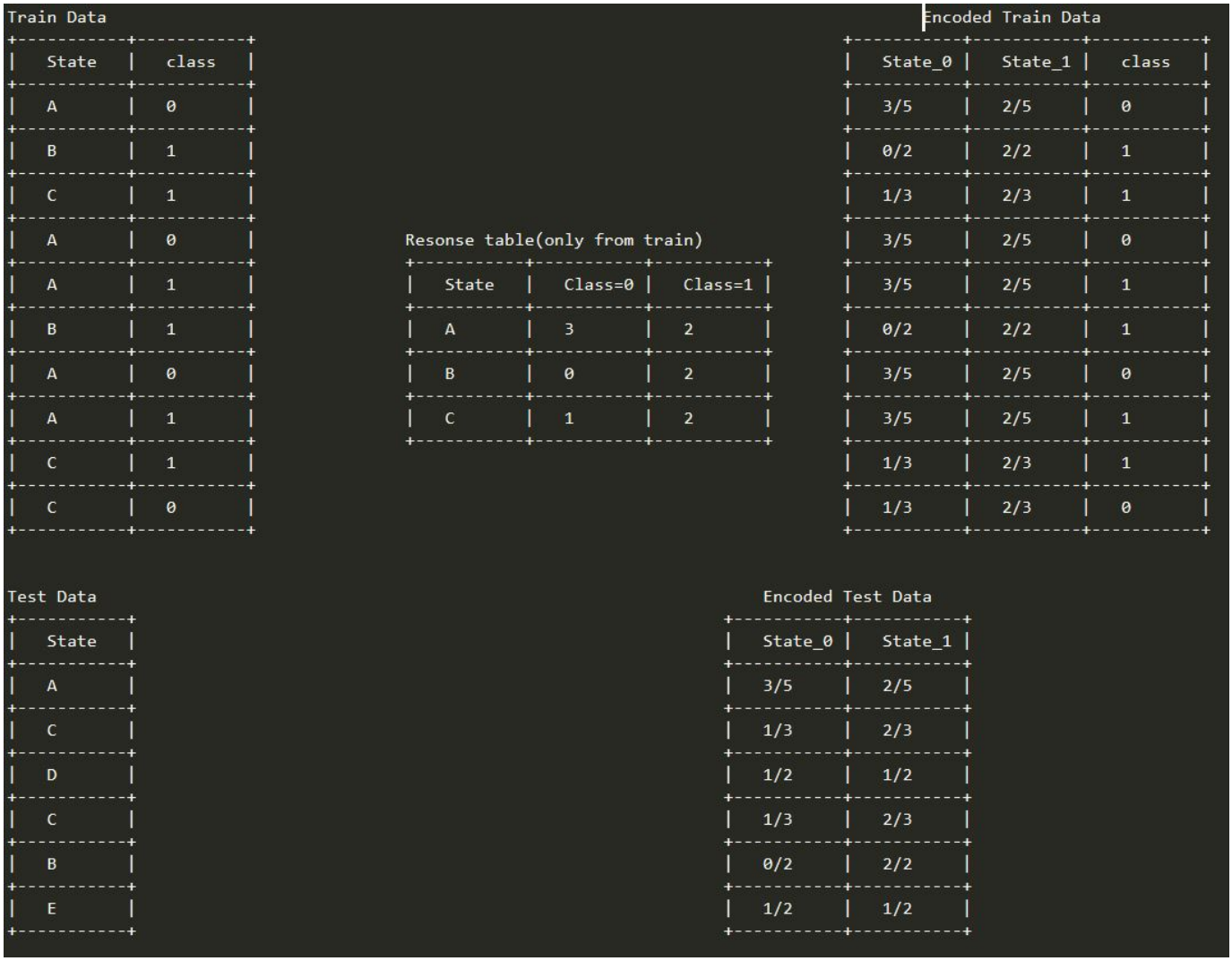
Assignment 9: GBDT

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

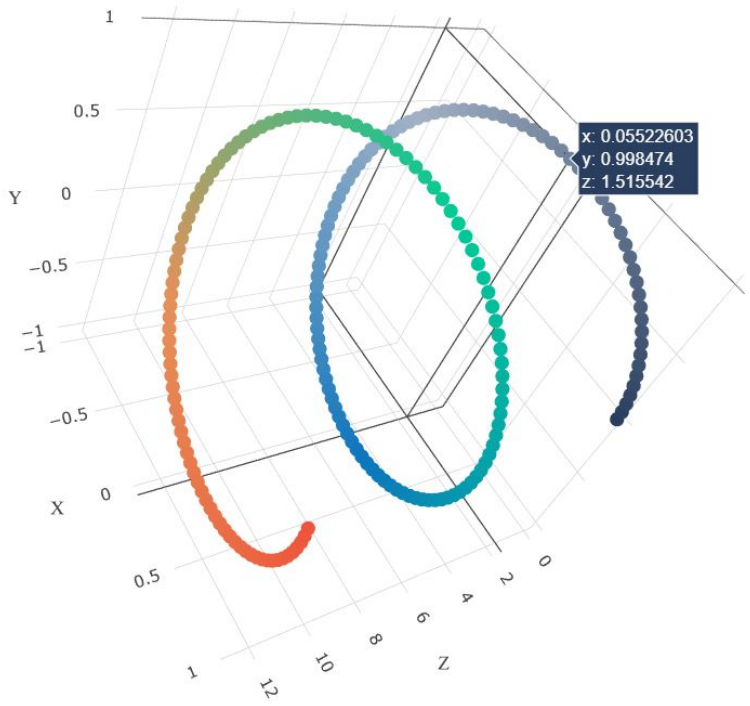
Encoded Test Data

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)

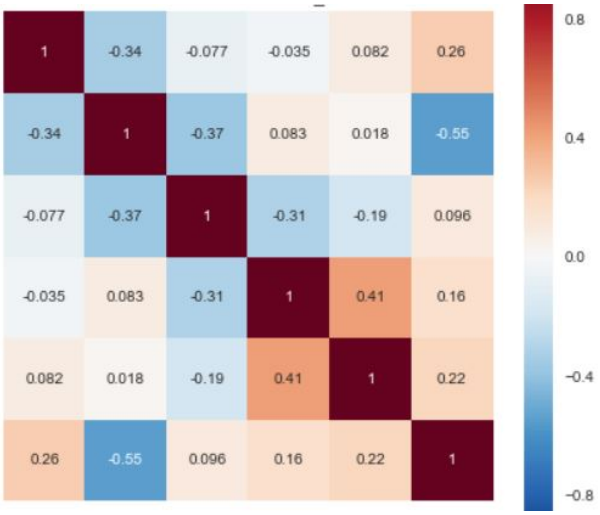
2. **The hyper paramter tuning (Consider any two hyper parameters)**
    - Find the best hyper parameter which will give the maximum AUC value
    - find the best hyper paramter using k-fold cross validation/simple cross validation data
    - use gridsearch cv or randomsearch cv or you can write your 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*

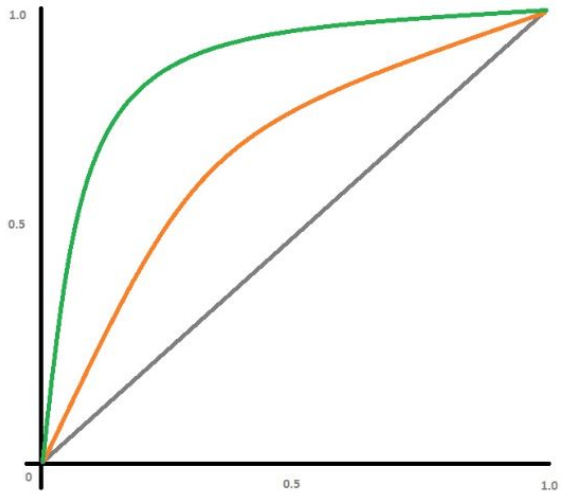
or

    - 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

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-package
```

```
!pip install plotly --upgrade
```

```
Collecting plotly
  Downloading https://files.pythonhosted.org/packages/68/47/cec583df9ffb6142b
     |████████████████████████████████| 13.1MB 239kB/s
Requirement already satisfied, skipping upgrade: retrying>=1.3.3 in /usr/loca
Requirement already satisfied, skipping upgrade: six in /usr/local/lib/python
Installing collected packages: plotly
  Found existing installation: plotly 4.4.1
    Uninstalling plotly-4.4.1:
      Successfully uninstalled plotly-4.4.1
Successfully installed plotly-4.11.0
```

```
path = '/content/drive/My Drive/AAIC/ASSIGN_13/11_Donors_choose_GBDT/'
```

Task 1

1.1 Loading Data

```
import pandas
data = pandas.read_csv(path+'preprocessed_data.csv',nrows=50000)
```

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

	school_state	teacher_prefix	project_grade_category	teacher_number_of_pr
0	ca	mrs	grades_prek_2	

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

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

import re
from tqdm import tqdm
```

1.1.0 Adding Sentiment analysis features such as NEG, POS, NEU and COMPOUND polarity scores

```
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

sid = SentimentIntensityAnalyzer()
NEG = []
NEU = []
POS = []
COMP = []
essays = data['essay'].values
for senetence in essays:
  ss = sid.polarity_scores(senetence)
  NEG.append(ss['neg'])
  POS.append(ss['pos'])
  NEU.append(ss['neu'])
  COMP.append(ss['compound'])

df2 = pd.DataFrame({'NEG':NEG,
                    'POS':POS,
                    'NEU':NEU,
                    'COMPOUND':COMP})

```

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

```
df2.head(2)
```

	NEG	POS	NEU	COMPOUND
0	0.013	0.205	0.783	0.9867
1	0.072	0.248	0.680	0.9897

```
data = pd.concat([data, df2], axis=1)
data.head(2)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_pr
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	

1.1.1 Processing Project Essay.

```
def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

```
#https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'it
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'becau
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',
```

```
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'a
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'tha
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'm
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
            'won', "won't", 'wouldn', "wouldn't"]
```

```
# Combining all the above stundents
from tqdm import tqdm
def preprocess_text(text_data):
    preprocessed_text = []
    # tqdm is for printing the status bar
    for sentance in tqdm(text_data):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\n', ' ')
        sent = sent.replace('\\"', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
        preprocessed_text.append(sent.lower().strip())
    return preprocessed_text
```

```
data.essay.iloc[:1]
```

```
0    i fortunate enough use fairy tale stem kits cl...
Name: essay, dtype: object
```

As there are some stop words present in the text of essay of already preprocessed_data. For example word `**i**' is present in the first essay sentence. So, I am processing the essay text again inorder to remove the stop words if there are any by chance and also because unnecessary stop words increase the dimensionality of the bow/tfidf representation.

```
data['essay'] = preprocess_text(data['essay'].values)
```

```
100%|██████████| 50000/50000 [00:20<00:00, 2484.92it/s]
```

```
data.essay.iloc[:1]
```

```
0    fortunate enough use fairy tale stem kits clas...
Name: essay, dtype: object
```

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

	school_state	teacher_prefix	project_grade_category	teacher_number_of_pr

1.1.2 Loading data after processing essay

```
Y=data['project_is_approved']
X=data.drop('project_is_approved',axis=1)
```

```
X.head(1)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_pr
0	ca	mrs	grades_prek_2	

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

```
from sklearn.model_selection import train_test_split

X_train,X_te,Y_train,Y_te = train_test_split(X,Y,test_size=0.33, stratify=Y,
```

```
print(X_train.shape,Y_train.shape,X_te.shape,Y_te.shape,end=" ")
```

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

1.3 Make Data Model Ready: encoding project essay

1.3.1 TFIDF Representation of essay.

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4))
vectorizer.fit(X_train['essay'].values)
```

```
    TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                    dtype=<class 'numpy.float64'>, encoding='utf-8',
```

```

essay_tr_tfidf = vectorizer.transform(X_train['essay'].values)
essay_te_tfidf = vectorizer.transform(X_te['essay'].values)

```

```
print(essay_tr_tfidf.shape,essay_te_tfidf.shape,end=" ")
```

```
(33500, 99859) (16500, 99859)
```

1.3.2 TFIDF-W2V Representation of essay.

```
import pickle
with open(path+'glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

```
# average Word2Vec
# compute average word2vec for each review.
def tfidf_w2v_transform(preprocessed_essays):
  tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in
  for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/revie
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf v
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

  return tfidf_w2v_vectors

```

Train data

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer(min_df=10,ngram_range=(1,4))
tfidf_model.fit(X_train['essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

```
essay_tr_tfidf_w2v = np.array(tfidf_w2v_transform(X_train['essay']))
```

```
essay_te_tfidf_w2v = np.array(tfidf_w2v_transform(X_te['essay']))
#print(preprocessed_essays[:3])
```

```
100%|██████████| 33500/33500 [01:05<00:00, 511.80it/s]
100%|██████████| 16500/16500 [00:32<00:00, 508.49it/s]
```

```
print(essay_tr_tfidf_w2v.shape,essay_te_tfidf_w2v.shape,end=" ")
```

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

1.4 Make Data Model Ready: encoding numerical, categorical features

1.3.3 Encoding Categorical features

```
data.columns
```

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price', 'NEG',
       'POS', 'NEU', 'COMPOUND'],
      dtype='object')
```

School State

```
df = pd.DataFrame({'ftr' : X_train['school_state'], 'label' : Y_train})
sam_df = pd.DataFrame(columns = ['prob_0', 'prob_1'],index =  set(df['ftr']) )
for i in set(df['ftr']):
  sam_df.loc[i].prob_0 = len(df[(df['ftr'] == (i)) & (df['label'] == 0)])/ len
  sam_df.loc[i].prob_1 = len(df[(df['ftr'] == (i)) & (df['label'] == 1)])/ len
```

```
cat_data_tr = pd.DataFrame(columns= ['prob_0','prob_1'],index = df.index)
for i in list(df.index.values):
  j = df.loc[i][['ftr']].values
  #print(j[0])
  cat_data_tr.loc[i] = {'prob_0':sam_df.loc[j[0]].prob_0 , 'prob_1':sam_df.loc
cat_data_tr= cat_data_tr.values
print(cat_data_tr.shape)
```

```
(33500, 2)
```

```
df3 = X_te['school_state']
cat_data_te = pd.DataFrame(columns= ['prob_0','prob_1'],index = df3.index)
for i in (df3.index):
  if df3.loc[i] not in sam_df.index:
    cat_data_te.loc[i] = {'prob_0':0.5 , 'prob_1':0.5}
  else:
```

```
    cat_data_te.loc[i] = {'prob_0':sam_df.loc[df3.loc[i]].prob_0 , 'prob_1':sa
cat_data_te = cat_data_te.values
```

```
state_tr = cat_data_tr
state_te = cat_data_te
```

```
print(state_tr.shape,state_te.shape,end=" ")
```

```
(33500, 2) (16500, 2)
```

Teacher prefix

```
df = pd.DataFrame({'ftr' : X_train['teacher_prefix'], 'label' : Y_train})
sam_df = pd.DataFrame(columns = ['prob_0', 'prob_1'],index =  set(df['ftr']) )
for i in set(df['ftr']):
  sam_df.loc[i].prob_0 = len(df[(df['ftr'] == (i)) & (df['label'] == 0)])/ len
  sam_df.loc[i].prob_1 = len(df[(df['ftr'] == (i)) & (df['label'] == 1)])/ len
```

```
cat_data_tr = pd.DataFrame(columns= ['prob_0','prob_1'],index = df.index)
for i in list(df.index.values):
  j = df.loc[i][['ftr']].values
  #print(j[0])
  cat_data_tr.loc[i] = {'prob_0':sam_df.loc[j[0]].prob_0 , 'prob_1':sam_df.loc
cat_data_tr= cat_data_tr.values
print(cat_data_tr.shape)
```

```
(33500, 2)
```

```
df3 = X_te['teacher_prefix']
cat_data_te = pd.DataFrame(columns= ['prob_0','prob_1'],index = df3.index)
for i in (df3.index):
  if df3.loc[i] not in sam_df.index:
    cat_data_te.loc[i] = {'prob_0':0.5 , 'prob_1':0.5}
  else:
    cat_data_te.loc[i] = {'prob_0':sam_df.loc[df3.loc[i]].prob_0 , 'prob_1':sa
cat_data_te = cat_data_te.values
```

```
tchr_prefix_tr = cat_data_tr
tchr_prefix_te = cat_data_te
```

```
print(tchr_prefix_tr.shape,tchr_prefix_te.shape,end=" ")
```

```
(33500, 2) (16500, 2)
```

Project grade_category

```
df = pd.DataFrame({'ftr' : X_train['project_grade_category'], 'label' : Y_tra
sam_df = pd.DataFrame(columns = ['prob_0', 'prob_1'],index =  set(df['ftr']) )
```

```
sam_df = pd.DataFrame(columns = ['prob_0', 'prob_1'],index = set(df['ftr']))
for i in set(df['ftr']):
  sam_df.loc[i].prob_0 = len(df[(df['ftr'] == (i)) & (df['label'] == 0)])/ len
  sam_df.loc[i].prob_1 = len(df[(df['ftr'] == (i)) & (df['label'] == 1)])/ len
```

```
cat_data_tr = pd.DataFrame(columns= ['prob_0','prob_1'],index = df.index)
for i in list(df.index.values):
  j = df.loc[i][['ftr']].values
  #print(j[0])
  cat_data_tr.loc[i] = {'prob_0':sam_df.loc[j[0]].prob_0 , 'prob_1':sam_df.loc
cat_data_tr= cat_data_tr.values
print(cat_data_tr.shape)
```

```
(33500, 2)
```

```
df3 = X_te['project_grade_category']
cat_data_te = pd.DataFrame(columns= ['prob_0','prob_1'],index = df3.index)
for i in (df3.index):
  if df3.loc[i] not in sam_df.index:
    cat_data_te.loc[i] = {'prob_0':0.5 , 'prob_1':0.5}
  else:
    cat_data_te.loc[i] = {'prob_0':sam_df.loc[df3.loc[i]].prob_0 , 'prob_1':sa
cat_data_te = cat_data_te.values
```

```
grade_tr = cat_data_tr
grade_te = cat_data_te
```

```
print(grade_tr.shape,grade_te.shape,end=" ")
```

```
(33500, 2) (16500, 2)
```

Subject categories

```
df = pd.DataFrame({'ftr' : X_train['clean_categories'], 'label' : Y_train})
sam_df = pd.DataFrame(columns = ['prob_0', 'prob_1'],index =  set(df['ftr']) )
for i in set(df['ftr']):
  sam_df.loc[i].prob_0 = len(df[(df['ftr'] == (i)) & (df['label'] == 0)])/ len
  sam_df.loc[i].prob_1 = len(df[(df['ftr'] == (i)) & (df['label'] == 1)])/ len
```

```
cat_data_tr = pd.DataFrame(columns= ['prob_0','prob_1'],index = df.index)
for i in list(df.index.values):
  j = df.loc[i][['ftr']].values
  #print(j[0])
  cat_data_tr.loc[i] = {'prob_0':sam_df.loc[j[0]].prob_0 , 'prob_1':sam_df.loc
cat_data_tr= cat_data_tr.values
print(cat_data_tr.shape)
```

```
(33500, 2)
```

```
df3 = X_te['clean_categories']
```

```
cat_data_te = pd.DataFrame(columns= ['prob_0','prob_1'],index = df3.index)
for i in (df3.index):
  if df3.loc[i] not in sam_df.index:
    cat_data_te.loc[i] = {'prob_0':0.5 , 'prob_1':0.5}
  else:
    cat_data_te.loc[i] = {'prob_0':sam_df.loc[df3.loc[i]].prob_0 , 'prob_1':sa
cat_data_te = cat_data_te.values
```

```
subject_tr = cat_data_tr
subject_te = cat_data_te
```

```
print(subject_tr.shape,subject_te.shape,end=" ")
```

```
(33500, 2) (16500, 2)
```

Subject sub categories

```
df = pd.DataFrame({'ftr' : X_train['clean_subcategories'], 'label' : Y_train})
sam_df = pd.DataFrame(columns = ['prob_0', 'prob_1'],index =  set(df['ftr']) )
for i in set(df['ftr']):
  sam_df.loc[i].prob_0 = len(df[(df['ftr'] == (i)) & (df['label'] == 0)])/ len
  sam_df.loc[i].prob_1 = len(df[(df['ftr'] == (i)) & (df['label'] == 1)])/ len
```

```
print(df.shape,sam_df.shape)
```

```
(33500, 2) (342, 2)
```

```
cat_data_tr = pd.DataFrame(columns= ['prob_0','prob_1'],index = df.index)
for i in list(df.index.values):
  j = df.loc[i][['ftr']].values
  #print(j[0])
  cat_data_tr.loc[i] = {'prob_0':sam_df.loc[j[0]].prob_0 , 'prob_1':sam_df.loc
cat_data_tr= cat_data_tr.values
print(cat_data_tr.shape)
```

```
(33500, 2)
```

```
df3 = X_te['clean_subcategories']
cat_data_te = pd.DataFrame(columns= ['prob_0','prob_1'],index = df3.index)
for i in (df3.index):
  if df3.loc[i] not in sam_df.index:
    cat_data_te.loc[i] = {'prob_0':0.5 , 'prob_1':0.5}
  else:
    cat_data_te.loc[i] = {'prob_0':sam_df.loc[df3.loc[i]].prob_0 , 'prob_1':sa
cat_data_te = cat_data_te.values
```

```
print(df3.shape,cat_data_te.shape)
```

```
(16500,) (16500, 2)
```

```
subj_subct_tr = cat_data_tr
subj_subct_te = cat_data_te
```

```
print(subj_subct_tr.shape,subj_subct_te.shape,end=" ")
```

```
(33500, 2) (16500, 2)
```

1.3.4 Encoding Numerical features

teacher_number_of_previously_posted_projects

```
#For z-score normalization the values will be both negative and positive. Mult
#So we use Min-Max normalization.
#from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(X_train['teacher_number_of_previously_posted_projects'].values.resh

tchr_prj_tr = scaler.transform(X_train['teacher_number_of_previously_posted_pr
tchr_prj_te = scaler.transform(X_te['teacher_number_of_previously_posted_proje
```

```
print(list(tchr_prj_tr[:5]),'\n',list(tchr_prj_te[:5]))
```

```
[array([0.00561798]), array([0.09269663]), array([0.02808989]), array([0.0056
 [array([0.00842697]), array([0.00561798]), array([0.01966292]), array([0.036
```

```
#from sklearn.preprocessing import MinMaxScaler

#scaler = MinMaxScaler()
#scaler.fit(data['teacher_number_of_previously_posted_projects'].values.reshap
#data['teacher_number_of_previously_posted_projects']=scaler.transform(data['t

```

price

```
# Min-Max Normalization

scaler = MinMaxScaler()
scaler.fit(X_train['price'].values.reshape(-1, 1))

price_tr = scaler.transform(X_train['price'].values.reshape(-1, 1) )
price_te = scaler.transform(X_te['price'].values.reshape(-1, 1) )
```

```
print(list(price_tr[:5]),'\n',list(price_te[:5]))
```

```
[array([0.00185336]), array([0.07942025]), array([0.03560584]), array([0.0789
 [array([0.07126968]), array([0.00790852]), array([0.02513283]), array([0.021
```

```
# Min-Max Normalization Use only one Normalization but not both.

#scaler = MinMaxScaler()
#scaler.fit(data['price'].values.reshape(-1, 1))
#data['price']=scaler.transform(data['price'].values.reshape(-1, 1))
```

1.4 Stacking all features to form set1 and set2 features.

```
data.columns
```

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price', 'NEG',
       'POS', 'NEU', 'COMPOUND'],
      dtype='object')
```

1.4.1. Set1(TFIDF) features

```
from scipy.sparse import hstack

X_train_tfidf = hstack((essay_tr_tfidf, state_tr.astype('float64'), tchr_prefi
X_te_tfidf = hstack((essay_te_tfidf, state_te.astype('float64'), tchr_prefix_t

print("Final Data matrix")
print(X_train_tfidf.shape, Y_train.shape)
print(X_te_tfidf.shape, Y_te.shape)
print("="*100)
```

```
Final Data matrix
(33500, 99875) (33500,)
(16500, 99875) (16500,)
====================================================================================
```

1.4.2 Set2(TFIDF-W2V) features

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

X_train_tfidf_w2v = hstack((csr_matrix(essay_tr_tfidf_w2v), state_tr.astype('f
X_te_tfidf_w2v = hstack((csr_matrix(essay_te_tfidf_w2v), state_te.astype('floa

print("Final Data matrix")
print(X_train_tfidf_w2v.shape, Y_train.shape)
print(X_te_tfidf_w2v.shape, Y_te.shape)
print("="*100)
```

```
Final Data matrix
(33500, 316) (33500,)
(16500, 316) (16500,)
==========================================================================
```

GBDT (xgboost/lightgbm)

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

Apply GBDT on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instructions

2. Hyper parameter Tuning

2.0 functions for plotly plotting

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

```
#https://stackoverflow.com/questions/47230817/plotly-notebook-mode-with-google
def enable_plotly_in_cell():
  import IPython
  from plotly.offline import init_notebook_mode
  display(IPython.core.display.HTML('''<script src="/static/components/require
  init_notebook_mode(connected=False)
```

2.1 Set1 (TF-IDF) Features

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV

depth = [3, 4, 5] # 6, 7]
no_of_est = [100, 110, 120] #, 130, 140, 150]
hyper_param = {'max_depth' : depth, 'n_estimators' : no_of_est }

clf = xgb.XGBClassifier(booster = 'gbtree',objective = 'binary:logistic', lear
classifier = GridSearchCV(clf, hyper_param, cv=5,return_train_score=True, scor
```

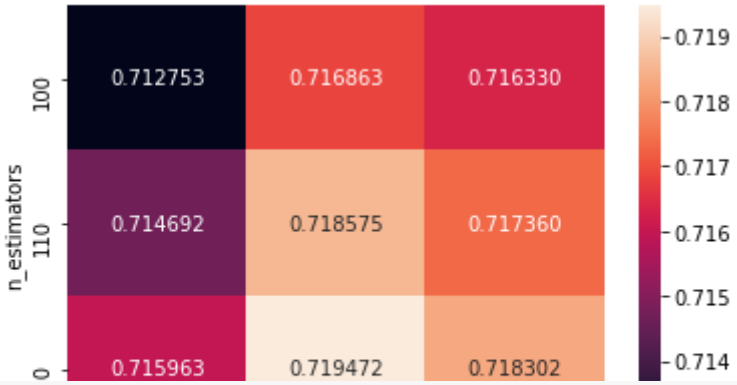
```
classifier.fit(X_train_tfidf, Y_train)

results = pd.DataFrame.from_dict(classifier.cv_results_)
```

```
results = results.sort_values(['rank_test_score'])
results.head()
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_dep
5	230.624216	0.996491	1.094910	0.011630	
4	214.412764	1.369309	1.105326	0.016504	
8	268.062441	38.728753	1.016270	0.231020	
7	263.698027	1.218970	1.109874	0.034966	
3	195.380259	1.210561	1.115658	0.023528	

```
train_auc= results['mean_train_score']
cv_auc = results['mean_test_score']
depth =  results['param_max_depth']
no_of_est = results['param_n_estimators']
# we need not generate mesh grid explicitly as depth and min_samples combined
#print(depth)
#print(no_of_est)
```

```
import seaborn as sns
cv_auc_values = pd.DataFrame({'max_depth':depth,
                              'n_estimators': no_of_est,
                              'auc': cv_auc})
cv_auc_values = cv_auc_values.pivot("n_estimators","max_depth", "auc")
sns.heatmap(cv_auc_values, annot=True, fmt="f")
plt.show()
```




```
results = results.sort_values(['rank_test_score'])
best_depth = int(results[results['rank_test_score'] == 1].param_max_depth)
best_n_estimators = int(results[results['rank_test_score'] == 1].param_n_estir
results.head(2)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_dep
5	230.624216	0.996491	1.094910	0.011630	
4	214.412764	1.369309	1.105326	0.016504	

```
print('GBDT having depth {} and no of estimators(boosting stages) {} gives the
```

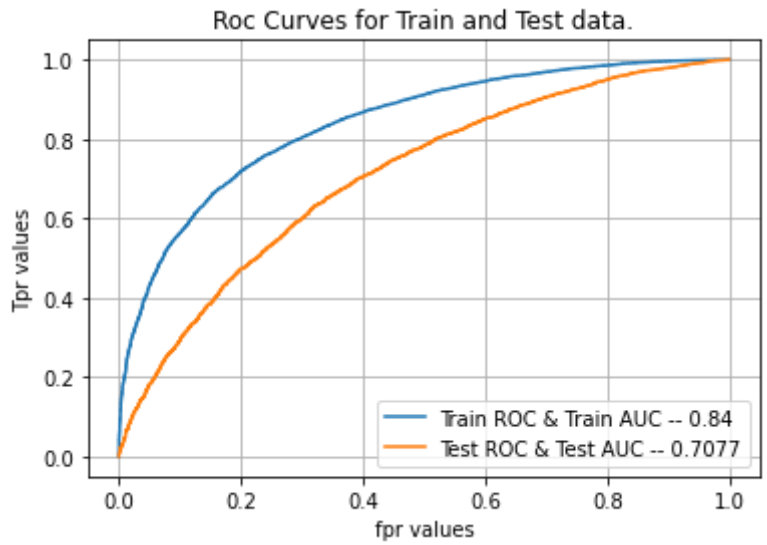
```
GBDT having depth 4 and no of estimators(boosting stages) 120 gives the best
```

From the above plot and table, we can observe that best hyper parameters are max_depth = 10 and n_estimators = 500 and we take these values for testing.

2.1.1 Testing

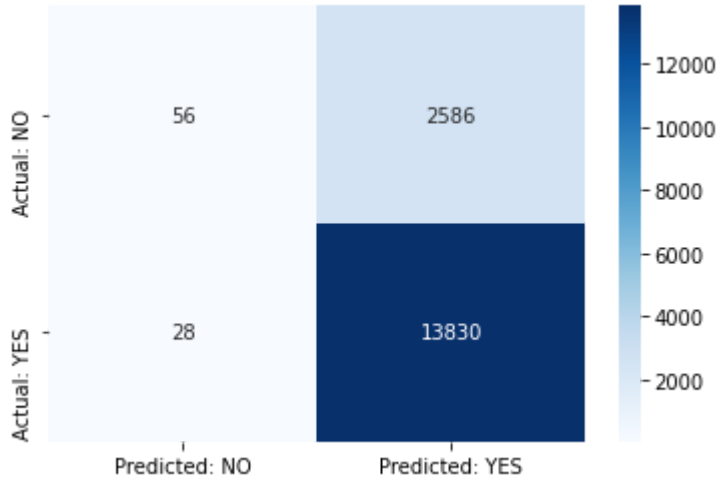
```
from sklearn.metrics import roc_curve,roc_auc_score,confusion_matrix

clf = xgb.XGBClassifier(max_depth = best_depth, n_estimators = best_n_estimato
clf.fit(X_train_tfidf,Y_train)

y_prob_tr = clf.predict_proba(X_train_tfidf)[:,1]
fpr_tr, tpr_tr, thresholds_tr = roc_curve(Y_train,y_prob_tr)
plt.plot(fpr_tr, tpr_tr)

y_prob_te = clf.predict_proba(X_te_tfidf)[:,1]
fpr_te, tpr_te, thresholds_te = roc_curve(Y_te,y_prob_te)
plt.plot(fpr_te, tpr_te)

AUC_tr_tfidf = roc_auc_score(Y_train, y_prob_tr)
```

```
AUC_te_tfidf = roc_auc_score(Y_te, y_prob_te)

plt.grid()
plt.legend( ('Train ROC & Train AUC -- '+str(np.round(AUC_tr_tfidf, decimals=4
plt.xlabel('fpr values')
plt.ylabel('Tpr values')
plt.title('Roc Curves for Train and Test data.')
plt.show()

```




```
y_pred_tfidf=clf.predict(X_te_tfidf)
cnf_mtrx_tfidf = confusion_matrix(Y_te, y_pred_tfidf)

print('Confusion matrix for the Test data.')
#print(cnf_mtrx_tfidf)
x=pd.DataFrame(cnf_mtrx_tfidf, columns = ["Predicted: NO","Predicted: YES"], i
sns.heatmap(x, annot=True,fmt="d", cmap='Blues')
plt.show()
```

```
Confusion matrix for the Test data.
```



2.2 Set2(TF-IDF Weighted W2V) Features dup

```
import xgboost as xgb
```

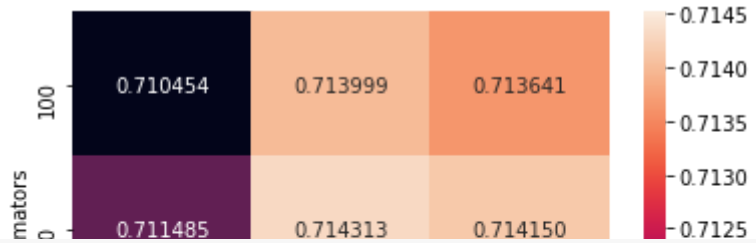
```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV

depth = [3, 4, 5] #, 6, 7]
no_of_est = [100, 110, 120] #, 130, 140, 150]
hyper_param = {'max_depth' : depth, 'n_estimators' : no_of_est }

clf = xgb.XGBClassifier(booster = 'gbtree',objective = 'binary:logistic', lear

classifier = GridSearchCV(clf, hyper_param, cv=5,return_train_score=True, scor
classifier.fit(X_train_tfidf_w2v, Y_train)

results = pd.DataFrame.from_dict(classifier.cv_results_)
```

```
results = results.sort_values(['rank_test_score'])
results.head(2)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_dep
8	208.193783	27.187026	1.374231	0.332773	
4	163.315378	0.197586	1.507982	0.022203	

```
train_auc= results['mean_train_score']
cv_auc = results['mean_test_score']
depth =  results['param_max_depth']
no_of_est = results['param_n_estimators']
# we need not generate mesh grid explicitly as depth and min_samples combined
#print(depth)
#print(no_of_est)
```

```
import seaborn as sns
cv_auc_values = pd.DataFrame({'max_depth':depth,
                              'n_estimators': no_of_est,
                              'auc': cv_auc})
cv_auc_values = cv_auc_values.pivot("n_estimators","max_depth", "auc")
sns.heatmap(cv_auc_values, annot=True, fmt="f")
plt.show()
```




```
results = results.sort_values(['rank_test_score'])
best_depth_tfidf_w2v = int(results[results['rank_test_score'] == 1].param_max_
best_n_estimators_tfidf_w2v = int(results[results['rank_test_score'] == 1].par
results.head(2)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_dep
8	208.193783	27.187026	1.374231	0.332773	
4	163.315378	0.197586	1.507982	0.022203	

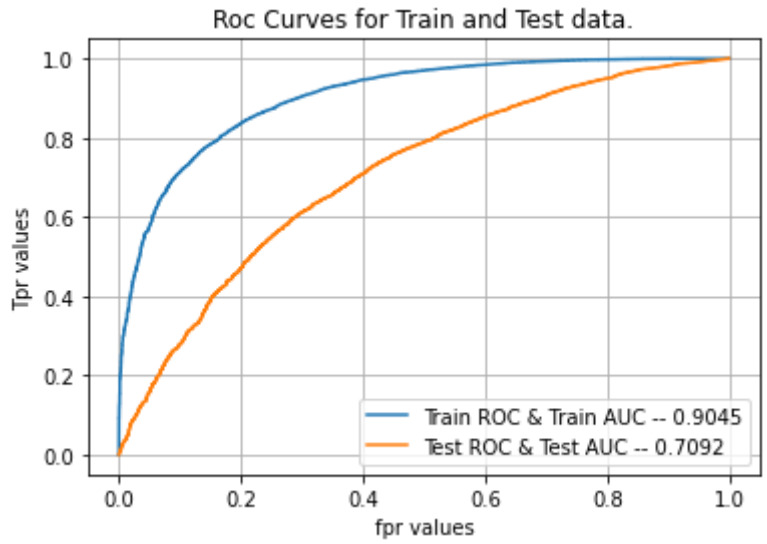
```
print('GBDT having depth {} and no of estimators(boosting stages) {} gives the
```

```
GBDT having depth 5 and no of estimators(boosting stages) 120 gives the best
```

2.2.1 Testing

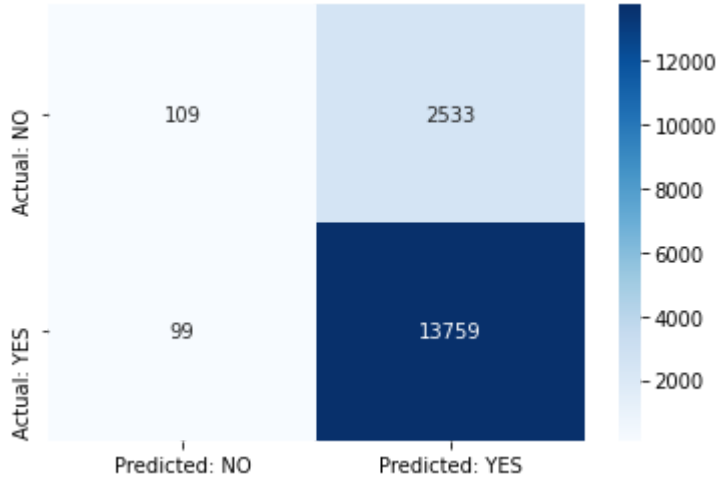
```
from sklearn.metrics import roc_curve,roc_auc_score,confusion_matrix

clf = xgb.XGBClassifier(max_depth = best_depth_tfidf_w2v, n_estimators = best_
clf.fit(X_train_tfidf_w2v,Y_train)

y_prob_tr = clf.predict_proba(X_train_tfidf_w2v)[:,1]
fpr_tr, tpr_tr, thresholds_tr = roc_curve(Y_train,y_prob_tr)
plt.plot(fpr_tr, tpr_tr)

y_prob_te = clf.predict_proba(X_te_tfidf_w2v)[:,1]
fpr_te, tpr_te, thresholds_te = roc_curve(Y_te,y_prob_te)
plt.plot(fpr_te, tpr_te)

AUC_tr_tfidf_w2v = roc_auc_score(Y_train, y_prob_tr)
AUC_te_tfidf_w2v = roc_auc_score(Y_te, y_prob_te)

plt.grid()
plt.legend( ('Train ROC & Train AUC -- '+str(np.round(AUC_tr_tfidf_w2v, decima
plt.xlabel('fpr values')
plt.ylabel('Tpr values')
plt.title('Roc Curves for Train and Test data.')
plt.show()
```

```
23
```




```
y_pred_tfidf_w2v=clf.predict(X_te_tfidf_w2v)
cnf_mtrx_tfidf_w2v = confusion_matrix(Y_te, y_pred_tfidf_w2v)

print('Confusion matrix for the Test data.')
#print(cnf_mtrx_tfidf)
x=pd.DataFrame(cnf_mtrx_tfidf_w2v, columns = ["Predicted: NO","Predicted: YES"
sns.heatmap(x, annot=True,fmt="d", cmap='Blues')
plt.show()
```

```
Confusion matrix for the Test data.
```



3. Summary

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

x = PrettyTable()
x.field_names = ["Vectorizer","Model", "Hyper parameters (depth and no of est
for i in range(1):
    x.add_row(['TFIDF', "GBDT", (best_depth,best_n_estimators) , np.round(AUC
```

```
    x.add_row(["TFIDF-W2V", "GBDT" , (best_depth_tfidf_w2v,best_n_estimators_t
print(x)
```

```
+------------+-------+-------------------------------------------------+-------
| Vectorizer | Model | Hyper parameters (depth and no of estimators) | Train
+------------+-------+-------------------------------------------------+-------
|   TFIDF    |  GBDT |                   (4, 120)                      |    0.8
| TFIDF-W2V  |  GBDT |                   (5, 120)                      |   0.90
+------------+-------+-------------------------------------------------+-------
```