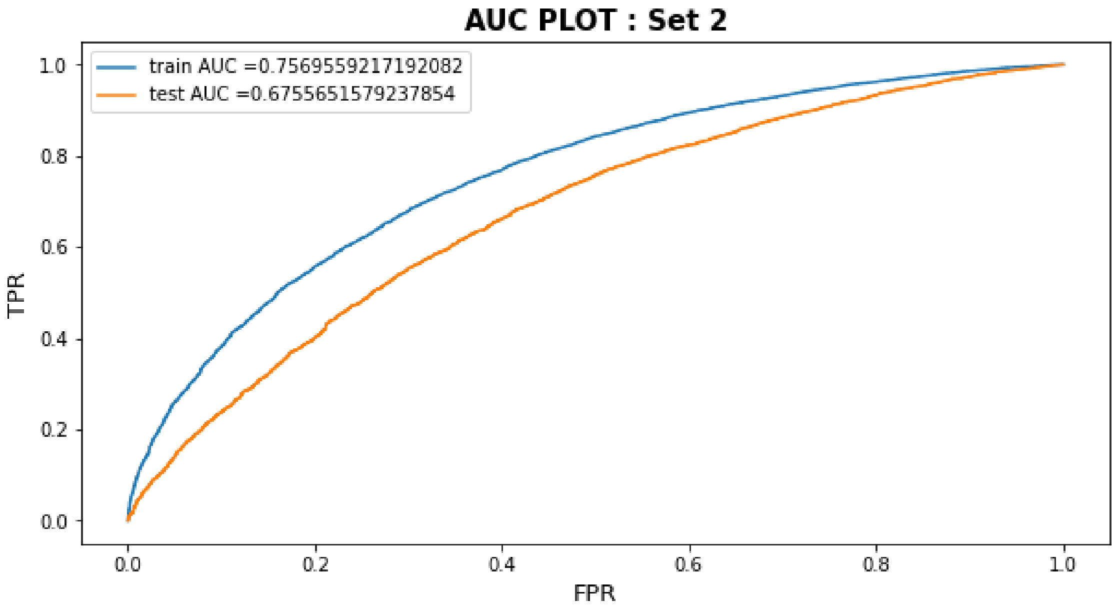
In [37]:

```
from sklearn.metrics import roc_curve, auc

xgb_model_optimal2 = XGBClassifier(colsample_bytree = 0.5,subsample = 0.5,scale_pos_weigh
t=0.18,max_depth = best_depth2,n_estimators = best_n_estimators2)
xgb_model_optimal2.fit(X_tr_2, Y_train)
```

```
Y_train_pred2 = batch_predict(xgb_model_optimal2, X_tr_2)
Y_test_pred2 = batch_predict(xgb_model_optimal2, X_te_2)
```

In [38]:

```
plt.figure(figsize=(10, 5))
plt.grid()
train_fpr2, train_tpr2,tr_thresholds2 = roc_curve(Y_train, Y_train_pred2)
test_fpr2, test_tpr2,te_thresholds2 = roc_curve(Y_test, Y_test_pred2)

AUC2 = auc(test_fpr2, test_tpr2)
plt.plot(train_fpr2, train_tpr2, label="train AUC ="+str(auc(train_fpr2, train_tpr2)))
plt.plot(test_fpr2, test_tpr2, label="test AUC ="+str(auc(test_fpr2, test_tpr2)))
plt.legend()
plt.xlabel("FPR",fontsize = 13)
plt.ylabel("TPR",fontsize = 13)
plt.title("AUC PLOT : Set 2",fontsize = 15,weight = "bold")
plt.grid()
plt.show()
```



In [39]:

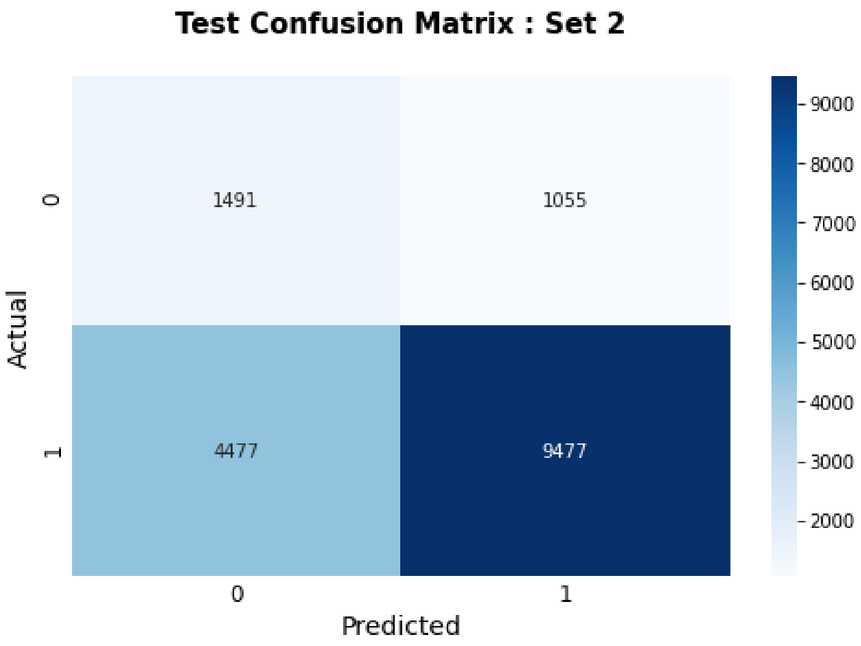
```
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]

    # (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))
    print ('\n')
    return t

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

In [40]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

best_t2 = find_best_threshold(tr_thresholds2, train_fpr2, train_tpr2)

print("Test confusion matrix : Set 2\n")
print (confusion_matrix(Y_test, predict_with_best_t(Y_test_pred2, best_t2)))
```

```
tn2, fp2, fn2, tp2 = confusion_matrix(Y_test, predict_with_best_t(Y_test_pred2, best_t2)
).ravel()
print ("\nTrue Negative  : ",tn2)
print ("False Positive : ",fp2)
print ("False Negative : ",fn2)
print ("True Positive  : ",tp2)
test_cm = np.array([[tn2,fp2 ],[fn2, tp2 ]])

plt.figure(figsize=(8, 5))
cm = sns.heatmap(test_cm, annot=True,fmt="d",cmap='Blues')
cm.tick_params(axis='both', which='major', labelsize=12, labelbottom = True, bottom=False, top = False, labeltop=False,length = 0)
cm.set_title('\nTest Confusion Matrix : Set 2\n',fontsize = 15,weight = 'bold')
plt.xlabel('Predicted',fontsize = 14)
plt.ylabel('Actual',fontsize = 14)
plt.show()
```

```
The maximum value of tpr*(1-fpr) 0.4776964690889842 for threshold 0.488


Test confusion matrix : Set 2

[[1491 1055]
 [4477 9477]]

True Negative  :  1491
False Positive :  1055
False Negative :  4477
True Positive  :  9477
```



In [59]:

```
Vectorizer = ['TFIDF','TFIDFW2V']
Model = ['XGBoost','XGBoost']
Hyper_Parameter = [(results1.loc[results1['rank_test_score'] == 1,'params']).to_string(index = False),(results2.loc[results2['rank_test_score']==1,'params']).to_string(index = False)]
AUC = [AUC1,AUC2]

summary_df = pd.DataFrame(list(zip(Vectorizer,Model,Hyper_Parameter,AUC)),columns= ['Vectorizer','Model','Hyper_Parameter','AUC'])
```

In [60]:

```
from tabulate import tabulate
print(tabulate(summary_df, headers='keys', tablefmt='psql',showindex=False))
```

```
+--------------+---------+---------------------------------------+----------+
| Vectorizer   | Model   | Hyper_Parameter                       |      AUC |
|--------------+---------+---------------------------------------+----------|
| TFIDF        | XGBoost | {'n_estimators': 250, 'max_depth': 2} | 0.69415  |
| TFIDFW2V     | XGBoost | {'n_estimators': 200, 'max_depth': 2} | 0.675565 |
+--------------+---------+---------------------------------------+----------+
```