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
> encode them with default values Ex: in our test data if have State: D then we encode it as [0.5,

1. **Apply GBDT on these feature sets**
    - Set 1: categorical(instead of one hot encoding, try [response coding](#): use probability val project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check features)
    - Set 2: categorical(instead of one hot encoding, try [response coding](#): use probability val 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 ta

3. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data

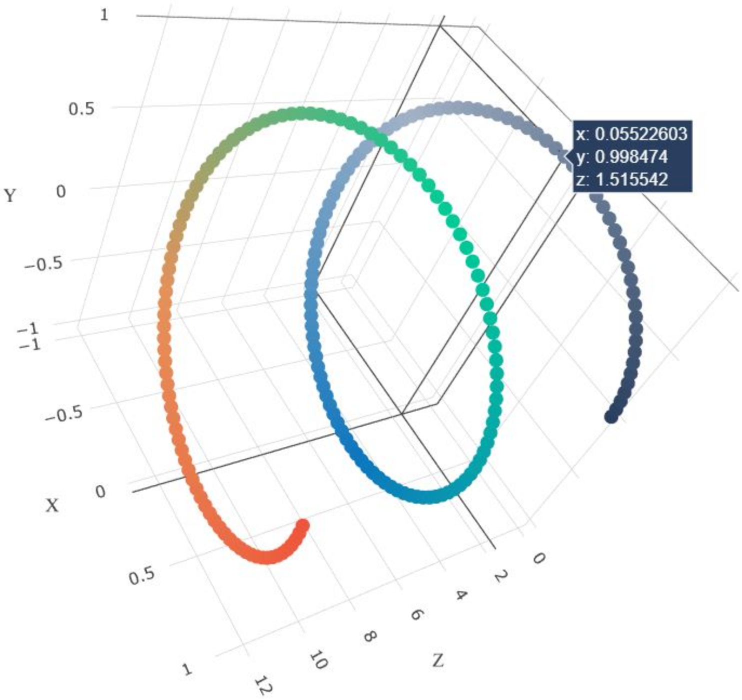
     

     figure with X-ax
     Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d p
     3d_scatter_plot.ipynb

     or

   - You need to plot the performance of model both on train data and cross validation data

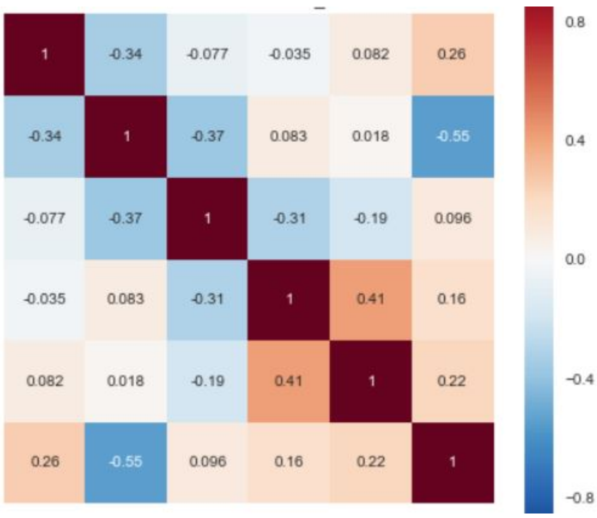
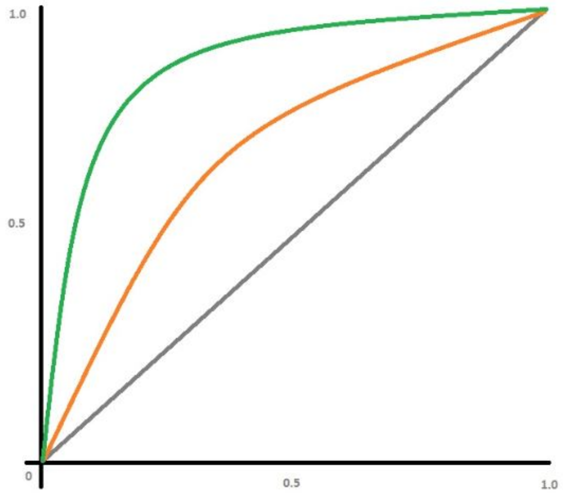
     

     figure [seaborn heat maps](#) with rows as **n_es**
     inside the cell representing **AUC Score**
   - You choose either of the plotting techniques out of 3d plot or heat map

- Once after you found the best hyper parameter, you need to train your model with it, an



curve on both train and test.

- Along with plotting ROC curve, you need to print the [confusion matrix](https://en.wikipedia.org/wiki/Confusion_matrix) with predicted an

	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 form

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

1. GBDT (xgboost/lightgbm)

```
!pip install chart_studio
```

```
Collecting chart_studio
  Downloading https://files.pythonhosted.org/packages/ca/ce/330794a6b6ca4b9182c38fc69
     |████████████████████████████████| 71kB 4.3MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from ch
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packa
```

/nlmq4/1591844175000/00484516897554883881/03543900857199698311/1lDca_ge-GYO0iQ6_XDLWePQFMdA

```
--2020-06-11 02:58:02--  https://doc-0k-0g-docs.googleusercontent.com/docs/securesc/4
Resolving doc-0k-0g-docs.googleusercontent.com (doc-0k-0g-docs.googleusercontent.com)
Connecting to doc-0k-0g-docs.googleusercontent.com (doc-0k-0g-docs.googleusercontent.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/octet-stream]
Saving to: ‘glove_vectors’

glove_vectors           [       <=>          ] 121.60M  44.4MB/s    in 2.7s

2020-06-11 02:58:05 (44.4 MB/s) - ‘glove_vectors’ saved [127506004]
```

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

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

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

import pickle
from tqdm import tqdm
import os

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

1.1 Loading Data

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

```
data  = pd.read_csv('preprocessed_data.csv', nrows=50000)
data.head(2)
```

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

```
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
neg=[]
pos=[]
neu=[]
compound =[]
for  for_sentiment in data["essay"]:
  ss = sid.polarity_scores(for_sentiment)
  neg.append(ss["neg"])
  pos.append(ss["pos"])
  neu.append(ss["neu"])
  compound.append(ss["compound"])
```

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

```
data['neg'] =neg
data['pos'] = pos
data['neu'] = neu
data['compound'] = compound
```

```
# separating y from dataframe
y = data['project_is_approved'].values
X = data
X.head(2)
```

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

```
# separating data into train and test.
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)
```

1.3 Make Data Model Ready: encoding eassay, and project_title

TFIDF vectorizer

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)
vectorizer_tfidf = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer_tfidf.fit(X_train['essay'].values)
X_train_eassy_tfidf = vectorizer_tfidf.fit_transform(X_train['essay'].values)
X_test_eassy_tfidf = vectorizer_tfidf.fit_transform(X_test['essay'].values)
print("After vectorizations")
print(X_train_eassy_tfidf.shape, y_train.shape)
print(X_test_eassy_tfidf.shape, y_test.shape)
print("="*100)
```

```
(33500, 13) (33500,)
(16500, 13) (16500,)
====================================================================================
After vectorizations
(33500, 5000) (33500,)
(16500, 5000) (16500,)
====================================================================================
```

TFIDF weighted W2V

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pick
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())


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


# average Word2Vec
# compute average word2vec for each review.
def tf_idf_done(word_list):
  # average Word2Vec
  # compute average word2vec for each review.
  tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
  for sentence in tqdm(word_list): # for each review/sentence
      vector = np.zeros(300) # as word vectors are of zero length
      tf_idf_weight =0; # num of words with a valid vector in the sentence/review
      for word in sentence.split(): # for each word in a review/sentence
          if (word in glove_words) and (word in tfidf_words):
              vec = model[word] # getting the vector for each word
              # here we are multiplying idf value(dictionary[word]) and the tf value((sent
              tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # get
              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_tfidf_w2v_essays=tf_idf_done(X_train['essay'].values)
test_tfidf_w2v_essays=tf_idf_done(X_test['essay'].values)
```

```
100%|██████████| 33500/33500 [01:08<00:00, 487.43it/s]
100%|██████████| 16500/16500 [00:33<00:00, 487.33it/s]
```

```
train_tfidf_w2v_essays = np.array(train_tfidf_w2v_essays)
test_tfidf_w2v_essays  = np.array(test_tfidf_w2v_essays)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)
print("After vectorizations")
print(train_tfidf_w2v_essays.shape,y_train.shape)
print(test_tfidf_w2v_essays.shape,y_test.shape)
print("="*100)
```

```
(33500, 13) (33500,)
(16500, 13) (16500,)
==================================================================================
After vectorizations
(33500, 300) (33500,)
(16500, 300) (16500,)
==================================================================================
```

1.4 Make Data Model Ready: encoding numerical, categorical featur

response coding

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature:categorical features
# df: ['X_train', 'X_test']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train data d
# build a vector (1*2) , the first element = (number of times it occured in class1 + 10*al
# gv_dict is like a look up table, for every gene it store a (1*2) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/2, 1/2] to 'gv_fea'
# return 'gv_fea'
# ----------------------

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    value_count = X_train[feature].value_counts()

    # gv_dict : Gene Variation Dict, which contains the probability array for each gene/va
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in whole
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticula
        # vec is 2 diamensional vector
        vec = []
        for k in range(1,3):

            cls_cnt = X_train.loc[(X_train['project_is_approved']==k) & (X_train[feature]=

            # cls_cnt.shape[0](numerator) will contain the number of time that particular
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 20*alpha))

        # we are adding the gene/variation to the dict as key and vec as value
```

```
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = X_train[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each feature value i
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the
    # if not we will add [1/2,1/2] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/2,1/2])
    return gv_fea
```

encoding categorical features: School State

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_school_state_feature_responseCoding = np.array(get_gv_feature(alpha, "school_state",
# test gene feature
test_school_state_feature_responseCoding = np.array(get_gv_feature(alpha, "school_state",


print("after encoding")
print(train_school_state_feature_responseCoding.shape)
print(test_school_state_feature_responseCoding.shape)
```

```
after encoding
(33500, 2)
(16500, 2)
```

encoding categorical features: teacher_prefix

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_teacher_prefix_feature_responseCoding = np.array(get_gv_feature(alpha, "teacher_pref
# test gene feature
test_teacher_prefix_feature_responseCoding = np.array(get_gv_feature(alpha, "teacher_prefi
###########
print("after encoding")
```

```
print(train_teacher_prefix_feature_responseCoding.shape)
print(test_teacher_prefix_feature_responseCoding.shape)
```

```
after encoding
(33500, 2)
(16500, 2)
```

encoding categorical features: project_grade_category

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_project_grade_category_feature_responseCoding = np.array(get_gv_feature(alpha, "proj
# test gene feature
test_project_grade_category_feature_responseCoding = np.array(get_gv_feature(alpha, "proje
###########
print("after encoding")
print(train_project_grade_category_feature_responseCoding.shape)
print(test_project_grade_category_feature_responseCoding.shape)
```

```
after encoding
(33500, 2)
(16500, 2)
```

encoding categorical features: clean_categories

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_clean_categories_feature_responseCoding = np.array(get_gv_feature(alpha, "clean_cate
# test gene feature
test_clean_categories_feature_responseCoding = np.array(get_gv_feature(alpha, "clean_categ
###########
print("after encoding")
print(train_clean_categories_feature_responseCoding.shape)
print(test_clean_categories_feature_responseCoding.shape)
```

```
after encoding
(33500, 2)
(16500, 2)
```

encoding categorical features: clean_subcategories

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
```

```
train_clean_subcategories_feature_responseCoding = np.array(get_gv_feature(alpha, "clean_s
# test gene feature
test_clean_subcategories_feature_responseCoding = np.array(get_gv_feature(alpha, "clean_su
###########
print("after encoding")
print(train_clean_subcategories_feature_responseCoding.shape)
print(test_clean_subcategories_feature_responseCoding.shape)
```

```
after encoding
(33500, 2)
(16500, 2)
```

Encoding numerical features: Price

```
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")
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
=====================================================================================
```

Encoding numerical features : teacher_number_of_previously_postec

```
normalizer = Normalizer()

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1

X_train_teacher_posted_projects_norm = normalizer.transform(X_train['teacher_number_of_pre
X_test_teacher_posted_projects_norm = normalizer.transform(X_test['teacher_number_of_previ
```

```
print("After vectorizations")
print(X_train_teacher_posted_projects_norm.shape, y_train.shape)
print(X_test_teacher_posted_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

Encoding numerical features : sentiment Score of eassay

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

X_train_neg_norm = normalizer.transform(X_train['neg'].values.reshape(-1,1))
X_test_neg_norm = normalizer.transform(X_test['neg'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_neg_norm.shape, y_train.shape)
print(X_test_neg_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

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

X_train_pos_norm = normalizer.transform(X_train['pos'].values.reshape(-1,1))
X_test_pos_norm = normalizer.transform(X_test['pos'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_pos_norm.shape, y_train.shape)
print(X_test_pos_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

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

X_train_neu_norm = normalizer.transform(X_train['neu'].values.reshape(-1,1))
X_test_neu_norm = normalizer.transform(X_test['neu'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_neu_norm.shape, y_train.shape)
print(X_test_neu_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
=====================================================================================
```

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

X_train_compound_norm = normalizer.transform(X_train['compound'].values.reshape(-1,1))
X_test_compound_norm = normalizer.transform(X_test['compound'].values.reshape(-1,1))
```

```
print("After vectorizations")
print(X_train_compound_norm.shape, y_train.shape)
print(X_test_compound_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
=====================================================================================
```

```
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')
```

Concatinating all the features

set1 with tfidf

```
from scipy.sparse import hstack
X_tr_set1= hstack((X_train_eassy_tfidf,train_school_state_feature_responseCoding,train_tea
X_te_set1 = hstack((X_test_eassy_tfidf,test_school_state_feature_responseCoding,test_teach

print("Final Data matrix")
print(X_tr_set1.shape, y_train.shape)
print(X_te_set1.shape, y_test.shape)
```

```
print("="*100)
```

```
Final Data matrix
(33500, 5016) (33500,)
(16500, 5016) (16500,)
====================================================================================
```

data set2 with tfidf_weighted_w2v

```
X_tr_set2 = np.hstack((train_tfidf_w2v_essays,train_school_state_feature_responseCoding,tr
X_te_set2 = np.hstack((test_tfidf_w2v_essays,test_school_state_feature_responseCoding,test_
print("Final Data matrix")
print(X_tr_set2.shape, y_train.shape)
print(X_te_set2.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 312) (33500,)
(16500, 312) (16500,)
====================================================================================
```

1.5 Appling Models on different kind of featurization as mentioned ir

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 instrucations

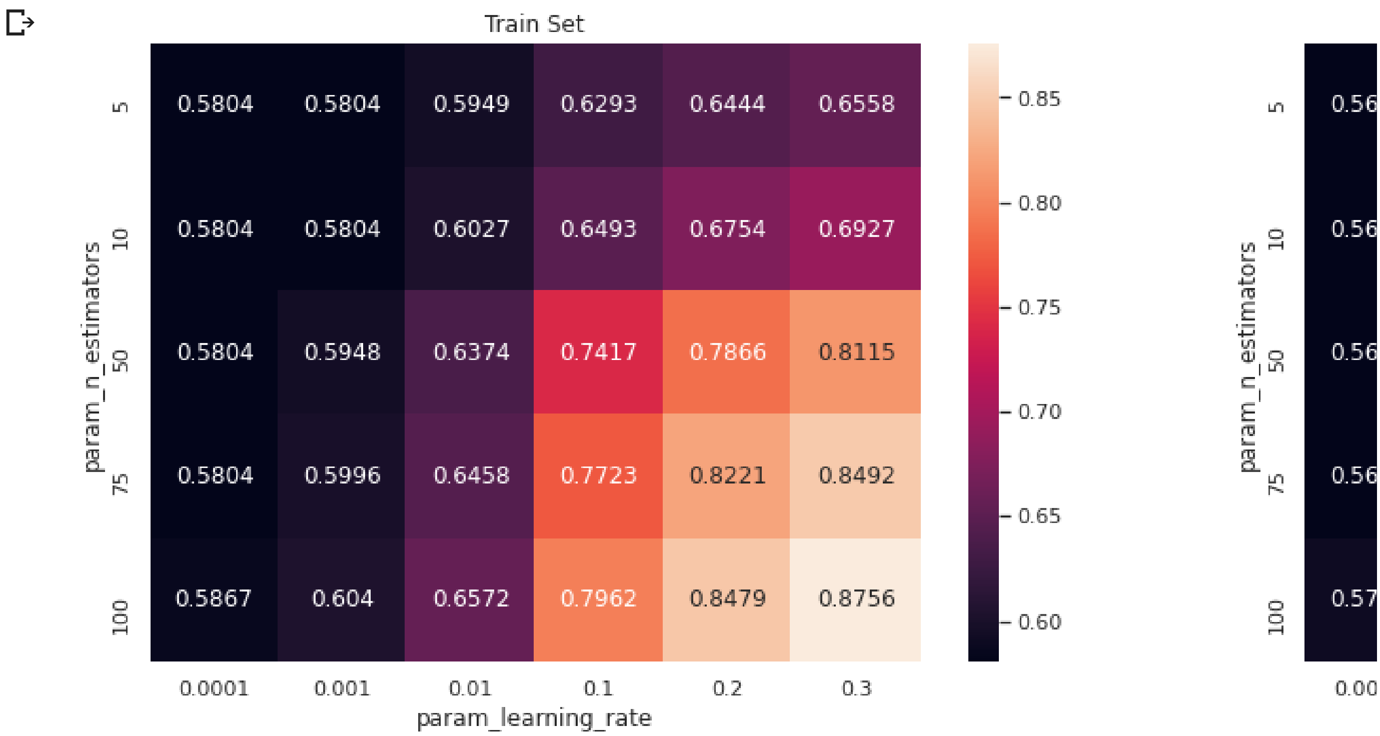
Apply GBDT on set1(tfidf)

```
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]

n_estimators=[5,10,50, 75, 100]
parameters = {'learning_rate':[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3],'n_estimators':[5,10,50
xgb = XGBClassifier(class_weight = 'balanced')
clf1 = GridSearchCV(xgb,parameters, cv=3, scoring='roc_auc',return_train_score=True)
se1 = clf1.fit(X_tr_set1, y_train)
```

```
results = pd.DataFrame.from_dict(clf1.cv_results_)
print(results.columns)
```

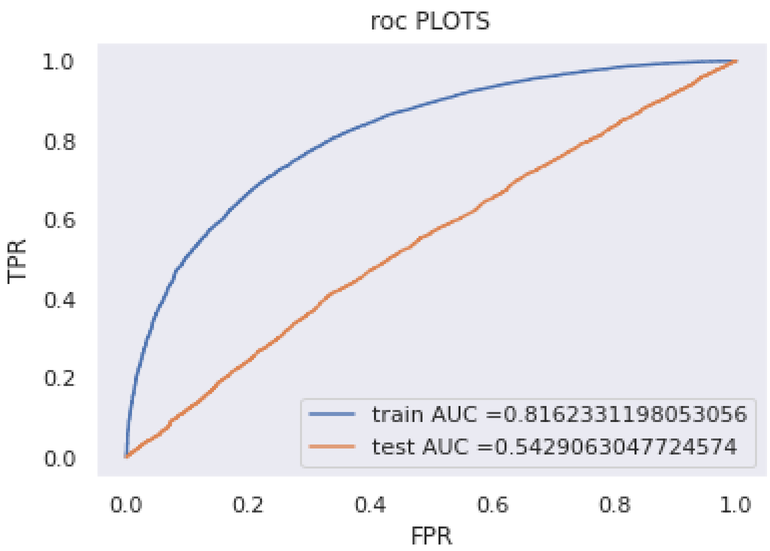
```
Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
       'param_learning_rate', 'param_n_estimators', 'params',
       'split0_test_score', 'split1_test_score', 'split2_test_score',
       'mean_test_score', 'std_test_score', 'rank_test_score',
       'split0_train_score', 'split1_train_score', 'split2_train_score',
       'mean_train_score', 'std_train_score'],
      dtype='object')
```

```
:sults = pd.DataFrame.from_dict(clf1.cv_results_)
'ain_auc= results['mean_train_score']
'ain_auc_std= results['std_train_score']
'_auc = results['mean_test_score']
:v_auc_std= results['std_test_score']
'esults = results.sort_values(['param_max_depth'])
'esults = results.sort_values(['param_min_samples_split'])
ıram_learning_rate=  results['param_learning_rate']
ıram_n_estimators= results['param_n_estimators']
ıport seaborn as sns; sns.set()
ıx_scores1 = pd.DataFrame(results).groupby(['param_n_estimators', 'param_learning_rate']).m
.g, ax = plt.subplots(1,2, figsize=(20,6))
ıs.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
ıs.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
:[0].set_title('Train Set')
:[1].set_title('CV Set')
.t.show()
```




```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn
from sklearn.metrics import roc_curve, auc
best_learing_rate=0.2
best_n_estimator=100

clf11= XGBClassifier(class_weight = 'balanced',learning_rate=.02,n_estimators=100,random_s
clf11.fit(X_tr_set1, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
```

```
# not the predicted outputs

y_train_pred= clf11.predict_proba(X_tr_set1)[:,1]
y_test_pred = clf11.predict_proba(X_te_set1)[:,1]


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("roc PLOTS")
plt.grid()
plt.show()
```



confusion_matrix

```
y_train_pred= clf11.predict(X_tr_set1)
y_test_pred = clf11.predict(X_te_set1)


from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
 C = confusion_matrix(test_y, predict_y)

 A =(((C.T)/(C.sum(axis=1))).T)

 B =(C/C.sum(axis=0))
 plt.figure(figsize=(20,4))

 labels = [0,1]
 # representing A in heatmap format
 cmap=sns.light_palette("blue")
 plt.subplot(1, 3, 1)
```

```
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()


print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

Train confusion_matrix


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

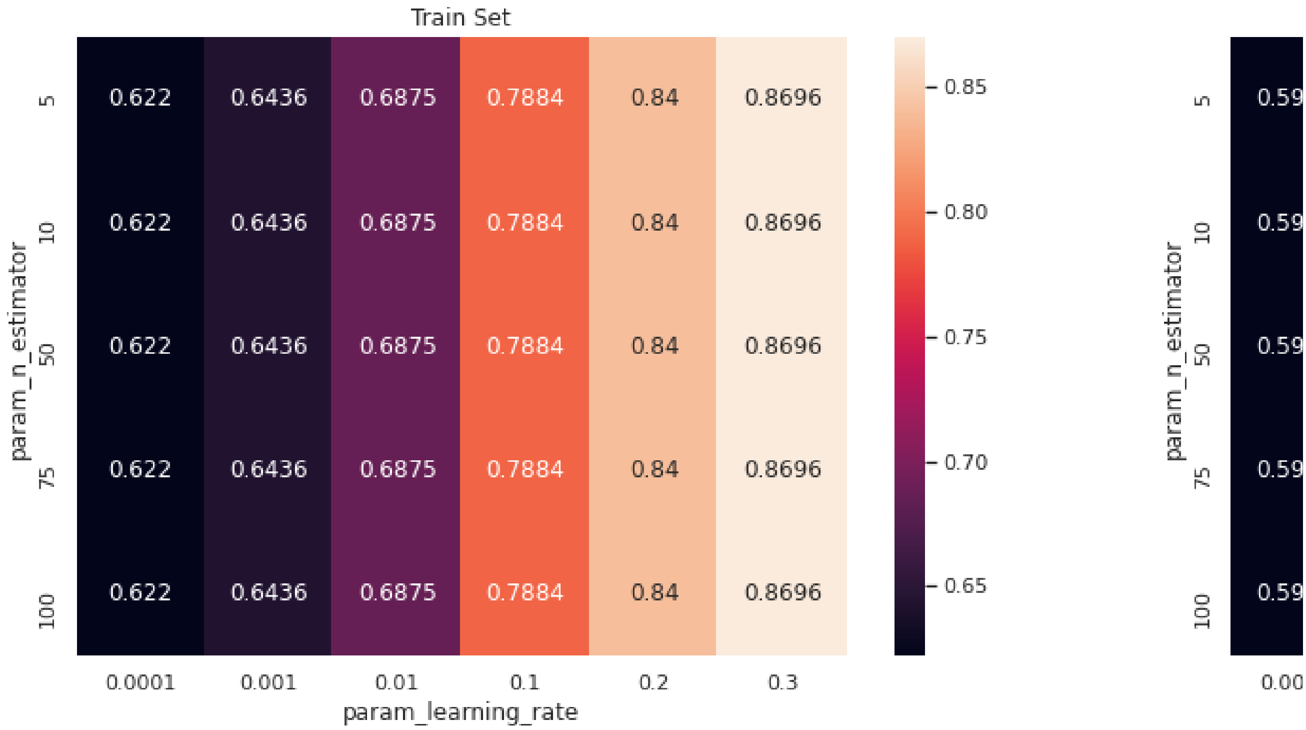
Predicted Class Predicted Class

```
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier
parameters = {'learning_rate':[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3],'n_estimator':[5,10,50,
xgb = XGBClassifier(class_weight = 'balanced')
clf2 = GridSearchCV(xgb,parameters, cv=3, scoring='roc_auc',return_train_score=True)
se2 = clf2.fit(X_tr_set2, y_train)
```

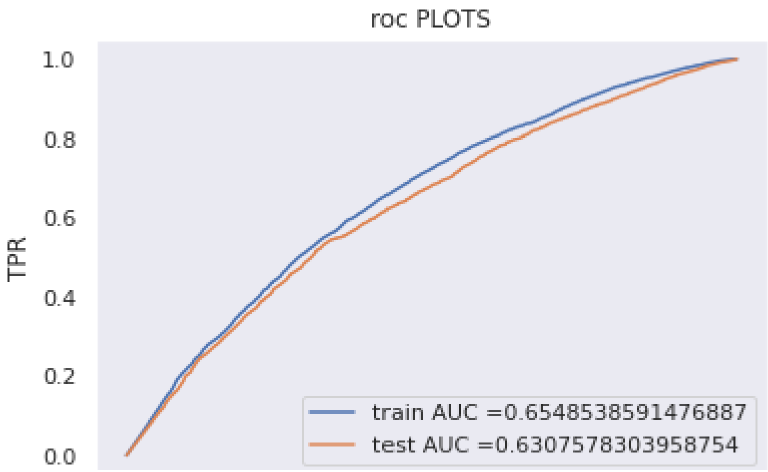
```
results = pd.DataFrame.from_dict(clf2.cv_results_)
print(results.columns)
```

```
Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
       'param_learning_rate', 'param_n_estimator', 'params',
       'split0_test_score', 'split1_test_score', 'split2_test_score',
       'mean_test_score', 'std_test_score', 'rank_test_score',
       'split0_train_score', 'split1_train_score', 'split2_train_score',
       'mean_train_score', 'std_train_score'],
      dtype='object')
```

```
results = pd.DataFrame.from_dict(clf2.cv_results_)
train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
#cv_auc_std= results['std_test_score']
#results = results.sort_values(['param_max_depth'])
#results = results.sort_values(['param_min_samples_split'])
param_learning_rate=  results['param_learning_rate']
param_n_estimator= results['param_n_estimator']
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(results).groupby(['param_n_estimator', 'param_learning_rate']).
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



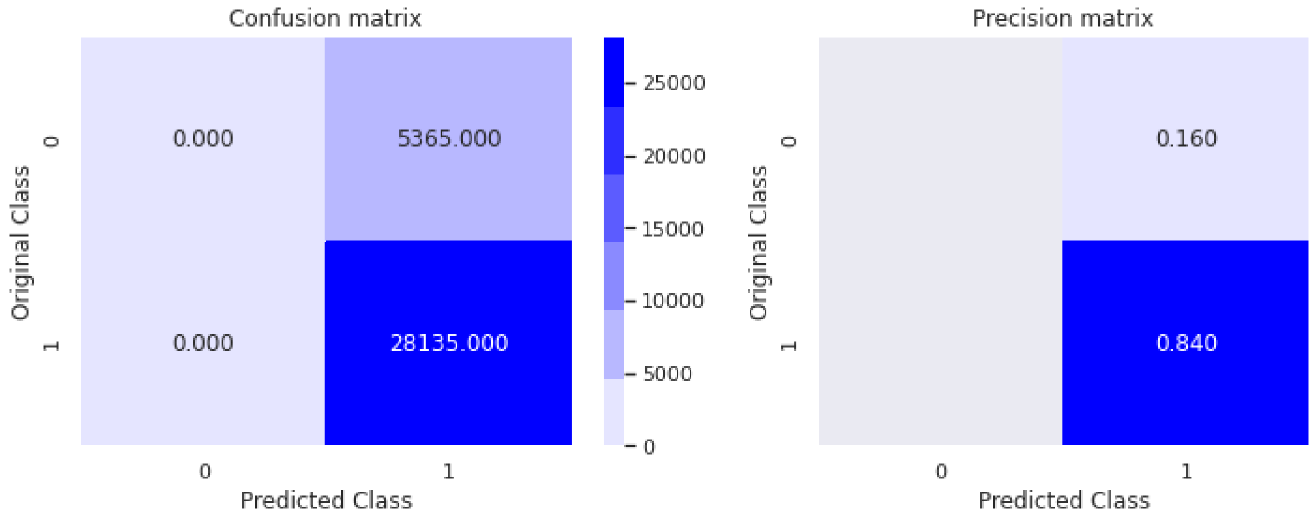

```
m2=clf2.best_params_['max_depth']
n2=clf2.best_params_['n_estimators']



# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn
from sklearn.metrics import roc_curve, auc
best_learing_rate=0.1
best_n_estimator=5

clf21= XGBClassifier(class_weight = 'balanced',learning_rate=0.1,n_estimators=5,random_sta
clf21.fit(X_tr_set2, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs

y_train_pred= clf21.predict_proba(X_tr_set2)[:,1]
y_test_pred = clf21.predict_proba(X_te_set2)[:,1]


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("roc PLOTS")
plt.grid()
plt.show()
```



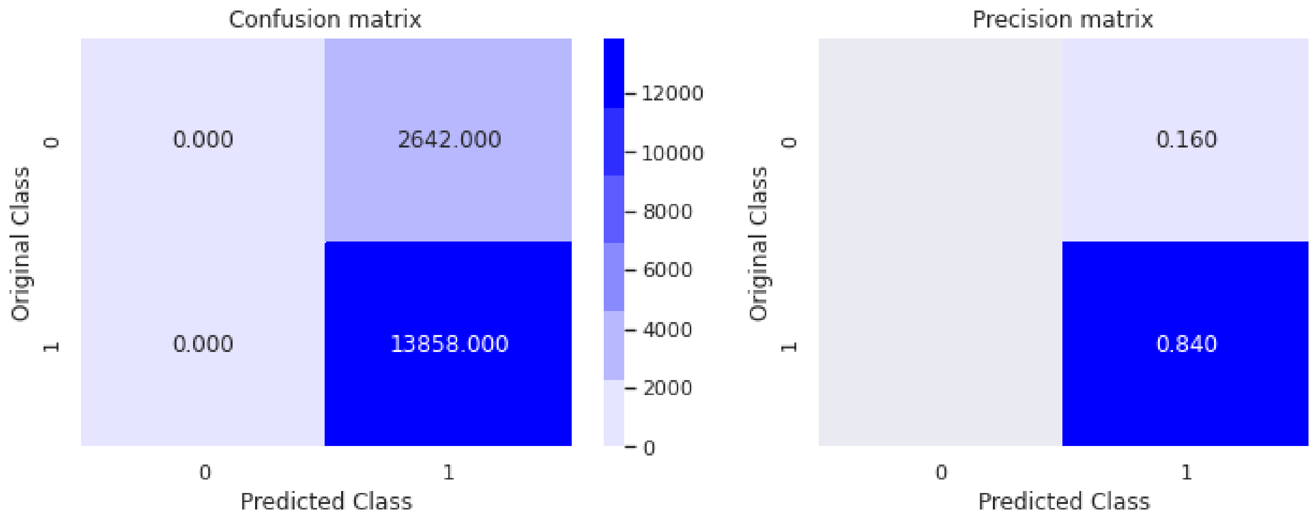

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1= predictions
    return predictions


y_train_pred= clf21.predict(X_tr_set2)
y_test_pred = clf21.predict(X_te_set2)


from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(test_y, predict_y):
 C = confusion_matrix(test_y, predict_y)

 A =(((C.T)/(C.sum(axis=1))).T)

 B =(C/C.sum(axis=0))
 plt.figure(figsize=(20,4))

 labels = [0,1]
 # representing A in heatmap format
 cmap=sns.light_palette("blue")
 plt.subplot(1, 3, 1)
 sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
 plt.xlabel('Predicted Class')
 plt.ylabel('Original Class')
 plt.title("Confusion matrix")

 plt.subplot(1, 3, 2)
```

```
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()


print('Train confusion_matrix')
plot_confusion_matrix(y_train,y_train_pred)
print('Test confusion_matrix')
plot_confusion_matrix(y_test,y_test_pred)
```

```
Train confusion_matrix
```




```
Test confusion_matrix
```



3. Summary

as mentioned in the step 4 of instructions

```
import pandas as pd
from tabulate import tabulate
weather_data = [('TFIDF','GBDT',"'learning_rate': 4, 'n_estimators': 100", 0.54),
                ('TFIDF_weighted_w2v', 'GBDT', "'learning_rate': 4, 'n_estimators':5", 0.6
               ]
df = pd.DataFrame(weather_data, columns=['Vectorizer', 'Model', 'Hyper parameter', 'AUC'])


#Ref: https://pypi.org/project/tabulate/
print(tabulate(df, headers='keys', tablefmt='github'))
```

```
|    | Vectorizer         | Model    | Hyper parameter                          |   AUC
|----|--------------------|----------|------------------------------------------|------
|  0 | TFIDF              | GBDT     | 'learning_rate': 4, 'n_estimators': 100  |  0.54
|  1 | TFIDF_weighted_w2v | GBDT     | 'learning_rate': 4, 'n_estimators':5     |  0.63
```