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

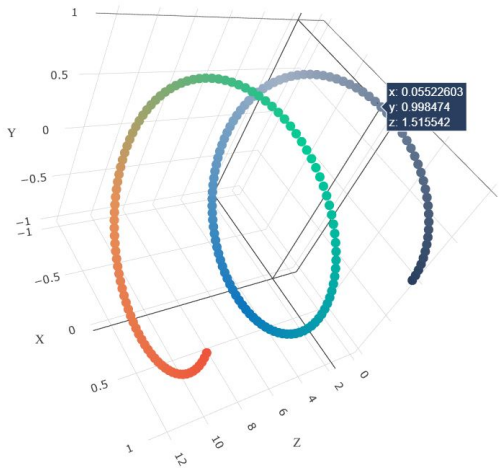
State
A
C
D
C
B
E

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]

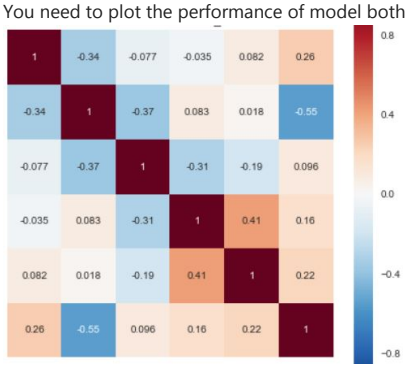
1. **Apply GBDT on these feature sets**
   - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check the bellow example, include all 4 values as 4 features)
   - 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*

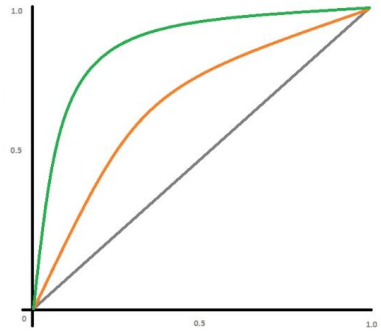
or

- 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 [ ]:

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

```
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
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
/usr/local/lib/python3.6/dist-packages/nltk/twitter/__init__.py:20: UserWarning: The twython library has not been installed. Some functio
nality from the twitter package will not be available.
  warnings.warn("The twython library has not been installed. "
```

1. GBDT (xgboost/lightgbm)

1.1 Loading Data

In [4]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

!pip install chart_studio
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 joblib

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

import pickle
from tqdm import tqdm
import os
from sklearn import preprocessing

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

import nltk
nltk.download('vader_lexicon')

import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.metrics import roc_curve, auc

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

sid = SentimentIntensityAnalyzer()
```

```
Collecting chart_studio
  Downloading https://files.pythonhosted.org/packages/ca/ce/330794a6b6ca4b9182c38fc69dd2a9cbff60fd49421cb8648ee5fee352dc/chart_studio-1.
1.0-py3-none-any.whl (64kB)
    |████████████████████████████████| 71kB 5.2MB/s
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from chart_studio) (1.15.0)
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from chart_studio) (4.4.1)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packages (from chart_studio) (1.3.3)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from chart_studio) (2.23.0)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests->chart_st
udio) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests->chart_studio) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests->chart_studio) (2020.12.5)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests->chart_studio) (3.0.4)
Installing collected packages: chart-studio
Successfully installed chart-studio-1.1.0

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

In [ ]:
```
data  = pd.read_csv('preprocessed_data2.csv',nrows = 70000)
data.head(5)
```

Out[ ]:

Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_subca

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_subca
0	0	ca	mrs	grades_prek_2	53	1	math_science	applied health_lif
1	1	ut	ms	grades_3_5	4	1	specialneeds	spec
2	2	ca	mrs	grades_prek_2	10	1	literacy_language	
3	3	ga	mrs	grades_prek_2	2	1	appliedlearning	earlydeve
4	4	wa	mrs	grades_3_5	2	1	literacy_language	

In [ ]:
```
print("Number of data points in train data", data.shape)
print('-'*50)
print("The attributes of data :", data.columns.values)
```

```
Number of data points in train data (70000, 11)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'school_state' 'teacher_prefix' 'project_grade_category'
 'teacher_number_of_previously_posted_projects' 'project_is_approved'
 'clean_categories' 'clean_subcategories' 'essay' 'price' 'project_title']
```

In [ ]:
```
# check if we have any nan values are there
print(data['project_title'].isnull().values.any())
print("number of nan values",data['project_title'].isnull().values.sum())
```

```
True
number of nan values 33
```

In [ ]:
```
#Droping nan datapoints from DataFrame
data = data.dropna()
```

In [ ]:
```
# check if we have any nan values are there
print(data['project_title'].isnull().values.any())
print("number of nan values",data['project_title'].isnull().values.sum())
```

```
False
number of nan values 0
```

In [ ]:
```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(3)
```

Out[ ]:

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	essay
0	0	ca	mrs	grades_prek_2	53	math_science	appliedsciences health_lifescience	i fortunate enough use fairy tale stem kits cl...
1	1	ut	ms	grades_3_5	4	specialneeds	specialneeds	imagine 8 9 years old you third grade classroo...

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	essay
2	2	ca	mrs	grades_prek_2	10	literacy_language	literacy	having class 24 students comes diverse learner...

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

In [ ]:
```
# train test split
from sklearn.model_selection import train_test_split
index = np.arange(len(X))
X_train, X_test, y_train, y_test,ix ,iy,data_train, data_test  = train_test_split(X, y,index,data, test_size=0.3, stratify=y)
```

In [ ]:
```
(X_train.shape),(X_test.shape)
```

Out[ ]:
```
((48976, 10), (20991, 10))
```

1.3 Make Data Model Ready: encoding eassay, and project_title

In [ ]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print("="*100)

vectorizer_1 = TfidfVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer_1.fit(X_train['essay'].values) # fit has to happen only on train data


X_train_essay_tfidf = vectorizer_1.transform(X_train['essay'].values)
X_test_essay_tfidf = vectorizer_1.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
(48976, 10) (48976,)
(20991, 10) (20991,)
====================================================================================================
After vectorizations
(48976, 5000) (48976,)
(20991, 5000) (20991,)
====================================================================================================
```

In [ ]:
```
np.save('X_train2', X_train1)
np.save('X_test2', X_test1)
```

In [ ]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print("="*100)

vectorizer_2 = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=500)
vectorizer_2.fit(X_train['project_title'].values) # fit has to happen only on train data


X_train_project_title_tfidf = vectorizer_2.transform(X_train['project_title'].values)
X_test_project_title_tfidf = vectorizer_2.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
(48976, 10) (48976,)
(20991, 10) (20991,)
====================================================================================================
After vectorizations
(48976, 5000) (48976,)
(20991, 5000) (20991,)
====================================================================================================
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [ ]:
```
def group_by(data, col1, col2='project_is_approved'):
  temp = pd.DataFrame(data.groupby(col1)[col2].agg(lambda x: x.eq(1).sum())).reset_index()
  temp1 = pd.DataFrame(data.groupby(col1)[col2].agg(lambda x: x.count())).reset_index()
  temp['total'] = temp1[col2]
  temp['neg'] = (temp['total']-temp[col2])
  temp['positive'] = (temp[col2]/temp['total'])
  temp['negative'] = (temp['neg']/temp['total'])
```

```
  return temp
```

In [ ]:
```
def response_coding(train,col):
  pos , neg = list(), list()
  df = pd.DataFrame()
  for i, row in train.iterrows():
    if row[col] in temp[col].values:
      for i, rows in temp.iterrows():
        if rows[col] == row[col]:
          pos.append(rows['positive'])
          neg.append(rows['negative'])
    else:
      pos.append(0.5)
      neg.append(0.5)

  data1 = {'pos':pos, 'neg':neg }
  df = pd.DataFrame(data1)

  return df.to_numpy()
```

1.4.1 encoding categorical features: School State

In [ ]:
```
temp = group_by(data_train, 'school_state', col2='project_is_approved')
df_school_state_train = response_coding(X_train,'school_state')
df_school_state_test = response_coding(X_test,'school_state')
```

1.4.2 encoding categorical features: teacher_prefix

In [ ]:
```
temp = group_by(data_train, 'teacher_prefix', col2='project_is_approved')
df_teacher_prefix_train = response_coding(X_train,'teacher_prefix')
df_teacher_prefix_test = response_coding(X_test,'teacher_prefix')
```

1.4.3 encoding categorical features: project_grade_category

In [ ]:
```
temp = group_by(data_train, 'project_grade_category', col2='project_is_approved')
df_project_grade_category_train = response_coding(X_train,'project_grade_category')
df_project_grade_category_test = response_coding(X_test,'project_grade_category')
```

1.4.4 encoding categorical features: project_subject_categories

In [ ]:
```
temp = group_by(data_train, 'clean_categories', col2='project_is_approved')
df_project_subject_categories_train = response_coding(X_train,'clean_categories')
df_project_subject_categories_test = response_coding(X_test,'clean_categories')
```

1.4.5 encoding categorical features: project_subject_subcategories

In [ ]:
```
temp = group_by(data_train, 'clean_subcategories', col2='project_is_approved')
df_project_subject_subcategories_train = response_coding(X_train,'clean_subcategories')
df_project_subject_subcategories_test = response_coding(X_test,'clean_subcategories')
```

1.4.6 encoding numerical features: Price

In [ ]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
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
(1, 48976) (48976,)
(1, 20991) (20991,)
====================================================================================================
```

In [ ]:
```
#we are defining this function to return arrray
def to_array(a):
  b = a.tolist()
  c = []
  for i in b:
    for  j in i:
      c.append(j)
  d = [i for i in range(len(c))]
```

```
  df = pd.DataFrame(list(zip(d, c)),columns = ['1','2'])
  e = df.drop(['1'], axis=1)
  array = e.to_numpy()

  return array
```

In [ ]:
```
X_train_price_norm_array = to_array(X_train_price_norm)
X_test_price_norm_array = to_array(X_test_price_norm)
```

1.4.6 sentiment analysis for essay

In [ ]:
```
def to_array1(a):
  d = [i for i in range(len(a))]
  df = pd.DataFrame(list(zip(d, a)),columns = ['1','2'])
  e = df.drop(['1'], axis=1)
  array = e.to_numpy()

  return array

def sentiment(X):
  neg, neu, pos, compound = list(), list(), list(), list()
  for i, row in X.iterrows():
    ss = sid.polarity_scores(row['essay'])
    neg.append(ss['neg'])
    neu.append(ss['neu'])
    pos.append(ss['pos'])
    compound.append(ss['compound'])

  neg_array = to_array1(neg)
  neu_array = to_array1(neu)
  pos_array = to_array1(pos)
  compound_array = to_array1(compound)

  return  neg_array,neu_array,pos_array,compound_array

X_train1_neg,X_train1_neu,X_train1_pos,X_train1_compound = sentiment(X_train)
X_test1_neg,X_test1_neu,X_test1_pos,X_test1_compound = sentiment(X_test)
```

1.4.7 Concatinating all the features

In [ ]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfidf,X_train_project_title_tfidf)).tocsr()
X_te = hstack((X_test_essay_tfidf,X_test_project_title_tfidf)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)

print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(48976, 5500) (48976,)
(20991, 5500) (20991,)
====================================================================================================
```

In [ ]:
```
X_tr1 = X_tr.toarray()
#Normalize Data
X_tr1 = preprocessing.normalize(X_tr1)
df_school_state_train =preprocessing.normalize(df_school_state_train)
df_teacher_prefix_train =preprocessing.normalize(df_teacher_prefix_train)
df_project_grade_category_train =preprocessing.normalize(df_project_grade_category_train)
df_project_subject_categories_train =preprocessing.normalize(df_project_subject_categories_train)
df_project_subject_subcategories_train =preprocessing.normalize(df_project_subject_subcategories_train)
X_train_neg = preprocessing.normalize(X_train1_neg)
X_train_neu = preprocessing.normalize(X_train1_neu)
X_train_pos = preprocessing.normalize(X_train1_pos)
X_train_compound = preprocessing.normalize(X_train1_compound)
X_tr2 = np.concatenate((X_tr1,df_school_state_train,df_teacher_prefix_train,df_project_grade_category_train,df_project_subject_categories_tr
```

In [ ]:
```
X_te1 = X_te.toarray()
#Normalize Data
X_te1 = preprocessing.normalize(X_te1)
df_school_state_test =preprocessing.normalize(df_school_state_test)
df_teacher_prefix_test =preprocessing.normalize(df_teacher_prefix_test)
df_project_grade_category_test =preprocessing.normalize(df_project_grade_category_test)
df_project_subject_categories_test =preprocessing.normalize(df_project_subject_categories_test)
df_project_subject_subcategories_test =preprocessing.normalize(df_project_subject_subcategories_test)
X_test_neg = preprocessing.normalize(X_test1_neg)
X_test_neu = preprocessing.normalize(X_test1_neu)
X_test_pos = preprocessing.normalize(X_test1_pos)
X_test_compound = preprocessing.normalize(X_test1_compound)
X_te2 = np.concatenate((X_te1,df_school_state_test,df_teacher_prefix_test,df_project_grade_category_test,df_project_subject_categories_test,
```

In [ ]:
```
np.save('X_train10', X_tr2)
np.save('X_test10', X_te2)
np.save('y_train10', y_train)
np.save('y_test10', y_test)
```

In [5]:
```
X_train1 = np.load('X_train10.npy')
X_test1 = np.load('X_test10.npy')
y_train1 = np.load('y_train10.npy')
y_test1 = np.load('y_test10.npy')
```

1.5 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 instrucations

In [5]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [6]:
```
#import gensim
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
from sklearn.decomposition import TruncatedSVD
from sklearn import tree
import graphviz
from sklearn.model_selection import GridSearchCV

import xgboost as xgb
```

In [8]:
```
clf_gb = xgb.XGBClassifier(n_jobs=-1)
param_grid = {
                'learning_rate' : [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3],
                'n_estimators' : [5,10,50, 75, 100, 200]}

# Instantiate the grid search model
GridSearch_xgb = GridSearchCV(clf_gb,param_grid,cv=3,scoring='roc_auc', return_train_score=True)
GridSearch_xgb.fit(X_train1, y_train1)
```

Out[8]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=-1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3],
                         'n_estimators': [5, 10, 50, 75, 100, 200]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=0)
```

In [10]:
```
f = open('GridSearch_xgb10.pkl', 'wb')   # 'wb' instead 'w' for binary file
pickle.dump(GridSearch_xgb, f, -1)       # -1 specifies highest binary protocol
f.close()
```

In [10]:
```
f = open('GridSearch_xgb10.pkl', 'rb')   # 'rb' for reading binary file
model = pickle.load(f)
f.close()
```

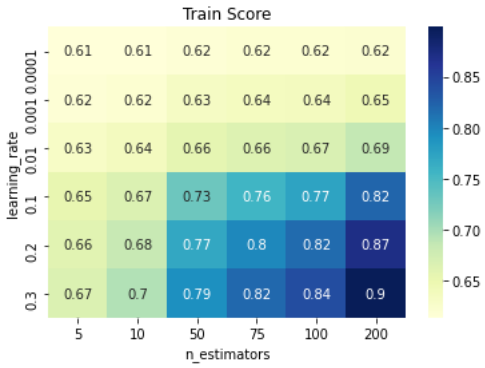
In [12]:
```
print(model.best_estimator_)
print("Best HyperParameter: ",model.best_params_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.2, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=nan, n_estimators=200, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
Best HyperParameter:  {'learning_rate': 0.2, 'n_estimators': 200}
```

In [13]:
```
Train_mean = model.cv_results_['mean_train_score']

X = [0.0001,0.0001,0.0001,0.0001,0.0001,0.0001,0.001,0.001,0.001,0.001,0.001,0.001,0.01
     ,0.01,0.01,0.01,0.01,0.01,0.1,0.1,0.1,0.1,0.1,0.1,0.2,0.2,0.2,0.2,0.2,0.2,0.3,0.3,0.3,0.3,0.3,0.3]
Y = [5,10,50, 75, 100, 200,5,10,50, 75, 100, 200,5,10,50, 75, 100, 200,5,10,50, 75, 100, 200,
     5,10,50, 75, 100, 200]
```

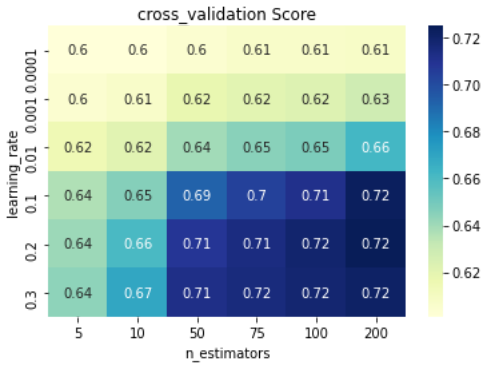
```
df = pd.DataFrame(list(zip(X,Y,Train_mean)),
                  columns =['learning_rate', 'n_estimators', 'Train_mean'])
plot_data = df.pivot("learning_rate", "n_estimators", "Train_mean")
ax = sns.heatmap(plot_data, annot=True, cmap="YlGnBu")
ax.set_title('Train Score')
```

Out[13]: Text(0.5, 1.0, 'Train Score')

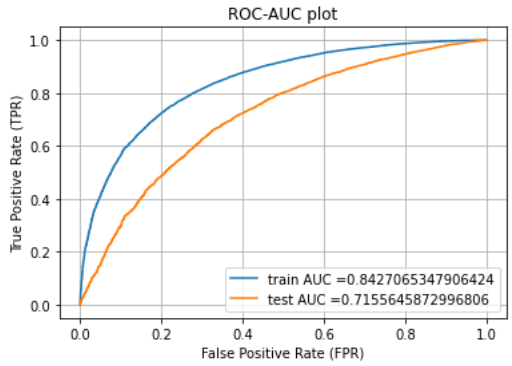


In [14]:
```
CV_mean = model.cv_results_['mean_test_score']

df = pd.DataFrame(list(zip(X,Y,CV_mean)),
                  columns =['learning_rate', 'n_estimators', 'CV_mean'])
plot_data = df.pivot("learning_rate", "n_estimators", "CV_mean")
ax = sns.heatmap(plot_data, annot=True, cmap="YlGnBu")
ax.set_title('cross_validation Score')
```

Out[14]: Text(0.5, 1.0, 'cross_validation Score')



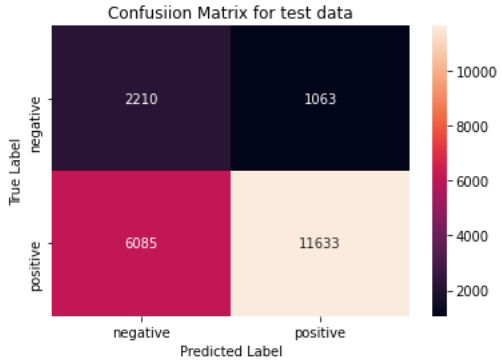
In [15]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve



clf_gb = xgb.XGBClassifier()

clf_gb = xgb.XGBClassifier(learning_rate =  0.2, n_estimators = 200)

clf_gb.fit(X_train1, y_train1)

y_train_pred = batch_predict(clf_gb, X_train1)
y_test_pred = batch_predict(clf_gb, X_test1)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train1, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test1, 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("False Positive Rate (FPR)")
plt.ylabel("True Positive Rate (TPR)")
plt.title("ROC-AUC plot")
plt.grid()
plt.show()
```



In [14]:
```
# 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(t,3))
    return t

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

def Confusion_matrix(y_test, test_pred):
  # Confusion matrix for test data
  plt.figure()
  cm = confusion_matrix(y_test, test_pred)
  class_label = ["negative", "positive"]
  df_cm_test = pd.DataFrame(cm, index = class_label, columns = class_label)
  sns.heatmap(df_cm_test , annot = True, fmt = "d")
  plt.title("Confusiion Matrix for test data")
  plt.xlabel("Predicted Label")
  plt.ylabel("True Label")
  plt.show()
```

In [17]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Test confusion matrix")
Confusion_matrix(y_test1, predict_with_best_t(y_test_pred, best_t))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.5830681532212019 for threshold 0.828
Test confusion matrix
```



Applying TFIDF weighted W2V on essay and project_title sor set2 model

Make Data Model Ready: encoding eassay, and project_title with TFIDF weighted W2V

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

In [ ]:
```
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())
```

In [ ]:
```
# average Word2Vec
# compute average word2vec for each review.
X_train_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay']): # 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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_essay_tfidf_w2v_vectors.append(vector)

X_train_essay = np.array(X_train_essay_tfidf_w2v_vectors)

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

```
100%|██████████| 48976/48976 [01:44<00:00, 467.52it/s]
48976
300
```

In [ ]:
```
# average Word2Vec
# compute average word2vec for each review.
X_test_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay']): # 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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_essay_tfidf_w2v_vectors.append(vector)

X_test_essay = np.array(X_test_essay_tfidf_w2v_vectors)

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

```
100%|██████████| 20991/20991 [00:45<00:00, 462.40it/s]
20991
300
```

In [ ]:
```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['project_title'].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())
```

In [ ]:
```
# average Word2Vec
# compute average word2vec for each review.
X_train_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title']): # 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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_title_tfidf_w2v_vectors.append(vector)

X_train_title = np.array(X_train_title_tfidf_w2v_vectors)

print(len(X_train_title_tfidf_w2v_vectors))
print(len(X_train_title_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 48976/48976 [00:02<00:00, 22179.82it/s]
48976
300
```

In [ ]:
```
# average Word2Vec
# compute average word2vec for each review.
X_test_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
```

```
for sentence in tqdm(X_test['project_title']): # 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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_title_tfidf_w2v_vectors.append(vector)

X_test_title = np.array(X_test_title_tfidf_w2v_vectors)

print(len(X_test_title_tfidf_w2v_vectors))
print(len(X_test_title_tfidf_w2v_vectors[0]))
```

```
100%|██████████| 20991/20991 [00:00<00:00, 23376.04it/s]
20991
300
```

Concatinating all the features

In [ ]:
```
X_tr = np.concatenate((X_train_essay,X_train_title), axis=1)
X_te = np.concatenate((X_test_essay,X_test_title), axis=1)

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(48976, 600) (48976,)
(20991, 600) (20991,)
====================================================================================================
```

In [ ]:
```
#Normalize Data
X_tr1 = preprocessing.normalize(X_tr)
df_school_state_train =preprocessing.normalize(df_school_state_train)
df_teacher_prefix_train =preprocessing.normalize(df_teacher_prefix_train)
df_project_grade_category_train =preprocessing.normalize(df_project_grade_category_train)
df_project_subject_categories_train =preprocessing.normalize(df_project_subject_categories_train)
df_project_subject_subcategories_train =preprocessing.normalize(df_project_subject_subcategories_train)
X_tr2 = np.concatenate((X_tr1,df_school_state_train,df_teacher_prefix_train,df_project_grade_category_train,df_project_subject_categories_tr
```

In [ ]:
```
#Normalize Data
X_te1 = preprocessing.normalize(X_te)
df_school_state_test =preprocessing.normalize(df_school_state_test)
df_teacher_prefix_test =preprocessing.normalize(df_teacher_prefix_test)
df_project_grade_category_test =preprocessing.normalize(df_project_grade_category_test)
df_project_subject_categories_test =preprocessing.normalize(df_project_subject_categories_test)
df_project_subject_subcategories_test =preprocessing.normalize(df_project_subject_subcategories_test)
X_te2 = np.concatenate((X_te1,df_school_state_test,df_teacher_prefix_test,df_project_grade_category_test,df_project_subject_categories_test,
```

In [ ]:
```
np.save('X_train11', X_tr2)
np.save('X_test11', X_te2)
```

In [11]:
```
X_train1 = np.load('X_train11.npy')
X_test1 = np.load('X_test11.npy')
y_train1 = np.load('y_train10.npy')
y_test1 = np.load('y_test10.npy')
```

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

In [9]:
```
clf_gb = xgb.XGBClassifier(n_jobs=-1)
param_grid = {
                'learning_rate' : [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3],
                'n_estimators' : [5,10,50, 75, 100, 200]}

# Instantiate the grid search model
GridSearch_xgb = GridSearchCV(clf_gb,param_grid,cv=3,scoring='roc_auc', return_train_score=True)
GridSearch_xgb.fit(X_train1, y_train1)
```

Out[9]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=-1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3],
                         'n_estimators': [5, 10, 50, 75, 100, 200]},
```

```
                pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                scoring='roc_auc', verbose=0)
```

In [10]:
```
f = open('GridSearch_xgb15.pkl', 'wb')   # 'wb' instead 'w' for binary file
pickle.dump(GridSearch_xgb, f, -1)       # -1 specifies highest binary protocol
f.close()
```

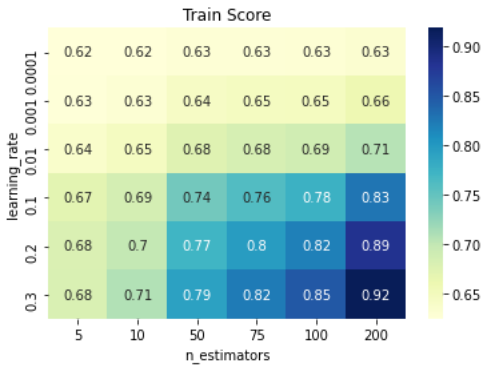
In [7]:
```
f = open('GridSearch_xgb15.pkl', 'rb')   # 'rb' for reading binary file
model = pickle.load(f)
f.close()
```

In [8]:
```
print(model.best_estimator_)
print("Best HyperParameter: ",model.best_params_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=nan, n_estimators=200, n_jobs=-1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
Best HyperParameter:  {'learning_rate': 0.1, 'n_estimators': 200}
```

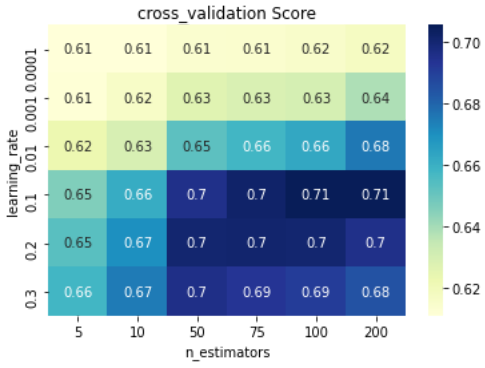
In [9]:
```
Train_mean = model.cv_results_['mean_train_score']

X = [0.0001,0.0001,0.0001,0.0001,0.0001,0.0001,0.001,0.001,0.001,0.001,0.001,0.001,0.01
     ,0.01,0.01,0.01,0.01,0.01,0.1,0.1,0.1,0.1,0.1,0.1,0.2,0.2,0.2,0.2,0.2,0.2,0.3,0.3,0.3,0.3,0.3,0.3]
Y = [5,10,50, 75, 100, 200,5,10,50, 75, 100, 200,5,10,50, 75, 100, 200,5,10,50, 75, 100, 200,
     5,10,50, 75, 100, 200]


df = pd.DataFrame(list(zip(X,Y,Train_mean)),
                  columns =['learning_rate', 'n_estimators', 'Train_mean'])
plot_data = df.pivot("learning_rate", "n_estimators", "Train_mean")
ax = sns.heatmap(plot_data, annot=True, cmap="YlGnBu")
ax.set_title('Train Score')
```

Out[9]:
```
Text(0.5, 1.0, 'Train Score')
```



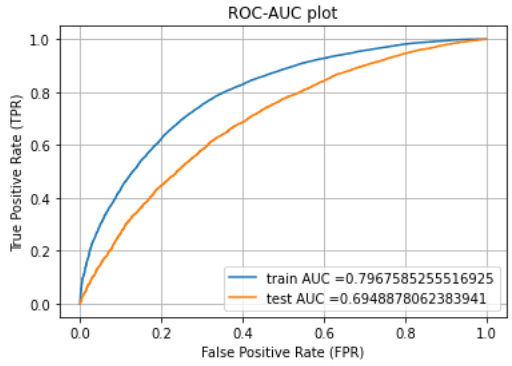
In [10]:
```
CV_mean = model.cv_results_['mean_test_score']

df = pd.DataFrame(list(zip(X,Y,CV_mean)),
                  columns =['learning_rate', 'n_estimators', 'CV_mean'])
plot_data = df.pivot("learning_rate", "n_estimators", "CV_mean")
ax = sns.heatmap(plot_data, annot=True, cmap="YlGnBu")
ax.set_title('cross_validation Score')
```

Out[10]:
```
Text(0.5, 1.0, 'cross_validation Score')
```



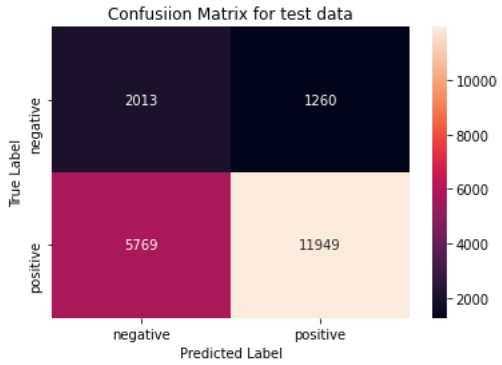
In [12]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve


clf_gb = xgb.XGBClassifier()
```

```
clf_gb = xgb.XGBClassifier(learning_rate =  0.1, n_estimators = 200)

clf_gb.fit(X_train1, y_train1)

y_train_pred = batch_predict(clf_gb, X_train1)
y_test_pred = batch_predict(clf_gb, X_test1)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train1, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test1, 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("False Positive Rate (FPR)")
plt.ylabel("True Positive Rate (TPR)")
plt.title("ROC-AUC plot")
plt.grid()
plt.show()
```



In [15]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Test confusion matrix")
Confusion_matrix(y_test1, predict_with_best_t(y_test_pred, best_t))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.5263783727070838 for threshold 0.82
Test confusion matrix
```



3. Summary

In [17]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper parameter", "AUC"]
x.add_row(["TFIDF", "GBDT", "learning_rate = 0.2, n_estimators = 200", 0.715])
x.add_row(["TFIDF weighted W2V", "GBDT", "learning_rate = 0.1, n_estimators = 200", 0.694])

print(x)
```

```
+--------------------+-------+-----------------------------------------+-------+
|     Vectorizer     | Model |             Hyper parameter             |  AUC  |
+--------------------+-------+-----------------------------------------+-------+
|       TFIDF        |  GBDT | learning_rate = 0.2, n_estimators = 200 | 0.715 |
| TFIDF weighted W2V |  GBDT | learning_rate = 0.1, n_estimators = 200 | 0.694 |
+--------------------+-------+-----------------------------------------+-------+
```