Assignment 9: GBDT

Response Coding: Example

> 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]

Few Notes

1. Use atleast 35k data points
2. Use classifier.Predict_proba() method instead of predict() method while calculating roc_auc scores
3. Be sure that you are using laplase smoothing in response encoding function. Laplase smoothing means applying the default (0.5) value to test data if the test data is not present in the train set

```
%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
from sklearn.model_selection import GridSearchCV

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

import pickle
from tqdm import tqdm
import os

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

```
import warnings
warnings.simplefilter("ignore", UserWarning)

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

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

sid = SentimentIntensityAnalyzer()

sample_sentence_1='I am happy.'
ss_1 = sid.polarity_scores(sample_sentence_1)
print('sentiment score for sentence 1',ss_1)

sample_sentence_2='I am sad.'
ss_2 = sid.polarity_scores(sample_sentence_2)
print('sentiment score for sentence 2',ss_2)

sample_sentence_3='I am going to New Delhi tommorow.'
ss_3 = sid.polarity_scores(sample_sentence_3)
print('sentiment score for sentence 3',ss_3)
```

```
sentiment score for sentence 1 {'neg': 0.0, 'neu': 0.213, 'pos':
0.787, 'compound': 0.5719}
sentiment score for sentence 2 {'neg': 0.756, 'neu': 0.244, 'pos':
0.0, 'compound': -0.4767}
sentiment score for sentence 3 {'neg': 0.0, 'neu': 1.0, 'pos': 0.0,
'compound': 0.0}
```

1.1 Loading Data

```
import pandas
data = pandas.read_csv('preprocessed_data.csv', nrows=15000) #Please
change the no of rows in dataset if required

data.head(2)
```

```
  school_state teacher_prefix project_grade_category  \
0           ca            mrs          grades_prek_2
1           ut             ms             grades_3_5

   teacher_number_of_previously_posted_projects
project_is_approved  \
0                                            53                    1

1                                             4                    1
```

```
  clean_categories                 clean_subcategories  \
0     math_science  appliedsciences health_lifescience
1     specialneeds                         specialneeds

                                               essay   price
0  i fortunate enough use fairy tale stem kits cl...  725.05
1  imagine 8 9 years old you third grade classroo...  213.03
```

```
# calculate sentiment scores for the essay feature
sentiment_list =[]
for sentence in data["essay"]:
    sentiment_list.append(sid.polarity_scores(sentence)["compound"])

#print(sentiment_list)
data["essay_sentimental_score"] = sentiment_list
data.head(2)
```

```
  school_state teacher_prefix project_grade_category  \
0           ca            mrs          grades_prek_2
1           ut             ms             grades_3_5

   teacher_number_of_previously_posted_projects
project_is_approved  \
0                                            53                    1

1                                             4                    1


  clean_categories                 clean_subcategories  \
0     math_science  appliedsciences health_lifescience
1     specialneeds                         specialneeds

                                               essay   price  \
0  i fortunate enough use fairy tale stem kits cl...  725.05
1  imagine 8 9 years old you third grade classroo...  213.03

   essay_sentimental_score
0                   0.9867
1                   0.9897
```

```
# please write all the code with proper documentation, and proper
titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull
in debugging your code
# when you plot any graph make sure you use
```

```
    # a. Title, that describes your plot, this will be very helpful to
the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

# 2. Split your data.
from sklearn.model_selection import train_test_split

Y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
data_set1 = X
data_set2 = X.copy()

#Data Set1:

# Split data set into train and test
X_train1, X_test1, y_train1, y_test1 = train_test_split( data_set1, Y,
test_size=0.2,stratify=Y, random_state=12)
print("X_train1 (60%):", X_train1.shape)
print("X_test1 (20%):",X_test1.shape)

#Data Set2:
Y = data.project_is_approved
# Split data set into train and test
X_train2, X_test2, y_train2, y_test2 = train_test_split( data_set2, Y,
test_size=0.2, stratify=Y, random_state=12)
print("X_train2 (60%):", X_train2.shape)
print("X_test2 (20%):",X_test2.shape)

X_train1.head(1)
```

```
X_train1 (60%): (12000, 9)
X_test1 (20%): (3000, 9)
X_train2 (60%): (12000, 9)
X_test2 (20%): (3000, 9)
```

```
     school_state teacher_prefix project_grade_category  \
4870           ca            mrs             grades_6_8

      teacher_number_of_previously_posted_projects clean_categories  \
4870                                             6     math_science

                  clean_subcategories  \
4870  environmentalscience mathematics

                                                 essay  price  \
4870  they come low income families neighborhood yea...  61.19
```

```
      essay_sentimental_score
4870                   0.1779
```

TFIDF vectorization of text data

```
# perform tfidf vectorization of text data.

preprocessed_essays_Xtrain1 = X_train1['essay'].values
preprocessed_essays_Xtest1 = X_test1['essay'].values
vectorizer = TfidfVectorizer(min_df=10)
text_tfidf_Xtrain1 =
vectorizer.fit_transform(preprocessed_essays_Xtrain1)
text_tfidf_Xtest1 = vectorizer.transform(preprocessed_essays_Xtest1)

print("Shape of matrix after one hot encodig (Xtrain1):
",text_tfidf_Xtrain1.shape)
print("Shape of matrix after one hot encodig (Xtest1):
",text_tfidf_Xtest1.shape)

#print(type(text_tfidf_Xtrain1))
```

```
Shape of matrix after one hot encodig (Xtrain1):  (12000, 6721)
Shape of matrix after one hot encodig (Xtest1):  (3000, 6721)
```

```
# 4. perform tfidf w2v vectorization of text data.

preprocessed_essays_Xtrain2 = X_train2['essay'].values
preprocessed_essays_Xtest2 = X_test2['essay'].values
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays_Xtrain2)

# we are converting a dictionary with word as a key, and the tf-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 get_tfidf_w2v(essays):

    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is
stored in this list
    for sentence in tqdm(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/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
        tfidf_w2v_vectors.append(vector)
    return tfidf_w2v_vectors

text_tfidf_w2v_Xtrain2 =
np.array(get_tfidf_w2v(preprocessed_essays_Xtrain2))
text_tfidf_w2v_Xtest2 =
np.array(get_tfidf_w2v(preprocessed_essays_Xtest2))


#print(len(text_tfidf_w2v_Xtrain2))
#print(len(text_tfidf_w2v_Xtrain2[0]))
```

```
100%|
| 12000/12000 [01:11<00:00, 167.53it/s]
100%|
| 3000/3000 [00:15<00:00, 191.34it/s]
```

Encoding of numerical features: Data set1

```
# 6. perform standardizing numerical features
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
# Price
scaler = StandardScaler()
standardized_price_Xtrain1 =
scaler.fit_transform(X_train1['price'].values.reshape(-1, 1))
standardized_price_Xtest1 =
scaler.transform(X_test1['price'].values.reshape(-1, 1))
scaler = MinMaxScaler()
nrm_price_Xtrain1=scaler.fit_transform(standardized_price_Xtrain1)
nrm_price_Xtest1=scaler.transform(standardized_price_Xtest1)

#print(nrm_price_Xtrain1.shape)
#print(type(nrm_price_Xtrain1))
```

```
#print(nrm_price_Xtrain1[:5])
#project_data['nrm_price_Xtrain1'].head()

# teacher_number_of_previously_posted_projects

scaler = StandardScaler()
standardized_teacher_number_of_previously_posted_projects_Xtrain1 =
scaler.fit_transform(X_train1['teacher_number_of_previously_posted_pro
jects'].values.reshape(-1, 1))
standardized_teacher_number_of_previously_posted_projects_Xtest1 =
scaler.transform(X_test1['teacher_number_of_previously_posted_projects
'].values.reshape(-1, 1))
scaler = MinMaxScaler()
nrm_teacher_number_of_previously_posted_projects_Xtrain1=scaler.fit_tr
ansform(standardized_teacher_number_of_previously_posted_projects_Xtra
in1)
nrm_teacher_number_of_previously_posted_projects_Xtest1=scaler.transfo
rm(standardized_teacher_number_of_previously_posted_projects_Xtest1)

# essay_sentimental_score

scaler = StandardScaler()
standardized_essay_sentimental_score_Xtrain1 =
scaler.fit_transform(X_train1['essay_sentimental_score'].values.reshap
e(-1, 1))
standardized_essay_sentimental_score_Xtest1 =
scaler.transform(X_test1['essay_sentimental_score'].values.reshape(-1,
1))
scaler = MinMaxScaler()
nrm_essay_sentimental_score_Xtrain1=scaler.fit_transform(standardized_
essay_sentimental_score_Xtrain1)
nrm_essay_sentimental_score_Xtest1=scaler.transform(standardized_essay
_sentimental_score_Xtest1)
```

Encoding of numerical features: Data set2

```
# 6. perform standardizing numerical features
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
# Price
scaler = StandardScaler()
standardized_price_Xtrain2 =
scaler.fit_transform(X_train2['price'].values.reshape(-1, 1))
standardized_price_Xtest2 =
scaler.transform(X_test2['price'].values.reshape(-1, 1))
scaler = MinMaxScaler()
nrm_price_Xtrain2=scaler.fit_transform(standardized_price_Xtrain2)
```

```
nrm_price_Xtest2=scaler.transform(standardized_price_Xtest2)

#print(nrm_price_Xtrain2.shape)
#print(type(nrm_price_Xtrain2))
#print(nrm_price_Xtrain2[:5])
#project_data['nrm_price_Xtrain2'].head()

# teacher_number_of_previously_posted_projects

scaler = StandardScaler()
standardized_teacher_number_of_previously_posted_projects_Xtrain2 =
scaler.fit_transform(X_train2['teacher_number_of_previously_posted_pro
jects'].values.reshape(-1, 1))
standardized_teacher_number_of_previously_posted_projects_Xtest2 =
scaler.transform(X_test2['teacher_number_of_previously_posted_projects
'].values.reshape(-1, 1))
scaler = MinMaxScaler()
nrm_teacher_number_of_previously_posted_projects_Xtrain2=scaler.fit_tr
ansform(standardized_teacher_number_of_previously_posted_projects_Xtra
in2)
nrm_teacher_number_of_previously_posted_projects_Xtest2=scaler.transfo
rm(standardized_teacher_number_of_previously_posted_projects_Xtest2)

# essay_sentimental_score

scaler = StandardScaler()
standardized_essay_sentimental_score_Xtrain2 =
scaler.fit_transform(X_train2['essay_sentimental_score'].values.reshap
e(-1, 1))
standardized_essay_sentimental_score_Xtest2 =
scaler.transform(X_test2['essay_sentimental_score'].values.reshape(-1,
1))
scaler = MinMaxScaler()
nrm_essay_sentimental_score_Xtrain2=scaler.fit_transform(standardized_
essay_sentimental_score_Xtrain2)
nrm_essay_sentimental_score_Xtest2=scaler.transform(standardized_essay
_sentimental_score_Xtest2)
```

Encoding of categorical features:

Define: Rensponse Encoding (RE)

```
def response_encoding (x,y):
    x = pd.DataFrame(np.array(x).reshape(-1,1))
    y = pd.DataFrame(np.array(y).reshape(-1,1))
    #print(x)
```

```
    x =
pd.DataFrame(np.hstack((x,y)),columns=["feature","class_label"])
    #print(x)
    feature_count = x['feature'].value_counts()
    #print(count)
    feature_dictionary = dict()

    for feature, denominator in feature_count.items():
        vector = []
        for class_label in range(2):
            extract_x_by_class_label =x.loc[ ( x['class_label'] ==
class_label ) & (x['feature'] == feature ) ]
            #print(extract_x_by_class_label)
            #print(len(extract_x_by_class_label)  / denominator )
            vector.append( len(extract_x_by_class_label)  /
denominator   )
        feature_dictionary[feature] = vector
    return feature_dictionary


def re_transform(features,x,y):
    encoded_value=[]
    for each_category in x:
        #print(each_category,dict(categories_count).keys())
        #print(each_category)
        if each_category in features.keys():
            encoded_value.append( features[each_category] )
        else:
            #Laplace smoothing. assign equal probabilities for new
features in test data
            encoded_value.append([0.5, 0.5])
    return encoded_value
```

Encoding of categorical features: Data set1

```
# perform encoding of categorical features.

# School_state
school_state_ohe_features =
response_encoding(X_train1['school_state'], y_train1)
school_state_ohe_Xtrain1 =
np.array(re_transform(school_state_ohe_features,X_train1['school_state
'], y_train1))
school_state_ohe_Xtest1 =
np.array(re_transform(school_state_ohe_features,X_test1['school_state'
], y_test1))

print("Shape of matrix after one hot encoding (Xtrain1: school_state):
",school_state_ohe_Xtrain1.shape)
```

```
print("Shape of matrix after one hot encoding (Xtest1: school_state):
",school_state_ohe_Xtest1.shape)

# teacher_prefix
teacher_prefix_ohe_features =
response_encoding(X_train1['teacher_prefix'], y_train1)
teacher_prefix_ohe_Xtrain1 =
np.array(re_transform(teacher_prefix_ohe_features,X_train1['teacher_pr
efix'], y_train1))
teacher_prefix_ohe_Xtest1 =
np.array(re_transform(teacher_prefix_ohe_features,X_test1['teacher_pre
fix'], y_test1))

print("Shape of matrix after one hot encoding (Xtrain1:
teacher_prefix): ",teacher_prefix_ohe_Xtrain1.shape)
print("Shape of matrix after one hot encoding (Xtest1:
teacher_prefix): ",teacher_prefix_ohe_Xtest1.shape)

# project_grade_category
project_grade_category_ohe_features =
response_encoding(X_train1['project_grade_category'], y_train1)
project_grade_category_ohe_Xtrain1 =
np.array(re_transform(project_grade_category_ohe_features,X_train1['pr
oject_grade_category'], y_train1))
project_grade_category_ohe_Xtest1 =
np.array(re_transform(project_grade_category_ohe_features,X_test1['pro
ject_grade_category'], y_test1))

print("Shape of matrix after one hot encoding (Xtrain1:
project_grade_category): ",project_grade_category_ohe_Xtrain1.shape)
print("Shape of matrix after one hot encoding (Xtest1:
project_grade_category): ",project_grade_category_ohe_Xtest1.shape)

#clean_categories
clean_categories_ohe_features =
response_encoding(X_train1['clean_categories'], y_train1)
clean_categories_ohe_Xtrain1 =
np.array(re_transform(clean_categories_ohe_features,X_train1['clean_ca
tegories'], y_train1))
clean_categories_ohe_Xtest1 =
np.array(re_transform(clean_categories_ohe_features,X_test1['clean_cat
egories'], y_test1))

print("Shape of matrix after one hot encoding (Xtrain1:
clean_categories): ",clean_categories_ohe_Xtrain1.shape)
print("Shape of matrix after one hot encoding (Xtest1:
clean_categories): ",clean_categories_ohe_Xtest1.shape)

#clean_subcategories
```

```
clean_subcategories_ohe_features =
response_encoding(X_train1['clean_subcategories'], y_train1)
clean_subcategories_ohe_Xtrain1 =
np.array(re_transform(clean_subcategories_ohe_features,X_train1['clean
_subcategories'], y_train1))
clean_subcategories_ohe_Xtest1 =
np.array(re_transform(clean_subcategories_ohe_features,X_test1['clean_
subcategories'], y_test1))

print("Shape of matrix after one hot encoding (Xtrain1:
clean_subcategories): ",clean_subcategories_ohe_Xtrain1.shape)
print("Shape of matrix after one hot encoding (Xtest1:
clean_subcategories): ",clean_subcategories_ohe_Xtest1.shape)
```

```
Shape of matrix after one hot encoding (Xtrain1: school_state):
(12000, 2)
Shape of matrix after one hot encoding (Xtest1: school_state):  (3000,
2)
Shape of matrix after one hot encoding (Xtrain1: teacher_prefix):
(12000, 2)
Shape of matrix after one hot encoding (Xtest1: teacher_prefix):
(3000, 2)
Shape of matrix after one hot encoding (Xtrain1:
project_grade_category):  (12000, 2)
Shape of matrix after one hot encoding (Xtest1:
project_grade_category):  (3000, 2)
Shape of matrix after one hot encoding (Xtrain1: clean_categories):
(12000, 2)
Shape of matrix after one hot encoding (Xtest1: clean_categories):
(3000, 2)
Shape of matrix after one hot encoding (Xtrain1: clean_subcategories):
(12000, 2)
Shape of matrix after one hot encoding (Xtest1: clean_subcategories):
(3000, 2)
```

Encoding of categorical features: Data set2

```
# perform encoding of categorical features.

# School_state
school_state_ohe_features =
response_encoding(X_train2['school_state'], y_train2)
school_state_ohe_Xtrain2 =
np.array(re_transform(school_state_ohe_features,X_train2['school_state
'], y_train2))
school_state_ohe_Xtest2 =
np.array(re_transform(school_state_ohe_features,X_test2['school_state'
], y_test2))

print("Shape of matrix after one hot encoding (Xtrain2: school_state):
```

```
",school_state_ohe_Xtrain2.shape)
print("Shape of matrix after one hot encoding (Xtest2: school_state):
",school_state_ohe_Xtest2.shape)

# teacher_prefix
teacher_prefix_ohe_features =
response_encoding(X_train2['teacher_prefix'], y_train2)
teacher_prefix_ohe_Xtrain2 =
np.array(re_transform(teacher_prefix_ohe_features,X_train2['teacher_pr
efix'], y_train2))
teacher_prefix_ohe_Xtest2 =
np.array(re_transform(teacher_prefix_ohe_features,X_test2['teacher_pre
fix'], y_test2))

print("Shape of matrix after one hot encoding (Xtrain2:
teacher_prefix): ",teacher_prefix_ohe_Xtrain2.shape)
print("Shape of matrix after one hot encoding (Xtest2:
teacher_prefix): ",teacher_prefix_ohe_Xtest2.shape)

# project_grade_category
project_grade_category_ohe_features =
response_encoding(X_train2['project_grade_category'], y_train2)
project_grade_category_ohe_Xtrain2 =
np.array(re_transform(project_grade_category_ohe_features,X_train2['pr
oject_grade_category'], y_train2))
project_grade_category_ohe_Xtest2 =
np.array(re_transform(project_grade_category_ohe_features,X_test2['pro
ject_grade_category'], y_test2))

print("Shape of matrix after one hot encoding (Xtrain2:
project_grade_category): ",project_grade_category_ohe_Xtrain2.shape)
print("Shape of matrix after one hot encoding (Xtest2:
project_grade_category): ",project_grade_category_ohe_Xtest2.shape)

#clean_categories
clean_categories_ohe_features =
response_encoding(X_train2['clean_categories'], y_train2)
clean_categories_ohe_Xtrain2 =
np.array(re_transform(clean_categories_ohe_features,X_train2['clean_ca
tegories'], y_train2))
clean_categories_ohe_Xtest2 =
np.array(re_transform(clean_categories_ohe_features,X_test2['clean_cat
egories'], y_test2))

print("Shape of matrix after one hot encoding (Xtrain2:
clean_categories): ",clean_categories_ohe_Xtrain2.shape)
print("Shape of matrix after one hot encoding (Xtest2:
clean_categories): ",clean_categories_ohe_Xtest2.shape)
```

```
#clean_subcategories
clean_subcategories_ohe_features =
response_encoding(X_train2['clean_subcategories'], y_train2)
clean_subcategories_ohe_Xtrain2 =
np.array(re_transform(clean_subcategories_ohe_features,X_train2['clean
_subcategories'], y_train2))
clean_subcategories_ohe_Xtest2 =
np.array(re_transform(clean_subcategories_ohe_features,X_test2['clean_
subcategories'], y_test2))

print("Shape of matrix after one hot encoding (Xtrain2:
clean_subcategories): ",clean_subcategories_ohe_Xtrain2.shape)
print("Shape of matrix after one hot encoding (Xtest2:
clean_subcategories): ",clean_subcategories_ohe_Xtest2.shape)
```

```
Shape of matrix after one hot encoding (Xtrain2: school_state):
(12000, 2)
Shape of matrix after one hot encoding (Xtest2: school_state):  (3000,
2)
Shape of matrix after one hot encoding (Xtrain2: teacher_prefix):
(12000, 2)
Shape of matrix after one hot encoding (Xtest2: teacher_prefix):
(3000, 2)
Shape of matrix after one hot encoding (Xtrain2:
project_grade_category):  (12000, 2)
Shape of matrix after one hot encoding (Xtest2:
project_grade_category):  (3000, 2)
Shape of matrix after one hot encoding (Xtrain2: clean_categories):
(12000, 2)
Shape of matrix after one hot encoding (Xtest2: clean_categories):
(3000, 2)
Shape of matrix after one hot encoding (Xtrain2: clean_subcategories):
(12000, 2)
Shape of matrix after one hot encoding (Xtest2: clean_subcategories):
(3000, 2)
```

```
#clean_subcategories_ohe_Xtrain2
```

Stacking features: Data set1

```
# 7. For task 1 set 1 stack up all the features

#Xtrain of data set1

#convert school_state_ohe sparse matrix to dense
#print(school_state_ohe_Xtrain1.shape)
column_names = [ "school_ohe_"+str(i) for i in
range(school_state_ohe_Xtrain1.shape[1])]
#print(column_names)
school_state_ohe_Xtrain1_df =
```

```
pd.DataFrame(school_state_ohe_Xtrain1,columns =column_names)
#school_state_ohe_df.column = column_names
school_state_ohe_Xtrain1_df.head(2)


#convert teacher_prefix_ohe_Xtrain1 sparse matrix to dense
#print(teacher_prefix_ohe_Xtrain1.shape)
column_names = [ "teacher_prefix_ohe_"+str(i) for i in
range(teacher_prefix_ohe_Xtrain1.shape[1])]
#print(column_names)
teacher_prefix_ohe_Xtrain1_df =
pd.DataFrame(teacher_prefix_ohe_Xtrain1,columns =column_names)
#school_state_ohe_df.column = column_names
teacher_prefix_ohe_Xtrain1_df.head(2)


#convert project_grade_category_ohe_Xtrain1 sparse matrix to dense
#print(project_grade_category_ohe_Xtrain1.shape)
column_names = [ "project_grade_category_ohe_"+str(i) for i in
range(project_grade_category_ohe_Xtrain1.shape[1])]
#print(column_names)
project_grade_category_ohe_Xtrain1_df =
pd.DataFrame(project_grade_category_ohe_Xtrain1,columns =column_names)
#school_state_ohe_df.column = column_names
project_grade_category_ohe_Xtrain1_df.head(2)

#convert clean_categories_ohe_Xtrain1 sparse matrix to dense
#print(clean_categories_ohe_Xtrain1.shape)
column_names = [ "clean_categories_ohe_"+str(i) for i in
range(clean_categories_ohe_Xtrain1.shape[1])]
#print(column_names)
clean_categories_ohe_Xtrain1_df =
pd.DataFrame(clean_categories_ohe_Xtrain1,columns =column_names)
#school_state_ohe_df.column = column_names
clean_categories_ohe_Xtrain1_df.head(2)

#convert clean_subcategories_ohe_Xtrain1 sparse matrix to dense
#print(clean_subcategories_ohe_Xtrain1.shape)
column_names = [ "clean_subcategories_ohe_"+str(i) for i in
range(clean_subcategories_ohe_Xtrain1.shape[1])]
#print(column_names)
clean_subcategories_ohe_Xtrain1_df =
pd.DataFrame(clean_subcategories_ohe_Xtrain1,columns =column_names)
#school_state_ohe_df.column = column_names
clean_subcategories_ohe_Xtrain1_df.head(2)

#convert text_tfidf sparse matrix to dense
#print(text_tfidf.shape)
column_names = [ "text_tfidf"+str(i) for i in
```

```
range(text_tfidf_Xtrain1.shape[1])]
#print(column_names)
text_tfidf_Xtrain1_df =
pd.DataFrame(text_tfidf_Xtrain1.todense(),columns =column_names)
#school_state_ohe_df.column = column_names
text_tfidf_Xtrain1_df.head(2)

#sent_score =X_train1.essay_sentimental_score

X_train1_df =
pd.concat([school_state_ohe_Xtrain1_df,teacher_prefix_ohe_Xtrain1_df,p
roject_grade_category_ohe_Xtrain1_df,

pd.DataFrame(nrm_teacher_number_of_previously_posted_projects_Xtrain1,
columns =["nrm_teacher_number_of_previously_posted_projects"]),


clean_categories_ohe_Xtrain1_df,clean_subcategories_ohe_Xtrain1_df,
                      text_tfidf_Xtrain1_df,
                      pd.DataFrame(nrm_price_Xtrain1,columns
=["nrm_price"]),

pd.DataFrame(nrm_essay_sentimental_score_Xtrain1,columns
=["nrm_essay_sentimental_score"])],axis=1)
#project_grade_category_ohe_df.head(2)
print("Xtrain of Data Set 1 ")
print("-"*50)
print("size: ",X_train1_df.shape)
X_train1_df.head(2)

#print(X_train1_df.shape)
```

```
Xtrain of Data Set 1
--------------------------------------------------
size:  (12000, 6734)

   school_ohe_0  school_ohe_1  teacher_prefix_ohe_0
teacher_prefix_ohe_1  \
0      0.116014      0.883986              0.136145
0.863855
1      0.142061      0.857939              0.136145
0.863855

   project_grade_category_ohe_0  project_grade_category_ohe_1  \
0                      0.149678                      0.850322
1                      0.149506                      0.850494

   nrm_teacher_number_of_previously_posted_projects
clean_categories_ohe_0  \
0                                          0.017391
```

```
0.169579
1                                          0.234783
0.166144

   clean_categories_ohe_1  clean_subcategories_ohe_0  \
0                0.830421                   0.280374
1                0.833856                   0.100000

              ...               text_tfidf6713  text_tfidf6714  \
0             ...                          0.0             0.0
1             ...                          0.0             0.0

   text_tfidf6715  text_tfidf6716  text_tfidf6717  text_tfidf6718  \
0             0.0             0.0             0.0             0.0
1             0.0             0.0             0.0             0.0

   text_tfidf6719  text_tfidf6720  nrm_price
nrm_essay_sentimental_score
0             0.0             0.0   0.006030
0.586803
1             0.0             0.0   0.072878
0.974049

[2 rows x 6734 columns]
```

```
# 7. For task 1 set 1 stack up all the features

#Xtest of data set1

#convert school_state_ohe sparse matrix to dense
#print(school_state_ohe_Xtest1.shape)
column_names = [ "school_ohe_"+str(i) for i in
range(school_state_ohe_Xtest1.shape[1])]
#print(column_names)
school_state_ohe_Xtest1_df =
pd.DataFrame(school_state_ohe_Xtest1,columns =column_names)
#school_state_ohe_df.column = column_names
school_state_ohe_Xtest1_df.head(2)


#convert teacher_prefix_ohe_Xtest1 sparse matrix to dense
#print(teacher_prefix_ohe_Xtest1.shape)
column_names = [ "teacher_prefix_ohe_"+str(i) for i in
range(teacher_prefix_ohe_Xtest1.shape[1])]
#print(column_names)
teacher_prefix_ohe_Xtest1_df =
pd.DataFrame(teacher_prefix_ohe_Xtest1,columns =column_names)
#school_state_ohe_df.column = column_names
teacher_prefix_ohe_Xtest1_df.head(2)
```

```
#convert project_grade_category_ohe_Xtest1 sparse matrix to dense
#print(project_grade_category_ohe_Xtest1.shape)
column_names = [ "project_grade_category_ohe_"+str(i) for i in
range(project_grade_category_ohe_Xtest1.shape[1])]
#print(column_names)
project_grade_category_ohe_Xtest1_df =
pd.DataFrame(project_grade_category_ohe_Xtest1,columns =column_names)
#school_state_ohe_df.column = column_names
project_grade_category_ohe_Xtest1_df.head(2)

#convert clean_categories_ohe_Xtest1 sparse matrix to dense
#print(clean_categories_ohe_Xtest1.shape)
column_names = [ "clean_categories_ohe_"+str(i) for i in
range(clean_categories_ohe_Xtest1.shape[1])]
#print(column_names)
clean_categories_ohe_Xtest1_df =
pd.DataFrame(clean_categories_ohe_Xtest1,columns =column_names)
#school_state_ohe_df.column = column_names
clean_categories_ohe_Xtest1_df.head(2)

#convert clean_subcategories_ohe_Xtest1 sparse matrix to dense
#print(clean_subcategories_ohe_Xtest1.shape)
column_names = [ "clean_subcategories_ohe_"+str(i) for i in
range(clean_subcategories_ohe_Xtest1.shape[1])]
#print(column_names)
clean_subcategories_ohe_Xtest1_df =
pd.DataFrame(clean_subcategories_ohe_Xtest1,columns =column_names)
#school_state_ohe_df.column = column_names
clean_subcategories_ohe_Xtest1_df.head(2)

#convert text_tfidf sparse matrix to dense
#print(text_tfidf.shape)
column_names = [ "text_tfidf"+str(i) for i in
range(text_tfidf_Xtest1.shape[1])]
#print(column_names)
text_tfidf_Xtest1_df =
pd.DataFrame(text_tfidf_Xtest1.todense(),columns =column_names)
#school_state_ohe_df.column = column_names
text_tfidf_Xtest1_df.head(2)

#sent_score =X_train1.essay_sentimental_score

X_test1_df =
pd.concat([school_state_ohe_Xtest1_df,teacher_prefix_ohe_Xtest1_df,pro
ject_grade_category_ohe_Xtest1_df,

pd.DataFrame(nrm_teacher_number_of_previously_posted_projects_Xtest1,c
```

```
olumns =["nrm_teacher_number_of_previously_posted_projects"]),


clean_categories_ohe_Xtest1_df,clean_subcategories_ohe_Xtest1_df,
                       text_tfidf_Xtest1_df,
                       pd.DataFrame(nrm_price_Xtest1,columns
=["nrm_price"]),

pd.DataFrame(nrm_essay_sentimental_score_Xtest1,columns
=["nrm_essay_sentimental_score"])],axis=1)
#project_grade_category_ohe_df.head(2)
print("Xtest of Data Set 1 ")
print("-"*50)
print("size: ",X_test1_df.shape)
X_train1_df.head(2)

#print(X_train1_df.shape)
```

```
Xtest of Data Set 1
--------------------------------------------------
size:  (3000, 6734)

   school_ohe_0  school_ohe_1  teacher_prefix_ohe_0
teacher_prefix_ohe_1  \
0      0.116014      0.883986              0.136145
0.863855
1      0.142061      0.857939              0.136145
0.863855

   project_grade_category_ohe_0  project_grade_category_ohe_1  \
0                      0.149678                      0.850322
1                      0.149506                      0.850494

   nrm_teacher_number_of_previously_posted_projects
clean_categories_ohe_0  \
0                                          0.017391
0.169579
1                                          0.234783
0.166144

   clean_categories_ohe_1  clean_subcategories_ohe_0  \
0                0.830421                   0.280374
1                0.833856                   0.100000

               ...               text_tfidf6713  text_tfidf6714  \
0              ...                          0.0             0.0
1              ...                          0.0             0.0

   text_tfidf6715  text_tfidf6716  text_tfidf6717  text_tfidf6718  \
0             0.0             0.0             0.0             0.0
```

```
1             0.0             0.0             0.0             0.0

   text_tfidf6719  text_tfidf6720  nrm_price
nrm_essay_sentimental_score
0             0.0             0.0   0.006030
0.586803
1             0.0             0.0   0.072878
0.974049

[2 rows x 6734 columns]
```

Stacking features: Dataset2

```
# 7. For task 2 set 2 stack up all the features

#Xtrain of data set2

#convert school_state_ohe sparse matrix to dense
#print(school_state_ohe_Xtrain2.shape)
column_names = [ "school_ohe_"+str(i) for i in
range(school_state_ohe_Xtrain2.shape[1])]
#print(column_names)
school_state_ohe_Xtrain2_df =
pd.DataFrame(school_state_ohe_Xtrain2,columns =column_names)
#school_state_ohe_df.column = column_names
school_state_ohe_Xtrain2_df.head(2)


#convert teacher_prefix_ohe_Xtrain2 sparse matrix to dense
#print(teacher_prefix_ohe_Xtrain2.shape)
column_names = [ "teacher_prefix_ohe_"+str(i) for i in
range(teacher_prefix_ohe_Xtrain2.shape[1])]
#print(column_names)
teacher_prefix_ohe_Xtrain2_df =
pd.DataFrame(teacher_prefix_ohe_Xtrain2,columns =column_names)
#school_state_ohe_df.column = column_names
teacher_prefix_ohe_Xtrain2_df.head(2)


#convert project_grade_category_ohe_Xtrain2 sparse matrix to dense
#print(project_grade_category_ohe_Xtrain2.shape)
column_names = [ "project_grade_category_ohe_"+str(i) for i in
range(project_grade_category_ohe_Xtrain2.shape[1])]
#print(column_names)
project_grade_category_ohe_Xtrain2_df =
pd.DataFrame(project_grade_category_ohe_Xtrain2,columns =column_names)
#school_state_ohe_df.column = column_names
project_grade_category_ohe_Xtrain2_df.head(2)
```

```
#convert clean_categories_ohe_Xtrain2 sparse matrix to dense
#print(clean_categories_ohe_Xtrain2.shape)
column_names = [ "clean_categories_ohe_"+str(i) for i in
range(clean_categories_ohe_Xtrain2.shape[1])]
#print(column_names)
clean_categories_ohe_Xtrain2_df =
pd.DataFrame(clean_categories_ohe_Xtrain2,columns =column_names)
#school_state_ohe_df.column = column_names
clean_categories_ohe_Xtrain2_df.head(2)

#convert clean_subcategories_ohe_Xtrain2 sparse matrix to dense
#print(clean_subcategories_ohe_Xtrain2.shape)
column_names = [ "clean_subcategories_ohe_"+str(i) for i in
range(clean_subcategories_ohe_Xtrain2.shape[1])]
#print(column_names)
clean_subcategories_ohe_Xtrain2_df =
pd.DataFrame(clean_subcategories_ohe_Xtrain2,columns =column_names)
#school_state_ohe_df.column = column_names
clean_subcategories_ohe_Xtrain2_df.head(2)

#convert text_tfidf sparse matrix to dense
#print(text_tfidf.shape)
column_names = [ "text_tfidf"+str(i) for i in
range(text_tfidf_w2v_Xtrain2.shape[1])]
#print(column_names)
text_tfidf_w2v_Xtrain2_df =
pd.DataFrame(text_tfidf_w2v_Xtrain2,columns =column_names)
#school_state_ohe_df.column = column_names
text_tfidf_w2v_Xtrain2_df.head(2)

#sent_score =X_train2.essay_sentimental_score

X_train2_df =
pd.concat([school_state_ohe_Xtrain2_df,teacher_prefix_ohe_Xtrain2_df,p
roject_grade_category_ohe_Xtrain2_df,

pd.DataFrame(nrm_teacher_number_of_previously_posted_projects_Xtrain2,
columns =["nrm_teacher_number_of_previously_posted_projects"]),


clean_categories_ohe_Xtrain2_df,clean_subcategories_ohe_Xtrain2_df,
                        text_tfidf_w2v_Xtrain2_df,
                        pd.DataFrame(nrm_price_Xtrain2,columns
=["nrm_price"]),

pd.DataFrame(nrm_essay_sentimental_score_Xtrain2,columns
=["nrm_essay_sentimental_score"])],axis=1)
#project_grade_category_ohe_df.head(2)
print("Xtrain of Data Set 2 ")
```

```
print("-"*50)
print("size: ",X_train2_df.shape)
X_train2_df.head(2)

#print(X_train2_df.shape)
```

```
Xtrain of Data Set 2
--------------------------------------------------
size:  (12000, 313)
```

```
   school_ohe_0  school_ohe_1  teacher_prefix_ohe_0
teacher_prefix_ohe_1  \
0      0.116014      0.883986              0.136145
0.863855
1      0.142061      0.857939              0.136145
0.863855

   project_grade_category_ohe_0  project_grade_category_ohe_1  \
0                      0.149678                      0.850322
1                      0.149506                      0.850494

   nrm_teacher_number_of_previously_posted_projects
clean_categories_ohe_0  \
0                                          0.017391
0.169579
1                                          0.234783
0.166144

   clean_categories_ohe_1  clean_subcategories_ohe_0  \
0                0.830421                   0.280374
1                0.833856                   0.100000

               ...               text_tfidf292  text_tfidf293
text_tfidf294  \
0              ...                   -0.094487      -0.007660
0.115856
1              ...                   -0.035168      -0.092808      -
0.059483

   text_tfidf295  text_tfidf296  text_tfidf297  text_tfidf298
text_tfidf299  \
0       0.040882      -0.010143       0.152234       0.202007
0.096801
1       0.026922      -0.050343       0.245331       0.191024
0.094191

   nrm_price  nrm_essay_sentimental_score
0   0.006030                     0.586803
1   0.072878                     0.974049
```

```
[2 rows x 313 columns]
```

```
# 7. For task 2 set 2 stack up all the features

#Xtrain of data set2

#convert school_state_ohe sparse matrix to dense
#print(school_state_ohe_Xtest2.shape)
column_names = [ "school_ohe_"+str(i) for i in
range(school_state_ohe_Xtest2.shape[1])]
#print(column_names)
school_state_ohe_Xtest2_df =
pd.DataFrame(school_state_ohe_Xtest2,columns =column_names)
#school_state_ohe_df.column = column_names
school_state_ohe_Xtest2_df.head(2)


#convert teacher_prefix_ohe_Xtest2 sparse matrix to dense
#print(teacher_prefix_ohe_Xtest2.shape)
column_names = [ "teacher_prefix_ohe_"+str(i) for i in
range(teacher_prefix_ohe_Xtest2.shape[1])]
#print(column_names)
teacher_prefix_ohe_Xtest2_df =
pd.DataFrame(teacher_prefix_ohe_Xtest2,columns =column_names)
#school_state_ohe_df.column = column_names
teacher_prefix_ohe_Xtest2_df.head(2)


#convert project_grade_category_ohe_Xtest2 sparse matrix to dense
#print(project_grade_category_ohe_Xtest2.shape)
column_names = [ "project_grade_category_ohe_"+str(i) for i in
range(project_grade_category_ohe_Xtest2.shape[1])]
#print(column_names)
project_grade_category_ohe_Xtest2_df =
pd.DataFrame(project_grade_category_ohe_Xtest2,columns =column_names)
#school_state_ohe_df.column = column_names
project_grade_category_ohe_Xtest2_df.head(2)

#convert clean_categories_ohe_Xtest2 sparse matrix to dense
#print(clean_categories_ohe_Xtest2.shape)
column_names = [ "clean_categories_ohe_"+str(i) for i in
range(clean_categories_ohe_Xtest2.shape[1])]
#print(column_names)
clean_categories_ohe_Xtest2_df =
pd.DataFrame(clean_categories_ohe_Xtest2,columns =column_names)
#school_state_ohe_df.column = column_names
clean_categories_ohe_Xtest2_df.head(2)
```

```
#convert clean_subcategories_ohe_Xtest2 sparse matrix to dense
#print(clean_subcategories_ohe_Xtest2.shape)
column_names = [ "clean_subcategories_ohe_"+str(i) for i in
range(clean_subcategories_ohe_Xtest2.shape[1])]
#print(column_names)
clean_subcategories_ohe_Xtest2_df =
pd.DataFrame(clean_subcategories_ohe_Xtest2,columns =column_names)
#school_state_ohe_df.column = column_names
clean_subcategories_ohe_Xtest2_df.head(2)

#convert text_tfidf sparse matrix to dense
#print(text_tfidf.shape)
column_names = [ "text_tfidf"+str(i) for i in
range(text_tfidf_w2v_Xtest2.shape[1])]
#print(column_names)
text_tfidf_w2v_Xtest2_df = pd.DataFrame(text_tfidf_w2v_Xtest2,columns
=column_names)
#school_state_ohe_df.column = column_names
text_tfidf_w2v_Xtest2_df.head(2)

#sent_score =X_test2.essay_sentimental_score

X_test2_df =
pd.concat([school_state_ohe_Xtest2_df,teacher_prefix_ohe_Xtest2_df,pro
ject_grade_category_ohe_Xtest2_df,

pd.DataFrame(nrm_teacher_number_of_previously_posted_projects_Xtest2,c
olumns =["nrm_teacher_number_of_previously_posted_projects"]),


clean_categories_ohe_Xtest2_df,clean_subcategories_ohe_Xtest2_df,
                        text_tfidf_w2v_Xtest2_df,
                        pd.DataFrame(nrm_price_Xtest2,columns
=["nrm_price"]),

pd.DataFrame(nrm_essay_sentimental_score_Xtest2,columns
=["nrm_essay_sentimental_score"])],axis=1)
#project_grade_category_ohe_df.head(2)
print("Xtest of Data Set 2 ")
print("-"*50)
print("size: ",X_test2_df.shape)
X_test2_df.head(2)

#print(X_test2_df.shape)
```

```
Xtest of Data Set 2
--------------------------------------------------
size:  (3000, 313)
```

```
   school_ohe_0  school_ohe_1  teacher_prefix_ohe_0
teacher_prefix_ohe_1  \
0      0.116608      0.883392              0.151037
0.848963
1      0.116014      0.883986              0.136145
0.863855

   project_grade_category_ohe_0  project_grade_category_ohe_1  \
0                      0.149506                      0.850494
1                      0.149506                      0.850494

   nrm_teacher_number_of_previously_posted_projects
clean_categories_ohe_0  \
0                                          0.000000
0.129118
1                                          0.344928
0.189655

   clean_categories_ohe_1  clean_subcategories_ohe_0  \
0                0.870882                   0.134293
1                0.810345                   0.187500

               ...               text_tfidf292  text_tfidf293
text_tfidf294  \
0              ...                    0.074125      -0.023337
0.032072
1              ...                   -0.018554      -0.109886
0.015966

   text_tfidf295  text_tfidf296  text_tfidf297  text_tfidf298
text_tfidf299  \
0       0.023109       0.057476       0.077482       0.142718
0.140667
1       0.047865      -0.006239       0.083502      -0.009681
0.049241

   nrm_price  nrm_essay_sentimental_score
0   0.001009                     0.938845
1   0.005765                     0.997888

[2 rows x 313 columns]
```

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

Applying GBDT on Dataset1

```
#The hyper paramter tuning (best depth in range [1, 3, 10, 30], and
the best min_samples_split in range [5, 10, 100, 500])
```

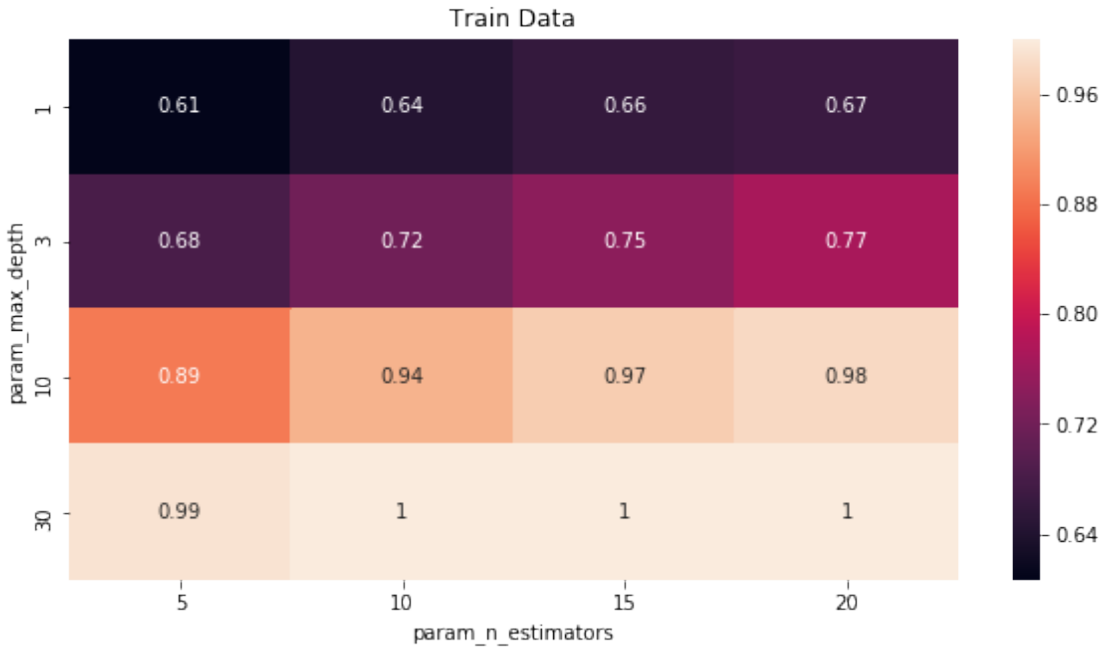
```
#Find the best hyper parameter which will give the maximum AUC value
#find the best hyper paramter using k-fold cross validation(use
gridsearch cv or randomsearch cv)/simple cross validation data(you can
write your own for loops refer sample solution)

from sklearn.ensemble import GradientBoostingClassifier

# fit the model with best parameters found
clf = GradientBoostingClassifier(random_state =20)
clf.fit(X_train1_df,y_train1)

parameters = {'max_depth':[1, 3, 10, 30],'n_estimators': [5, 10, 15,
20]}
GridSearch_clf = GridSearchCV(clf, parameters,scoring='roc_auc',cv
=3,return_train_score=True)
GridSearch_clf.fit(X_train1_df, y_train1)

print(GridSearch_clf.score(X_train1_df,y_train1))
print("------------Best Hyperparameters----------------")
# print best parameter after tuning
print(GridSearch_clf.best_params_)

# print how our model looks after hyper-parameter tuning
print(GridSearch_clf.best_estimator_)

#print(GridSearch_clf.cv_results_)

best_max_depth = GridSearch_clf.best_params_['max_depth']

best_n_estimators = GridSearch_clf.best_params_['n_estimators']
```

```
0.7462051644742902
------------Best Hyperparameters---------------
{'max_depth': 3, 'n_estimators': 20}
GradientBoostingClassifier(n_estimators=20, random_state=20)
```

```
#Perform hyperparameter tuning and plot either heatmap or 3d plot.
#On Train data
temp =
np.stack((GridSearch_clf.cv_results_['param_max_depth'],GridSearch_clf
.cv_results_['param_n_estimators'],GridSearch_clf.cv_results_['mean_tr
ain_score']),axis =1)
#print(temp)
scores = (pd.DataFrame(temp,columns =
['param_max_depth','param_n_estimators','mean_train_score']).groupby([
'param_max_depth','param_n_estimators'])).max().unstack()
print(scores)
fig = plt.figure(figsize=(10,5))
sns.heatmap(scores.mean_train_score,annot = True)
```

```
plt.title("Train Data")
plt.show()
```

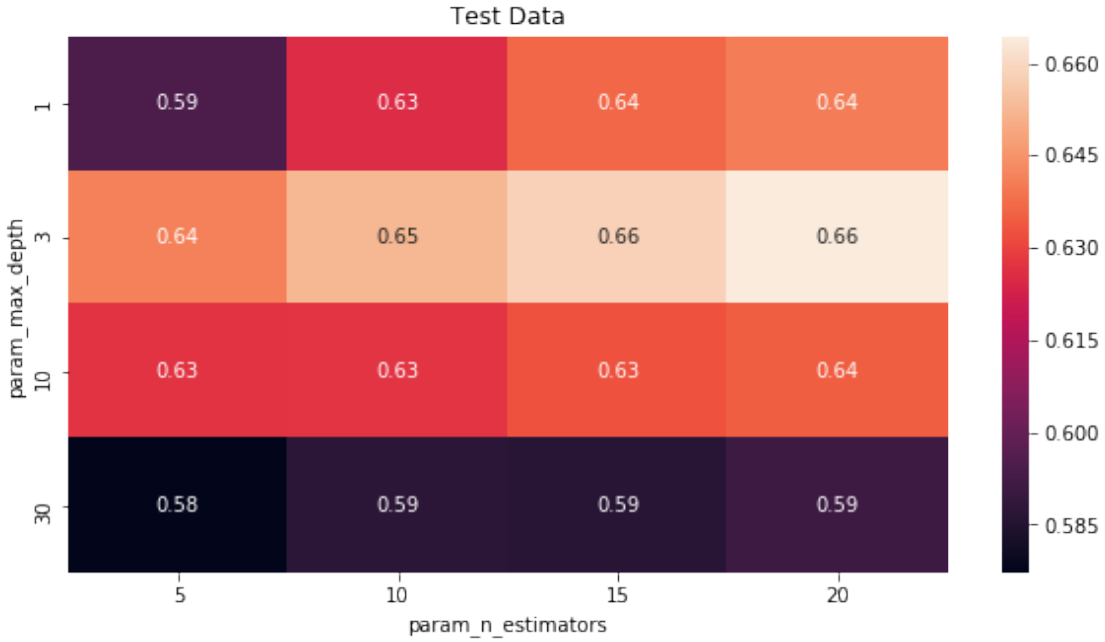
	mean_train_score			
param_n_estimators	5	10	15	20
param_max_depth				
1	0.606385	0.644029	0.661432	0.666871
3	0.682969	0.719166	0.745176	0.769903
10	0.889630	0.942125	0.969064	0.981302
30	0.991392	0.999828	1.000000	1.000000




```
#Perform hyperparameter tuning and plot either heatmap or 3d plot.
#On CV data
temp =
np.stack((GridSearch_clf.cv_results_['param_max_depth'],GridSearch_clf
.cv_results_['param_n_estimators'],GridSearch_clf.cv_results_['mean_te
st_score']),axis =1)
#print(temp)
scores = (pd.DataFrame(temp,columns =
['param_max_depth','param_n_estimators','mean_test_score']).groupby(['
param_max_depth','param_n_estimators'])).max().unstack()
print(scores)
fig = plt.figure(figsize=(10,5))
sns.heatmap(scores.mean_test_score,annot = True)
plt.title("Test Data")
plt.show()
```

	mean_test_score			
param_n_estimators	5	10	15	20
param_max_depth				
1	0.594815	0.625586	0.636424	0.640689

```
3                       0.641381  0.652509  0.658515  0.664340
10                      0.627154  0.627459  0.632910  0.635028
30                      0.577154  0.587349  0.586501  0.590930
```

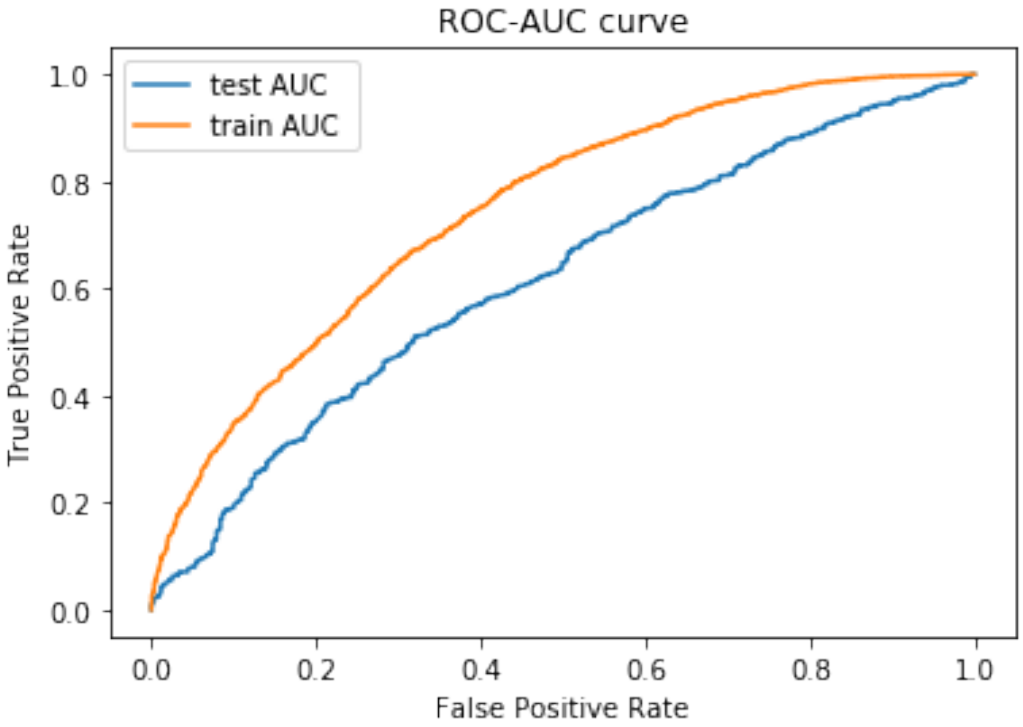



```
# 10. Find the best parameters and fit the model. Plot ROC-AUC
curve(using predict proba method)
# Re-fit the model with best parameters found
clf = GradientBoostingClassifier(max_depth =
best_max_depth ,n_estimators =best_n_estimators )
clf.fit(X_train1_df,y_train1)
```

```
GradientBoostingClassifier(n_estimators=20)
```

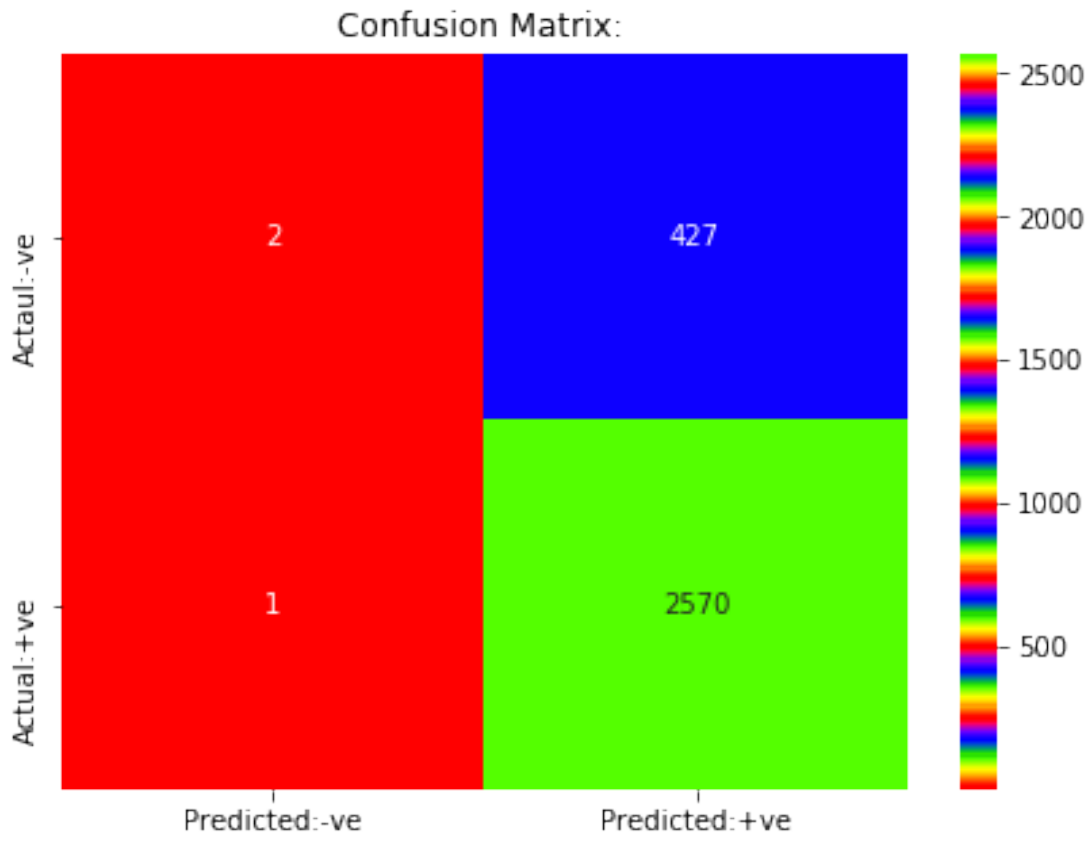
```
#create ROC curve

y_test_pred_proba = clf.predict_proba(X_test1_df)[::,1]
fpr_test, tpr_test, threshold_test = metrics.roc_curve(y_test1,
y_test_pred_proba)

y_train_pred_proba = clf.predict_proba(X_train1_df)[::,1]
fpr_train, tpr_train, threshold_train = metrics.roc_curve(y_train1,
y_train_pred_proba)

#roc_auc = auc(false_positive_rate, true_positive_rate)
plt.plot(fpr_test,tpr_test,label="test AUC ")
plt.plot(fpr_train,tpr_train,label="train AUC ")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title("ROC-AUC curve")
```

```
plt.legend()
plt.show()
#print(threshold_test)
#print(y_pred_proba)
```




```
# 11. Plot confusion matrix based on best threshold value
y_pred = clf.predict(X_test1_df)
cm = confusion_matrix(y_test1,  y_pred)

cm_df =pd.DataFrame(cm,columns=["Predicted:-ve","Predicted:+ve"],index
= ["Actaul:-ve","Actual:+ve"])
print(cm_df)
print("***************confusion matrix***********************")

fig = plt.figure(figsize=(7,5))
ax =sns.heatmap(cm_df,annot =True, xticklabels=True,
yticklabels=True,cmap="prism",fmt="")
plt.title("Confusion Matrix: ")
plt.show()
```

```
            Predicted:-ve  Predicted:+ve
Actaul:-ve              2            427
Actual:+ve              1           2570
***************confusion matrix***********************
```



Applying GBDT on Dataset2

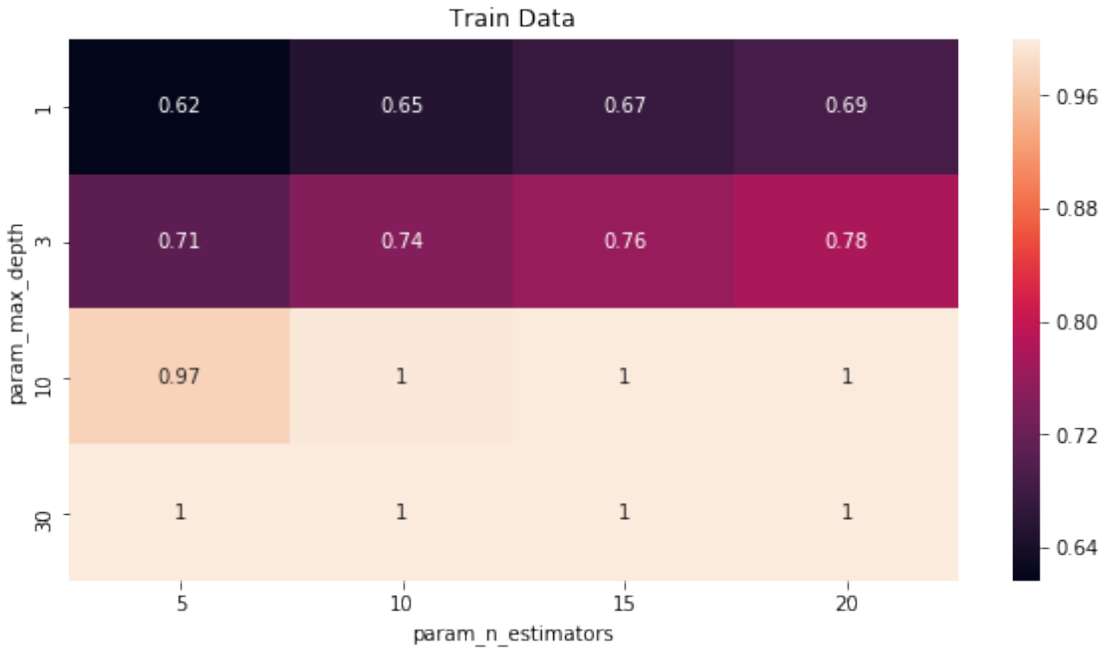
```
#The hyper paramter tuning (best depth in range [1, 3, 10, 30], and
the best min_samples_split in range [5, 10, 100, 500])
#Find the best hyper parameter which will give the maximum AUC value
#find the best hyper paramter using k-fold cross validation(use
gridsearch cv or randomsearch cv)/simple cross validation data(you can
write your own for loops refer sample solution)

clf = GradientBoostingClassifier(random_state =20)
clf.fit(X_train2_df,y_train2)

parameters = {'max_depth':[1, 3, 10, 30],'n_estimators': [5, 10, 15,
20]}
GridSearch_clf = GridSearchCV(clf, parameters,scoring='roc_auc',cv
=3,return_train_score=True)
GridSearch_clf.fit(X_train2_df, y_train2)

print(GridSearch_clf.score(X_train2_df,y_train2))
print("------------Best Hyperparameters----------------")
# print best parameter after tuning
print(GridSearch_clf.best_params_)
```

```
# print how our model looks after hyper-parameter tuning
print(GridSearch_clf.best_estimator_)

#print(GridSearch_clf.cv_results_)

best_max_depth = GridSearch_clf.best_params_['max_depth']

best_n_estimators = GridSearch_clf.best_params_['n_estimators']
```

```
0.7563047036996616
------------Best Hyperparameters---------------
{'max_depth': 3, 'n_estimators': 20}
GradientBoostingClassifier(n_estimators=20, random_state=20)
```

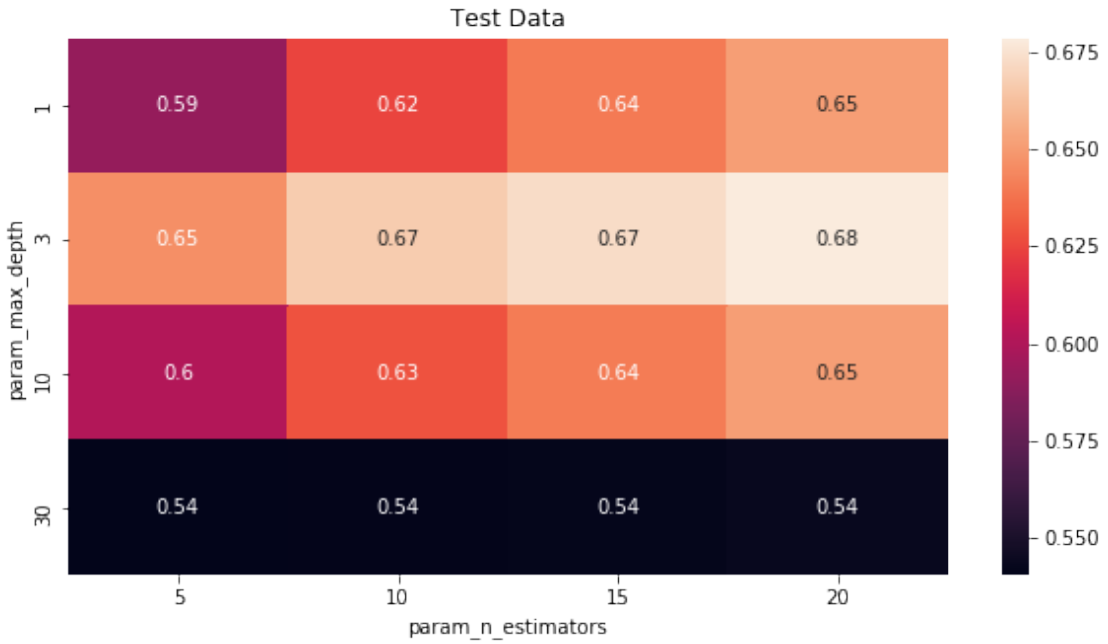
```
#Perform hyperparameter tuning and plot either heatmap or 3d plot.
#On Train data
temp =
np.stack((GridSearch_clf.cv_results_['param_max_depth'],GridSearch_clf
.cv_results_['param_n_estimators'],GridSearch_clf.cv_results_['mean_tr
ain_score']),axis =1)
#print(temp)
scores = (pd.DataFrame(temp,columns =
['param_max_depth','param_n_estimators','mean_train_score']).groupby([
'param_max_depth','param_n_estimators'])).max().unstack()
print(scores)
fig = plt.figure(figsize=(10,5))
sns.heatmap(scores.mean_train_score,annot = True)
plt.title("Train Data")
plt.show()
```

	mean_train_score			
param_n_estimators	5	10	15	20
param_max_depth				
1	0.616383	0.652849	0.672613	0.687271
3	0.708949	0.743993	0.761677	0.778859
10	0.974607	0.996165	0.999810	0.999977
30	1.000000	1.000000	1.000000	1.000000




```
#Perform hyperparameter tuning and plot either heatmap or 3d plot.
#On CV data
temp =
np.stack((GridSearch_clf.cv_results_['param_max_depth'],GridSearch_clf
.cv_results_['param_n_estimators'],GridSearch_clf.cv_results_['mean_te
st_score']),axis =1)
#print(temp)
scores = (pd.DataFrame(temp,columns =
['param_max_depth','param_n_estimators','mean_test_score']).groupby(['
param_max_depth','param_n_estimators'])).max().unstack()
print(scores)
fig = plt.figure(figsize=(10,5))
sns.heatmap(scores.mean_test_score,annot = True)
plt.title("Test Data")
plt.show()
```

	mean_test_score			
param_n_estimators	5	10	15	20
param_max_depth				
1	0.592035	0.624873	0.640073	0.651263
3	0.646453	0.666849	0.672519	0.678383
10	0.601167	0.628750	0.640541	0.651424
30	0.540550	0.541252	0.541831	0.543330

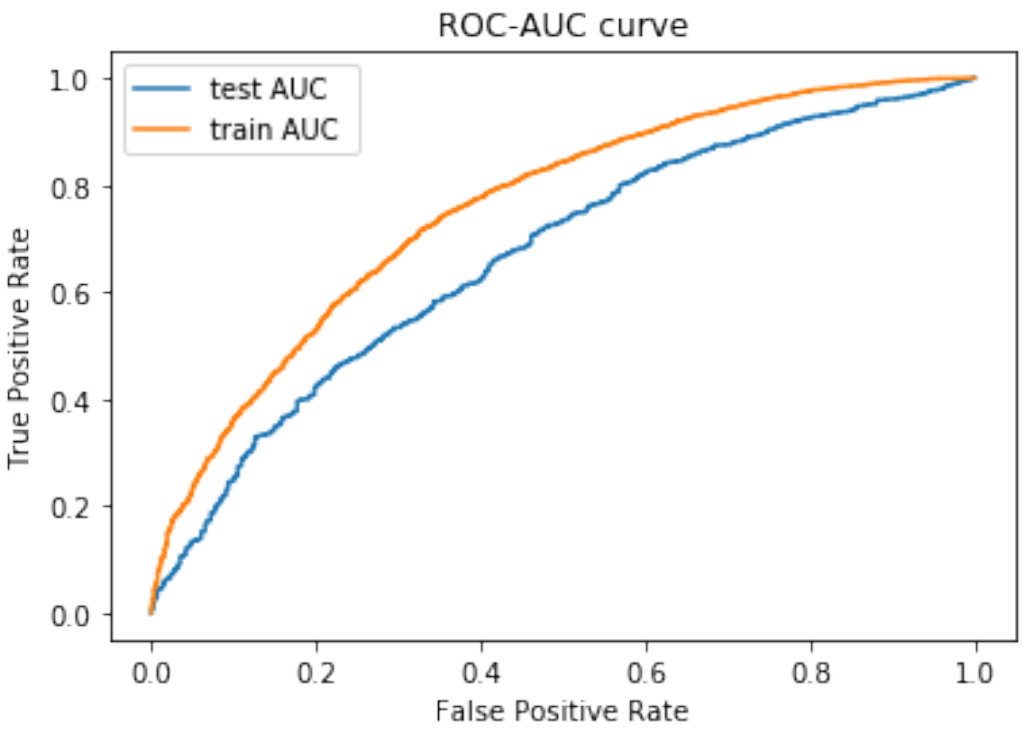



```
# 10. Find the best parameters and fit the model. Plot ROC-AUC
curve(using predict proba method)
# Re-fit the model with best parameters found
clf = GradientBoostingClassifier(max_depth =
best_max_depth ,n_estimators =best_n_estimators )
clf.fit(X_train2_df,y_train2)
```

```
GradientBoostingClassifier(n_estimators=20)
```

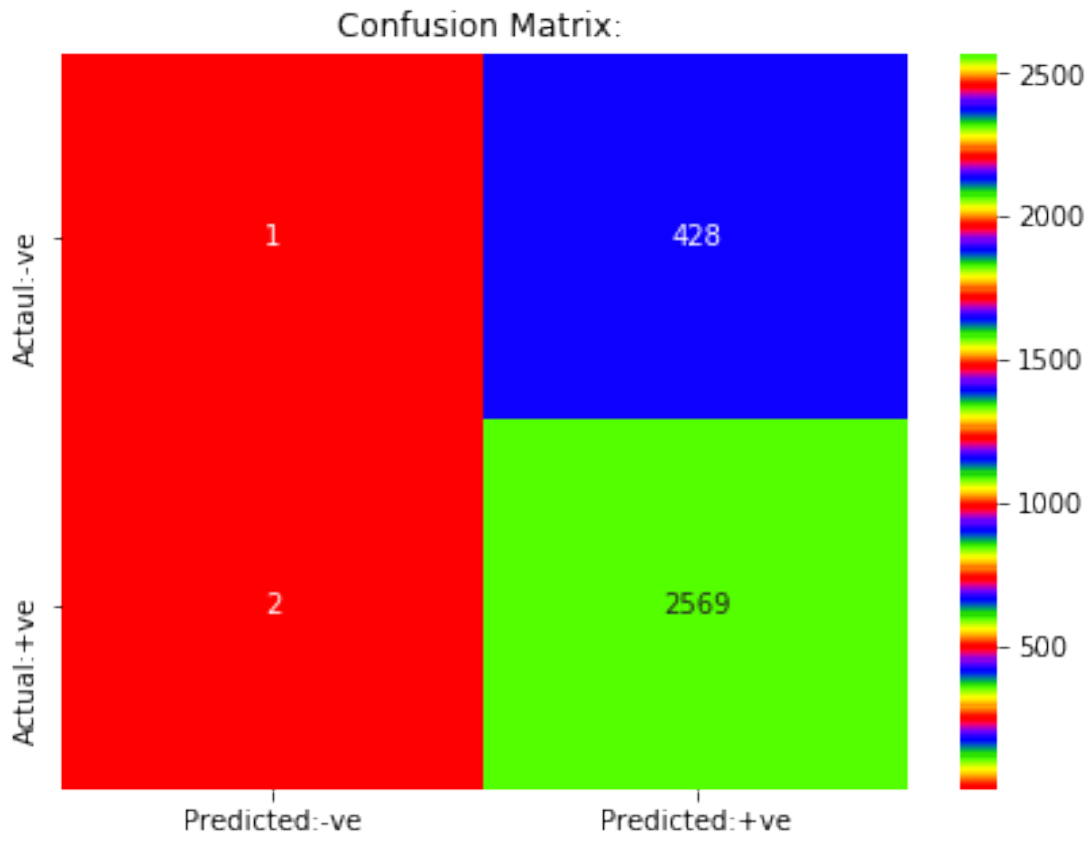
```
#create ROC curve

y_test_pred_proba = clf.predict_proba(X_test2_df)[::,1]
fpr_test, tpr_test, threshold_test = metrics.roc_curve(y_test2,
y_test_pred_proba)

y_train_pred_proba = clf.predict_proba(X_train2_df)[::,1]
fpr_train, tpr_train, threshold_train = metrics.roc_curve(y_train2,
y_train_pred_proba)

#roc_auc = auc(false_positive_rate, true_positive_rate)
plt.plot(fpr_test,tpr_test,label="test AUC ")
plt.plot(fpr_train,tpr_train,label="train AUC ")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.title("ROC-AUC curve")
plt.legend()
plt.show()
#print(threshold_test)
```

```
#print(y_pred_proba)
```




```
# 11. Plot confusion matrix based on best threshold value
y_pred = clf.predict(X_test2_df)
cm = confusion_matrix(y_test2,  y_pred)

cm_df =pd.DataFrame(cm,columns=["Predicted:-ve","Predicted:+ve"],index
= ["Actaul:-ve","Actual:+ve"])
print(cm_df)
print("***************confusion matrix***********************")

fig = plt.figure(figsize=(7,5))
ax =sns.heatmap(cm_df,annot =True, xticklabels=True,
yticklabels=True,cmap="prism",fmt="")
plt.title("Confusion Matrix: ")
plt.show()
```

```
            Predicted:-ve  Predicted:+ve
Actaul:-ve              1            428
Actual:+ve              2           2569
***************confusion matrix***********************
```



Observation (For all data sets)

- Both tasks, i.e., with TFIDF processed text and TFIDF-W2V processed text (essay) resulted in different results interms of choosing best parameters,confusion matrix, word-cloud.
- TFIDF-W2V processed text gave better AUC Score (0.7563) compared to TFIDF AUC score (0.7462)
- Response encoding converted all features to 2-encoded features there by reducing the dimension of dataset to process.
- Also, TFIDF-W2V resulted in less no of vector length(i.e., only 300) after text to vector convertion which has a edge over the other method with regards to memory and time complexity

as mentioned in the step 4 of instructions

```
#!pip install prettytable
# Tabulate your results
# Please compare all your models using Prettytable library
#Ref: https://pypi.org/project/prettytable/?
msclkid=31b07eccba8911ec909d9465bafb9d1e
from prettytable import PrettyTable
tb = PrettyTable()
```

```
tb.field_names= (" Vectorizer ", " Max_depth ", " n_estimators ","
Best -AUC ","False Positives")
tb.add_row([" Tf - Idf", 3 , 20 ,0.746,427 ])
tb.add_row([" AVG-W2V", 3, 20,0.7563,428])
print(tb.get_string(title = "Gradient Boosting Decision trees-
Observations"))
```

```
+-------------------------------------------------------------------------
--------+
|                Gradient Boosting Decision trees- Observations
|
+--------------+-------------+----------------+-------------
+-----------------+
|  Vectorizer  |  Max_depth  |  n_estimators  |  Best -AUC  | False
Positives |
+--------------+-------------+----------------+-------------
+-----------------+
|   Tf - Idf   |      3      |       20       |    0.746    |
427       |
|    AVG-W2V   |      3      |       20       |    0.7563   |
428       |
+--------------+-------------+----------------+-------------
+-----------------+
```