Assignment 9: GBDT

Response Coding: Example

Train Data

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

Resonse table(only from train)

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

Encoded Train Data

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

Test Data

State
A
C
D
C
B
E

Encoded Test Data

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

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

1. **Apply GBDT on these feature sets**
   - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check the bellow example, include all 4 values as 4 features)
   - Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (Consider any two hyper parameters)**
   - Find the best hyper parameter which will give the maximum AUC value
   - find the best hyper paramter using k-fold cross validation/simple cross validation data
   - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task
3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the

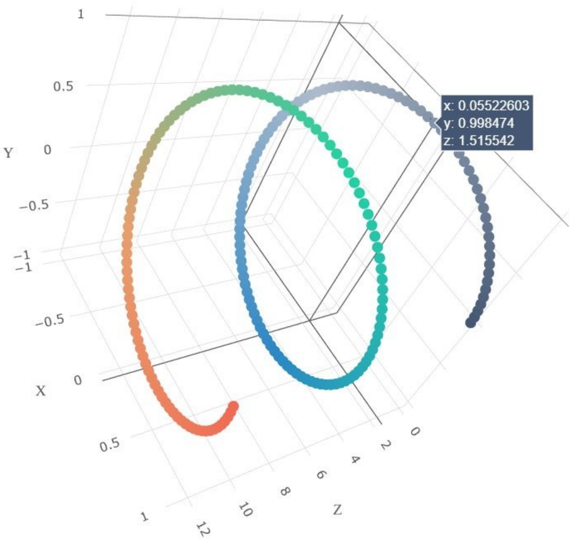
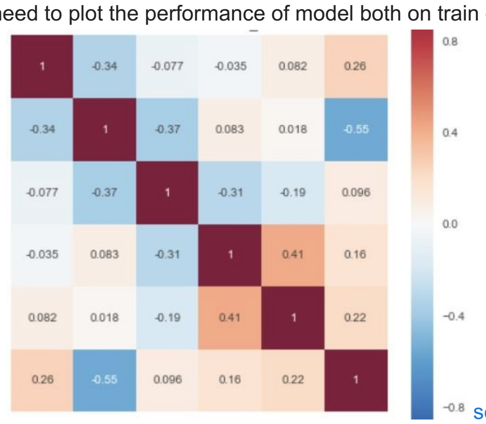


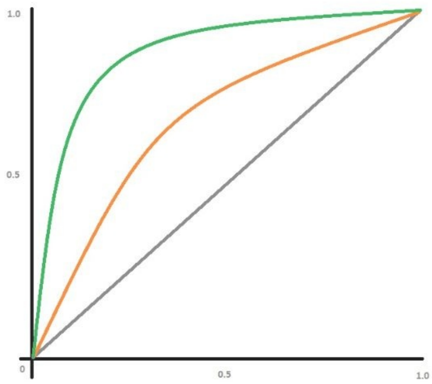
figure with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

or

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



seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**
- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.



- Along with plotting ROC curve, you need to print the confusion matrix with predicted 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 [32]:

```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\subhashini rajesh\anaconda3\lib\site-packages (1.4.2)
Requirement already satisfied: scipy in c:\users\subhashini rajesh\anaconda3\lib\site-packages (from xgboost) (1.
5.2)
Requirement already satisfied: numpy in c:\users\subhashini rajesh\anaconda3\lib\site-packages (from xgboost) (1.
19.2)
```

In [1]:

```
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 students with the biggest
for learning my students learn in many different ways using all of our senses and multiple intelligences i use a
of techniques to help all my students succeed students in my class come from a variety of different backgrounds w
for wonderful sharing of experiences and cultures including native americans our school is a caring community of
learners which can be seen through collaborative student project based learning in and out of the classroom kinde
in my class love to work with hands on materials and have many different opportunities to practice a skill before
mastered having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curr
montana is the perfect place to learn about agriculture and nutrition my students love to role play in our preter
in the early childhood classroom i have had several kids ask me can we try cooking with real food i will take the
and create common core cooking lessons where we learn important math and writing concepts while cooking delicious
food for snack time my students will have a grounded appreciation for the work that went into making the food and
of where the ingredients came from as well as how it is healthy for their bodies this project would expand our le
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce make our
and mix up healthy plants from our classroom garden in the spring we will also create our own cookbooks to be pri
shared with families students will gain math and literature skills as well as a life long enjoyment for healthy c
nannan'
ss = sid.polarity_scores(for_sentiment)

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

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

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

In [2]:

```
import warnings
warnings.filterwarnings("ignore")
import pickle
import pandas as pd
import numpy as np
from sklearn import tree
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Normalizer
from tqdm import tqdm
from sklearn.feature_extraction.text import TfidfVectorizer
import scipy
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
import seaborn as sns
from sklearn.metrics import roc_curve, auc
from wordcloud import WordCloud, STOPWORDS
from sklearn.linear_model import LogisticRegression
from scipy.sparse import hstack
from scipy.sparse import coo_matrix
from collections import Counter
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score
```

1. GBDT (xgboost/lightgbm)

1.1 Loading Data

In [3]:
```python
import pandas
data = pandas.read_csv('preprocessed_data.csv', nrows = 35000)
data.head()
```

Out[3]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	cl
0	ca	mrs	grades_prek_2	53	1	math_science	
1	ut	ms	grades_3_5	4	1	specialneeds	
2	ca	mrs	grades_prek_2	10	1	literacy_language	
3	ga	mrs	grades_prek_2	2	1	appliedlearning	
4	wa	mrs	grades_3_5	2	1	literacy_language	

In [4]:
```python
# Sentiment Analysis on 'essay'
sid    = SentimentIntensityAnalyzer()

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

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

```
100%|██████████| 35000/35000 [00:57<00:00, 608.49it/s]
```

In [5]:
```python
# Now append these sentiments columns/features to original preprocessed dataframe
data['negative_sent'] = negative_sentiments
data['positive_sent'] = positive_sentiments
data['neutral_sent'] = neutral_sentiments
data['compound_sent'] = compound_sentiments
data.columns
```

Out[5]:
```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price',
       'negative_sent', 'positive_sent', 'neutral_sent', 'compound_sent'],
      dtype='object')
```

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

In [6]:
```
# 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
# Sepearting input data and labels to have a proper data-matrix
Y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
X_train['negative_sent'].shape
```

```
(24500, 12) (24500,)
(10500, 12) (10500,)
```

Out[6]:
```
(24500,)
```

1.3 Make Data Model Ready: encoding eassay

In [7]:
```
#please use below code to load glove vectors
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [8]:
```
# As required for Task-1, applying TFIDF on the Essay column
vectorizer_essay_tfidf = TfidfVectorizer(min_df=10)

# Apply .fit() on this vectorizer on Train data
# Note .fit() is applied only on the train data, as test and cv should not be fitted
vectorizer_essay_tfidf.fit(X_train['essay'].values)

# Now use the fitted TfidfVectorizer for converting 'essay' text to Vector form
X_train_vectorized_tfidf_essay = vectorizer_essay_tfidf.transform(X_train['essay'].values)
X_test_vectorized_tfidf_essay = vectorizer_essay_tfidf.transform(X_test['essay'].values)

print('After TFIDF on Essay column checking the shapes ')
print(X_train_vectorized_tfidf_essay.shape, y_train.shape)
print(X_test_vectorized_tfidf_essay.shape, y_test.shape)
```

```
After TFIDF on Essay column checking the shapes
(24500, 9101) (24500,)
(10500, 9101) (10500,)
```

In [9]:
```
print('After TFIDF on Essay column checking the shapes ')
print(X_train_vectorized_tfidf_essay.shape, y_train.shape)
print(X_test_vectorized_tfidf_essay.shape, y_test.shape)
```

```
After TFIDF on Essay column checking the shapes
(24500, 9101) (24500,)
(10500, 9101) (10500,)
```

In [10]:
```
# Reference_Vectorization from AAIC
# average Word2Vec
# compute average word2vec for each review.
def tfidfw2v(data):
    avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(data): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        cnt_words =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:
                vector += model[word]
                cnt_words += 1
        if cnt_words != 0:
            vector /= cnt_words
        avg_w2v_vectors.append(vector)
    return np.array(avg_w2v_vectors)

#convert essay to vectors
```

```
X_train_tfidfw2v = tfidfw2v(X_train['essay'].values)
X_test_tfidfw2v = tfidfw2v(X_test['essay'].values)

print(X_train_tfidfw2v.shape, y_train.shape)
print(X_test_tfidfw2v.shape, y_test.shape)
```

```
100%|██████████| 24500/24500 [00:05<00:00, 4105.08it/s]
100%|██████████| 10500/10500 [00:02<00:00, 3925.88it/s]

(24500, 300) (24500,)
(10500, 300) (10500,)
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [11]:

```
# the numerical features are  teacher_number_of_previously_posted_projects      price
normalizer = Normalizer()
num_data = ['price', 'teacher_number_of_previously_posted_projects', 'negative_sent', 'positive_sent', 'neutral_s
for j in num_data:
    normalizer.fit(X_train[j].values.reshape(-1,1))

# scaling the numeric feature price
X_train_normalized_price = normalizer.transform(X_train['price'].values.reshape(-1,1))
X_test_normalized_price = normalizer.transform(X_test['price'].values.reshape(-1,1))

#  scaling the numeric feature prev posted projects
X_train_normalized_proj = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.res
X_test_normalized_proj = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.resha

# scaling the numeric feature negative sent
X_train_normalized_neg = normalizer.transform(X_train['negative_sent'].values.reshape(-1,1))
X_test_normalized_neg = normalizer.transform(X_test['negative_sent'].values.reshape(-1,1))

# scaling the numeric feature positive sent
X_train_normalized_pos = normalizer.transform(X_train['positive_sent'].values.reshape(-1,1))
X_test_normalized_pos = normalizer.transform(X_test['positive_sent'].values.reshape(-1,1))

# scaling the numeric feature neutral sent
X_train_normalized_neu = normalizer.transform(X_train['neutral_sent'].values.reshape(-1,1))
X_test_normalized_neu = normalizer.transform(X_test['neutral_sent'].values.reshape(-1,1))

# scaling the numeric feature compound sent
X_train_normalized_com = normalizer.transform(X_train['compound_sent'].values.reshape(-1,1))
X_test_normalized_com= normalizer.transform(X_test['compound_sent'].values.reshape(-1,1))

print(X_train_normalized_price.shape, y_train.shape)
print(X_test_normalized_price.shape, y_test.shape)
print("****************")

print(X_train_normalized_proj.shape, y_train.shape)
print(X_test_normalized_proj.shape, y_test.shape)
print("****************")

print(X_train_normalized_neg.shape, y_train.shape)
print(X_test_normalized_neg.shape, y_test.shape)
print("****************")

print(X_train_normalized_neu.shape, y_train.shape)
print(X_test_normalized_neu.shape, y_test.shape)
print("****************")

print(X_train_normalized_pos.shape, y_train.shape)
print(X_test_normalized_pos.shape, y_test.shape)
print("****************")

print(X_train_normalized_com.shape, y_train.shape)
print(X_test_normalized_com.shape, y_test.shape)
```

```
(24500, 1) (24500,)
(10500, 1) (10500,)
****************
(24500, 1) (24500,)
(10500, 1) (10500,)
****************
(24500, 1) (24500,)
(10500, 1) (10500,)
****************
(24500, 1) (24500,)
(10500, 1) (10500,)
****************
(24500, 1) (24500,)
(10500, 1) (10500,)
****************
```

```
(24500, 1) (24500,)
(10500, 1) (10500,)
```

In [ ]:

In [12]:

```
# Categorical features are school_state, teacher_prefix, project_grade_category, clean_categories,clean_subcatego
# prerequitie datas
# combine X_train with target variable to filter out categorical name with 1 and 0 count
df_cat = pd.DataFrame(y_train, columns=['project_is_approved'])
df_cat['school_state'] = X_train['school_state'].values
df_cat['teacher_prefix'] = X_train['teacher_prefix'].values
df_cat['project_grade_category'] = X_train['project_grade_category'].values
df_cat['clean_categories'] = X_train['clean_categories'].values
df_cat['clean_subcategories'] = X_train['clean_subcategories'].values
df_cat.head(5)
```

Out[12]:

	project_is_approved	school_state	teacher_prefix	project_grade_category	clean_categories	clean_subcategories
0	1	md	ms	grades_3_5	literacy_language	literacy
1	1	mo	mrs	grades_prek_2	health_sports	health_wellness
2	0	de	mrs	grades_9_12	literacy_language	foreignlanguages
3	1	ca	ms	grades_9_12	literacy_language	foreignlanguages literature_writing
4	0	ca	ms	grades_prek_2	math_science literacy_language	appliedsciences literacy

Functions for response code

In [13]:

```
#response code fit() which returns three dictionaries for total count of categories, count of approved and non ap
def response_code_fit(category, total, zero, one):

    total_dict = {category[i] : total[i]   for i in range(len(category))}
    zero_dict = {category[i] : zero[i] for i in range(len(zero))}
    one_dict = {category[i] : one[i] for i in range(len(one))}

    for i in total_dict:
        if i not in zero_dict:
            zero_dict[i] = 0

    for i in total_dict:
        if i not in one_dict:
            one_dict[i] = 0


    return total_dict, zero_dict, one_dict


# response code transform() which returns the probability of approved and non approved categories as list
def response_code_transform(category_in_X_train, total_count, zero_count, one_count):
    prob0 = []
    prob1 = []
    for i in category_in_X_train:
        if i in total_count.keys():
            prob0.append(zero_count[i]/total_count[i])
            prob1.append(one_count[i]/total_count[i])
        else:
            prob0.append(0.5)
            prob1.append(0.5)
    prob0 = np.array(prob0)
    prob1 = np.array(prob1)

    return prob0.reshape(-1, 1), prob1.reshape(-1, 1)
```

Response code for School State

In [14]:

```
scl_total = df_cat['school_state'].value_counts()
scl_idx = df_cat['school_state'].value_counts().index
scl_zero = df_cat.loc[df_cat['project_is_approved']==0]['school_state'].value_counts()
scl_one = df_cat.loc[df_cat['project_is_approved']==1]['school_state'].value_counts()
school_total, school_0, school_1 = response_code_fit(scl_idx, scl_total, scl_zero, scl_one)
X_train_school_prob_0, X_train_school_prob_1 = response_code_transform(X_train['school_state'], school_total, sch
X_test_school_prob_0, X_test_school_prob_1 = response_code_transform(X_test['school_state'], school_total, school


print(X_train_school_prob_0.shape, y_train.shape)
print(X_test_school_prob_0.shape, y_test.shape)
```

```
print(X_train_school_prob_0)
# print(scl_one)
```

```
(24500, 1) (24500,)
(10500, 1) (10500,)
[[0.16103896]
 [0.13533835]
 [0.13333333]
 ...
 [0.13533835]
 [0.15865701]
 [0.14893617]]
```

Response Encoding teacher_prefix

In [15]:

```
teacher_total = list(df_cat['teacher_prefix'].value_counts())
teacher_idx = list(df_cat['teacher_prefix'].value_counts().index)
teacher_zero = list(df_cat.loc[df_cat['project_is_approved']==0]['teacher_prefix'].value_counts())
teacher_one = list(df_cat.loc[df_cat['project_is_approved']==1]['teacher_prefix'].value_counts())


teacher_total, teacher_0, teacher_1 = response_code_fit(teacher_idx, teacher_total, teacher_zero, teacher_one)
X_train_teacher_prob_0, X_train_teacher_prob_1 = response_code_transform(X_train['teacher_prefix'], teacher_total
X_test_teacher_prob_0, X_test_teacher_prob_1 = response_code_transform(X_test['teacher_prefix'], teacher_total, t



print(X_train_teacher_prob_0.shape, y_train.shape)
print(X_test_teacher_prob_0.shape, y_test.shape)
```

```
(24500, 1) (24500,)
(10500, 1) (10500,)
```

Response Encoding project_grade_category

In [16]:

```
proj_total = list(df_cat['project_grade_category'].value_counts())
proj_idx = list(df_cat['project_grade_category'].value_counts().index)
proj_zero = list(df_cat.loc[df_cat['project_is_approved']==0]['project_grade_category'].value_counts())
proj_one = list(df_cat.loc[df_cat['project_is_approved']==1]['project_grade_category'].value_counts())


proj_total, proj_0, proj_1 = response_code_fit(proj_idx, proj_total, proj_zero, proj_one)
X_train_proj_prob_0, X_train_proj_prob_1 = response_code_transform(X_train['project_grade_category'], proj_total,
X_test_proj_prob_0, X_test_proj_prob_1 = response_code_transform(X_test['project_grade_category'], proj_total, pr

print(X_train_proj_prob_0.shape, y_train.shape)
print(X_test_proj_prob_0.shape, y_test.shape)
```

```
(24500, 1) (24500,)
(10500, 1) (10500,)
```

Response Encoding clean_categories

In [17]:

```
cln_total = list(df_cat['clean_categories'].value_counts())
cln_idx = list(df_cat['clean_categories'].value_counts().index)
cln_zero = list(df_cat.loc[df_cat['project_is_approved']==0]['clean_categories'].value_counts())
cln_one = list(df_cat.loc[df_cat['project_is_approved']==1]['clean_categories'].value_counts())

print(len(cln_total))
print(len(cln_idx))
print(len(cln_zero))
print(len(cln_one))
print((cln_zero))

cln_total, cln_0, cln_1 = response_code_fit(cln_idx, cln_total, cln_zero, cln_one)


X_train_cln_prob_0, X_train_cln_prob_1 = response_code_transform(X_train['clean_categories'], cln_total, cln_0, c
X_test_cln_prob_0, X_test_cln_prob_1 = response_code_transform(X_test['clean_categories'], cln_total, cln_0, cln_


print(X_train_cln_prob_0.shape, y_train.shape)
print(X_test_cln_prob_0.shape, y_test.shape)
```

```
43
43
41
43
[709, 630, 538, 452, 139, 136, 136, 120, 80, 68, 67, 61, 60, 55, 49, 45, 42, 39, 39, 38, 31, 29, 26, 22, 22, 21,
19, 13, 11, 10, 9, 8, 8, 7, 3, 3, 2, 2, 2, 2, 1]
(24500, 1) (24500,)
(10500, 1) (10500,)
```

Response Encoding - clean_subcategories

In [18]:
```
clnsub_total = list(df_cat['clean_subcategories'].value_counts())
clnsub_idx = list(df_cat['clean_subcategories'].value_counts().index)
clnsub_zero = list(df_cat.loc[df_cat['project_is_approved']==0]['clean_subcategories'].value_counts())
clnsub_one = list(df_cat.loc[df_cat['project_is_approved']==1]['clean_subcategories'].value_counts())


cln_sub_total, clnsub_0, clnsub_1 = response_code_fit(clnsub_idx, clnsub_total, clnsub_zero, clnsub_one)

X_train_clnsub_prob_0, X_train_clnsub_prob_1 = response_code_transform(X_train['clean_subcategories'], cln_sub_to
X_test_clnsub_prob_0, X_test_clnsub_prob_1 = response_code_transform(X_test['clean_subcategories'], cln_sub_total

print(len(clnsub_total))
print(len(clnsub_idx))
print(len(clnsub_zero))
print(len(clnsub_one))
print((clnsub_zero))


print(X_train_clnsub_prob_0.shape, y_train.shape)
print(X_test_clnsub_prob_0.shape, y_test.shape)
```

```
319
319
231
309
[248, 238, 226, 197, 194, 186, 153, 136, 120, 119, 88, 68, 65, 59, 53, 51, 47, 44, 40, 40, 40, 40, 38, 38, 36, 33
, 33, 31, 28, 27, 27, 26, 24, 24, 24, 23, 22, 22, 21, 21, 20, 20, 19, 18, 16, 16, 16, 15, 15, 15, 14, 14, 13, 13,
13, 13, 12, 12, 12, 11, 11, 11, 10, 10, 10, 10, 10, 10, 10, 9, 9, 9, 9, 9, 9, 8, 7, 7, 7, 7, 7, 7, 6, 6, 6, 6,
6, 5, 5, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
(24500, 1) (24500,)
(10500, 1) (10500,)
```

Set 1 : categorical(probability values), numerical features + preproc _essay (TFIDF) + sentiment Score of essay

In [19]:
```
#SET1 X_train
X_train_set1 = hstack((X_train_school_prob_0, X_train_school_prob_1, X_train_teacher_prob_0, X_train_teacher_prob
                                   X_train_normalized_com, X_train_vectorized_tfidf_essay )).tocsr()
```

In [20]:
```
#SET1 X_test
X_test_set1 = hstack((X_test_school_prob_0, X_test_school_prob_1, X_test_teacher_prob_0, X_test_teacher_prob_1, X
                                   X_test_normalized_com, X_test_vectorized_tfidf_essay )).tocsr()
```

Set 2 : categorical(probability values), numerical features + preproc _essay (TFIDFW2V) + sentiment Score of essay

In [21]:
```
#SET2 X_TRAIN
X_train_set2 = np.hstack((X_train_school_prob_0, X_train_school_prob_1, X_train_teacher_prob_0, X_train_teacher_p
                                   X_train_normalized_com, X_train_tfidfw2v ))
```

In [22]:
```
#SET2 X_TEST
X_test_set2 = np.hstack((X_test_school_prob_0, X_test_school_prob_1, X_test_teacher_prob_0, X_test_teacher_prob_1
                                   X_test_normalized_com, X_test_tfidfw2v))
```

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

Set S1 - GridSearchCV with XGBClassifier

In [40]:
```
xgb_clf_s1 = XGBClassifier(eval_metric='mlogloss')

params = {
'learning_rate' : [0.00001, 0.0001, 0.001, 0.01],
'n_estimators' : [5, 10, 50, 75, 100, 200],
'tree_method':['gpu_hist']
}

grid_search_s1 = GridSearchCV(xgb_clf_s1, params, cv=3, scoring='roc_auc', return_train_score=True)

grid_search_s1.fit(X_train_set1, y_train)

best_params_gridsearch_xgb_s1 = grid_search_s1.best_params_

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

```
Best Params from GridSearchCV with XGB for Set s2  {'learning_rate': 0.01, 'n_estimators': 200, 'tree_method': 'g
pu_hist'}
```

AUC for Set S1

In [41]:
```
learning_rates = [0.00001, 0.0001, 0.001, 0.01]
n_estimators = [5, 10, 50, 75, 100, 200]

def get_auc_matrix(x_train, x_test, y_train, y_test ):

    train_auc_final_arr, test_auc_final_arr = [], []

    for l_rate in tqdm(learning_rates):
        train_auc_batch, test_auc_batch = [], []

        for num in n_estimators:
            # Below gives large number of warnings
            # xgb_clf = XGBClassifier(n_estimators=num, eta=l_rate, reg_alpha=0, reg_lambda=0, tree_method='gpu_h

            # below works after including eval_metric='mlogloss'
            # xgb_clf = XGBClassifier(n_estimators=num, eval_metric='mlogloss', learning_rate=l_rate, reg_alpha=0

            # Only changing the name of the parameter learning_rate to eta
            xgb_clf = XGBClassifier(n_estimators=num, eval_metric='mlogloss', eta=l_rate, reg_alpha=0, reg_lambda

            xgb_clf.fit(x_train, y_train)

            # I have to predict probabilities (clf.predict_proba) instead of classes for calculating of ROC AUC s
            y_train_predicted = xgb_clf.predict_proba(x_train)[:, 1]
            y_test_predicted = xgb_clf.predict_proba(x_test)[:, 1]

            train_auc = roc_auc_score(y_train, y_train_predicted)
            test_auc = roc_auc_score(y_test, y_test_predicted)

            train_auc_batch.append(train_auc)
            test_auc_batch.append(test_auc)

        train_auc_final_arr.append(train_auc_batch)
        test_auc_final_arr.append(test_auc_batch)

    return train_auc_final_arr, test_auc_final_arr


train_auc_final_arr_s1, test_auc_final_arr_s1 = get_auc_matrix(X_train_set1, X_test_set1, y_train, y_test)

print(train_auc_final_arr_s1)
print(test_auc_final_arr_s1)
```

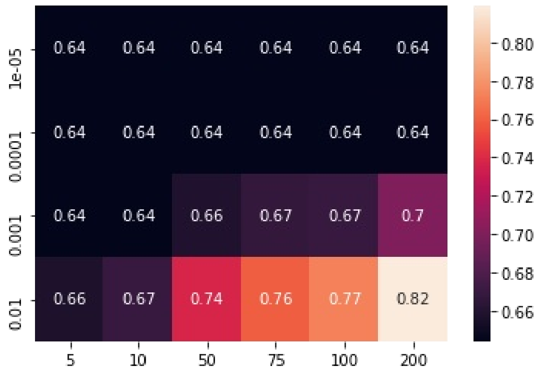
```
100%|██████████| 4/4 [43:35<00:00, 653.85s/it]
```

```
[[0.6440905400639267, 0.6440386849675973, 0.6440430377910852, 0.6439181926501638, 0.6438905092063886, 0.643918930
9609324], [0.6440764543784807, 0.6439164528047874, 0.6443552020041375, 0.644461197750132, 0.6441127645637128, 0.6
445691965653424], [0.6438186105777155, 0.6439635313514488, 0.6587956425643169, 0.6676475521133124, 0.669223049512
6353, 0.7003660570471031], [0.6570902795108465, 0.6697605911129161, 0.7381025906310494, 0.7597668627739909, 0.772
8233430085001, 0.8189132145095555]]
[[0.6093405503595112, 0.6093377655832425, 0.608726472382573, 0.6088246357460425, 0.6088240787907888, 0.6088188573
352853], [0.6087362190995134, 0.608727342625157, 0.6089333812593329, 0.6088899735592455, 0.6089168466502379, 0.60
88466354785648], [0.6088751098159116, 0.6088073701331771, 0.6170441813805543, 0.6162326279564664, 0.6163214971291
393, 0.6267845281450724], [0.6173108237082764, 0.6154742637591097, 0.6374664773854237, 0.6442217180384788, 0.6472
415990435965, 0.6649559786048425]]
```

Heatmap for Set S1

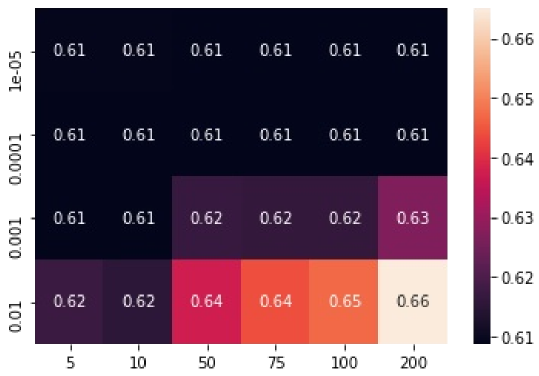
In [42]:
```
train_auc_final_df_s1 = pd.DataFrame(train_auc_final_arr_s1, columns=n_estimators, index=learning_rates)
sns.heatmap(train_auc_final_df_s1, annot=True)
# train_auc_final_df_s1
```

Out[42]: <AxesSubplot:>



In [43]:
```
test_auc_final_df_s1 = pd.DataFrame(test_auc_final_arr_s1, columns=n_estimators, index=learning_rates)
sns.heatmap(test_auc_final_df_s1, annot=True)
# test_auc_final_df_s1
```

Out[43]: <AxesSubplot:>

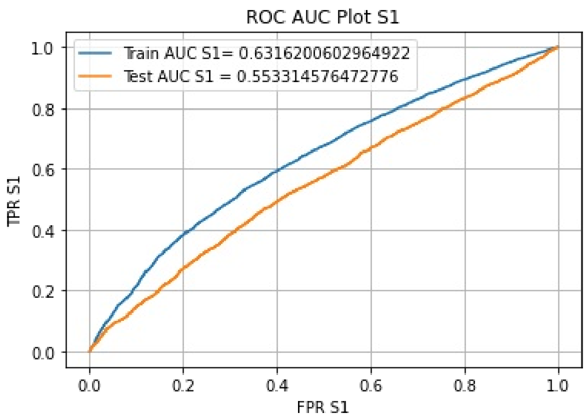


ROC Curve for Set S1

In [61]:
```
# Best Params from GridSearchCV with XGB for Set s1  {'learning_rate': 0.01, 'n_estimators': 200, 'tree_method':
# Whether i chose the learning rate which is given by GridSearchCV which takes the model to underfit so here i ch
xgb_clf = XGBClassifier(n_estimators=200, learning_rate=0.0001, reg_alpha=0, reg_lambda=0, booster='gblinear', tr

xgb_clf.fit(X_train_set1, y_train)

# I have to predict probabilities (clf.predict_proba) instead of classes for calculating of ROC AUC score:
y_train_predicted = xgb_clf.predict_proba(X_train_set1)[:, 1]
y_test_predicted = xgb_clf.predict_proba(X_test_set1)[:, 1]

train_fpr_s1, train_tpr_s1, train_thresholds_s1 = roc_curve(y_train, y_train_predicted)

test_fpr_s1, test_tpr_s1, test_thresholds_s1 = roc_curve(y_test, y_test_predicted)

plt.plot(train_fpr_s1, train_tpr_s1, label="Train AUC S1= "+str(auc(train_fpr_s1, train_tpr_s1)))
plt.plot(test_fpr_s1, test_tpr_s1, label="Test AUC S1 = "+str(auc(test_fpr_s1, test_tpr_s1)))
plt.legend()
plt.xlabel("FPR S1")
plt.ylabel('TPR S1')
plt.title('ROC AUC Plot S1')
plt.grid()
plt.show()
```

```
[10:23:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:573:
Parameters: { "tree_method" } might not be used.
```

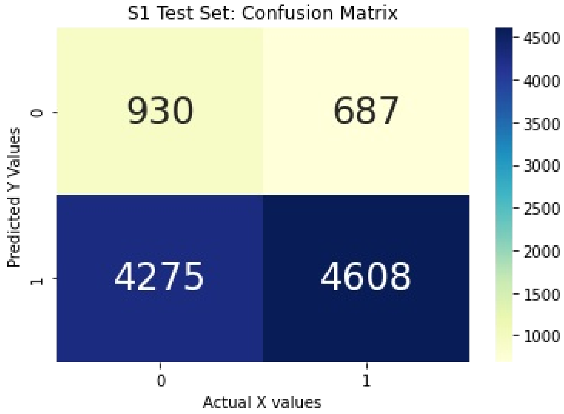
```
This may not be accurate due to some parameters are only used in language bindings but
passed down to XGBoost core.  Or some parameters are not used but slip through this
verification. Please open an issue if you find above cases.


[10:23:41] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in
XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' t
o 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```



Confusion Matrix for Set S1

In [49]:
```python
#https://medium.com/analytics-vidhya/xgboost-on-kaggle-donor-choose-dataset-28227cd8a869
from sklearn.metrics import confusion_matrix
def get_predicted_y_vec_from_threshold(proba, threshold, fpr, tpr):

    # Using argmax to return the position of the largest value.
    # based on the calculated value of tpr*(1-fpr)
    # tpr * (1-fpr) i.e. optimal_threshold is maximum when fpr is very low and tpr is very high
    optimal_threshold = threshold[np.argmax(tpr * (1-fpr))]

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

    return predicted_y_vector

confusion_matrix_s1_train = confusion_matrix(y_train, get_predicted_y_vec_from_threshold(y_train_predicted, trair

confusion_matrix_s1_test = confusion_matrix(y_test, get_predicted_y_vec_from_threshold(y_test_predicted, test_thr

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

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

heatmap_confusion_matrix_test_s1 = sns.heatmap(confusion_matrix_s1_test, annot=True, fmt='d', cmap="YlGnBu", annc

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

```
confusion_matrix_s1_train  [[ 2294  1460]
 [ 8592 12154]]
```



GridSearch on Set S2

In [51]:
```
# xgb_clf_s2 = XGBClassifier(booster='gblinear', reg_alpha=0, reg_lambda=0)

xgb_clf_s2 = XGBClassifier(eval_metric='mlogloss')

params = {
'eta': [0.0001, 0.001, 0.01, 0.1],
'n_estimators': [5, 10, 50, 75, 100, 200],
'tree_method':['gpu_hist']
}

grid_search_s2 = GridSearchCV(xgb_clf_s2, params, cv=3, scoring='roc_auc', return_train_score=True)

grid_search_s2.fit(X_train_set2, y_train)

best_params_gridsearch_xgb_s2 = grid_search_s2.best_params_

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

```
Best Params from GridSearchCV with XGB for Set s2  {'eta': 0.1, 'n_estimators': 100, 'tree_method': 'gpu_hist'}
```

AUC for Set S2

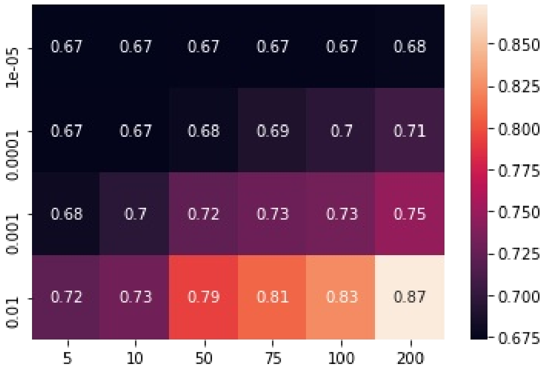
In [52]:
```
train_auc_final_arr_s2, test_auc_final_arr_s2 = get_auc_matrix(X_train_set2, X_test_set2, y_train, y_test)
print("train_auc_final_arr_s2 ", train_auc_final_arr_s2)
print('test_auc_final_arr_s2 ', test_auc_final_arr_s2)
```

```
100%|██████████| 4/4 [02:26<00:00, 36.58s/it]

train_auc_final_arr_s2  [[0.6734103886668192, 0.6734107610322503, 0.6747520726758709, 0.6748315149145709, 0.67487
86897626369, 0.6763113914392211], [0.6748289982378641, 0.6749433272654033, 0.6791887297464665, 0.6893437321216442
, 0.696329403910741, 0.7069628188237762], [0.6800997089334988, 0.695345717163237, 0.7229473882057538, 0.728297406
318122, 0.7321144280510635, 0.7480393419229392], [0.7218290399941532, 0.7315522140309245, 0.7932686319720357, 0.8
088240630348419, 0.8252978949129283, 0.8728585906066018]]
test_auc_final_arr_s2  [[0.6154231979242835, 0.615438618622871, 0.6152434406161429, 0.6150858918987446, 0.6150432
848218346, 0.6162834153136658], [0.6150851608949741, 0.6149717160717306, 0.6167370901775302, 0.6206860769749756,
0.6233714019211196, 0.6284869314974973], [0.6161909955512505, 0.6231521704093712, 0.6334882504371576, 0.635354711
9215088, 0.6374439206976477, 0.6412783835710453], [0.6308185898575246, 0.6367139612182311, 0.6615955542717737, 0.
6660430508310087, 0.6686637689677204, 0.6814666734336731]]
```

Heatmap for Set S2

In [53]:
```
train_auc_final_df_s2 = pd.DataFrame(train_auc_final_arr_s2, columns=n_estimators, index=learning_rates)
sns.heatmap(train_auc_final_df_s2, annot=True)
# train_auc_final_df_s2
```

Out[53]: <AxesSubplot:>



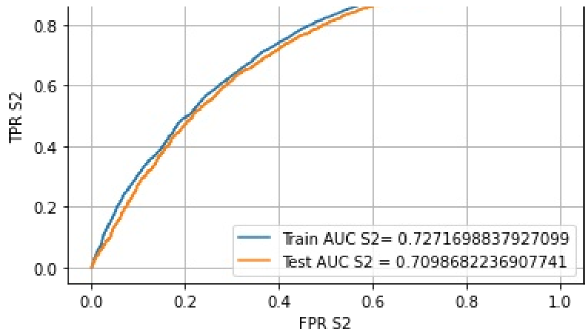
In [54]:
```python
test_auc_final_df_s2 = pd.DataFrame(test_auc_final_arr_s2, columns=n_estimators, index=learning_rates)
sns.heatmap(test_auc_final_df_s2, annot=True)
# test_auc_final_df_s2
```

Out[54]: <AxesSubplot:>

ROC Curve for Set S2

In [57]:
```python
# Best Params from GridSearchCV with XGB for Set s2  {'eta': 0.1, 'n_estimators': 100, 'tree_method': 'gpu_hist'}
xgb_clf = XGBClassifier(n_estimators=100, learning_rate=0.1, reg_alpha=0, reg_lambda=0, booster='gblinear', tree_

xgb_clf.fit(X_train_set2, y_train)

# I have to predict probabilities (clf.predict_proba) instead of classes for calculating of ROC AUC score:
y_train_predicted = xgb_clf.predict_proba(X_train_set2)[:, 1]
y_test_predicted = xgb_clf.predict_proba(X_test_set2)[:, 1]

train_fpr_s2, train_tpr_s2, train_thresholds_s2 = roc_curve(y_train, y_train_predicted)

test_fpr_s2, test_tpr_s2, test_thresholds_s2 = roc_curve(y_test, y_test_predicted)

plt.plot(train_fpr_s2, train_tpr_s2, label="Train AUC S2= "+str(auc(train_fpr_s2, train_tpr_s2)))
plt.plot(test_fpr_s2, test_tpr_s2, label="Test AUC S2 = "+str(auc(test_fpr_s2, test_tpr_s2)))
plt.legend()
plt.xlabel("FPR S2")
plt.ylabel('TPR S2')
plt.title('ROC AUC Plot S2')
plt.grid()
plt.show()
```

```
[10:22:36] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:573:
Parameters: { "tree_method" } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[10:22:36] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in
XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' t
o 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

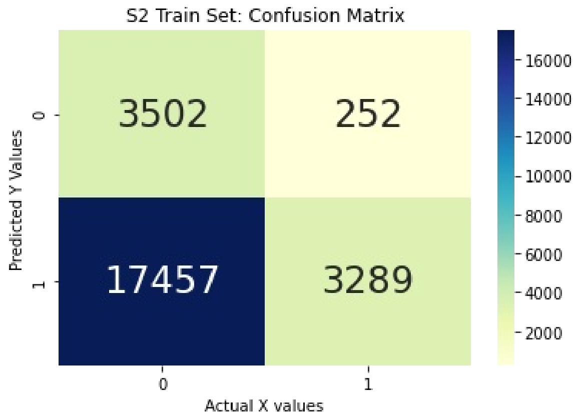


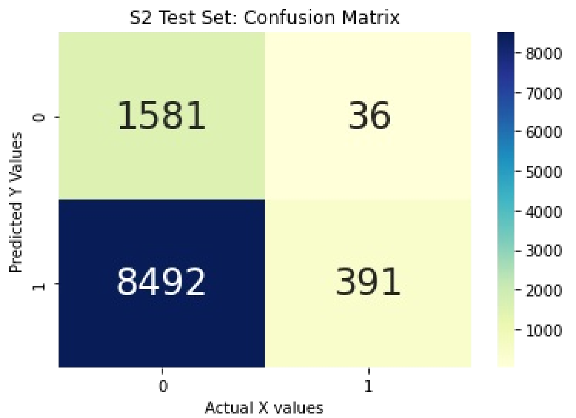
Confusion Matrix for Set S2

In [62]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix_s2_train = confusion_matrix(y_train, get_predicted_y_vec_from_threshold(y_train_predicted, trair

confusion_matrix_s2_test = confusion_matrix(y_test, get_predicted_y_vec_from_threshold(y_test_predicted, test_thr

print('confusion_matrix_s2_train ', confusion_matrix_s2_train)
# Heatmap for Confusion Matrix: Train and SET 1
heatmap_confusion_matrix_train_s2 = sns.heatmap(confusion_matrix_s2_train, annot=True, fmt='d', cmap="YlGnBu", ar

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

heatmap_confusion_matrix_test_s2 = sns.heatmap(confusion_matrix_s2_test, annot=True, fmt='d', cmap="YlGnBu", annc

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

```
confusion_matrix_s2_train  [[ 3502   252]
 [17457  3289]]
```





Summary

In [66]:
```
#ref: NaiveBays Assignment
column = ['Vectorizer', 'Model',  'Train_AUC', 'Test_AUC']
row1 = ['Tfidf', 'GBDT', 0.63, 0.55]
row2 = ['TfidfW2V', 'GBDT', 0.72, 0.709]
```

In [67]:
```
summary = pd.DataFrame(data = [row1, row2], columns = column, index = None)
```

In [68]:
```
import tabulate as tb
print(tb.tabulate(summary, headers='keys', tablefmt='psql'))
```

```
+----+--------------+---------+-------------+------------+
|    | Vectorizer   | Model   |   Train_AUC |   Test_AUC |
|----+--------------+---------+-------------+------------|
|  0 | Tfidf        | GBDT    |        0.63 |      0.55  |
|  1 | TfidfW2V     | GBDT    |        0.72 |      0.709 |
+----+--------------+---------+-------------+------------+
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js