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

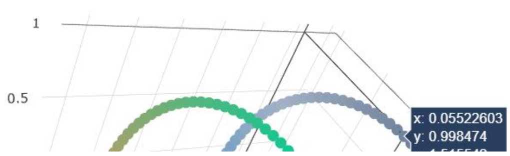
Encoded Test Data

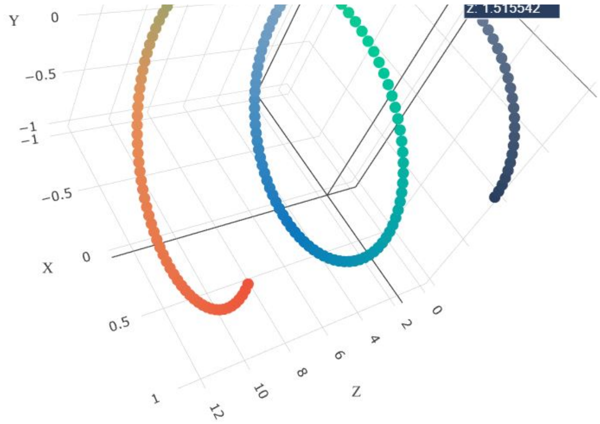
State_0	State_1
3/5	2/5
1/3	2/3
1/2	1/2
1/3	2/3
0/2	2/2
1/2	1/2

> 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. Instructions

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

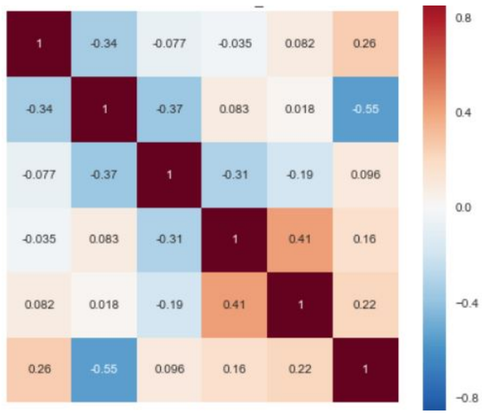




with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

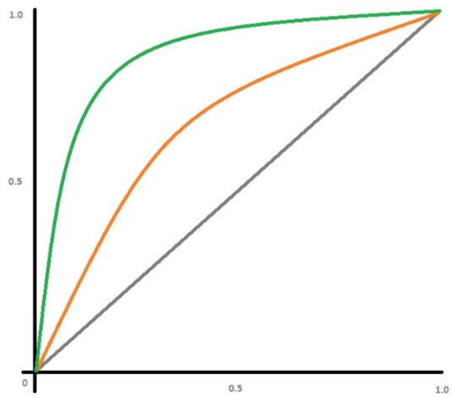
or

- 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

1. GBDT (xgboost/lightgbm)

1.1 Loading Data

In [1]:

```
!pip install chart_studio
```

```
Collecting chart_studio
  Downloading https://files.pythonhosted.org/packages/ca/ce/330794a6b6ca4b9182c38fc69dd2a9cbff60fd49421cb8648ee5fee352dc/chart_studio-1.1.0-py3-none-any.whl (64kB)
     |████████████████████████████████| 71kB 3.6MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from chart_studio) (2.23.0)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from chart_studio) (1.15.0)
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packages (from chart_studio) (1.3.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from chart_studio) (4.4.1)
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_studio) (1.24.3)
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: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests->chart_studio) (2.10)
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
```

In [2]:

```
%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.ensemble import GradientBoostingClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

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

import pickle
from tqdm import tqdm,tnrange,tqdm_notebook
import os
import plotly as ply
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
```

```
from collections import Counter
```

In [3]:

```
from sklearn.model_selection import GridSearchCV
```

In [4]:

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

```
Mounted at /content/drive
```

In [5]:

```
# preprocessed data contains title also.
path = "/content/drive/MyDrive/13_Apply GBDT on donors choose dataset/preprocessed_data.csv"
```

In [6]:

```
data  = pd.read_csv(path)
```

In [7]:

```
data.columns
```

Out[7]:

```
Index(['Unnamed: 0', 'teacher_prefix', 'school_state',
       'project_grade_category', 'clean_categories', 'clean_subcategories',
       'project_title', 'teacher_number_of_previously_posted_projects',
       'project_is_approved', 'essay', 'price'],
      dtype='object')
```

In [8]:

```
data['project_is_approved'].value_counts()
```

Out[8]:

```
1    92706
0    16542
Name: project_is_approved, dtype: int64
```

Observations:

1. Data is imbalanced
2. we need to balance the data to train a model

Downsampling the data

In [9]:

```
# reference: https://www.dezyre.com/recipes/deal-with-imbalance-classes-with-downsampling-in-python
# downsample the majority class to make data balance:
# np.where() returns the indices of elements in input array where the given condition satisfies

class_0_indices = np.where(data['project_is_approved']==0)[0]
class_1_indices = np.where(data['project_is_approved']==1)[0]
print("length of class_0_indices = "+str(len(class_0_indices)),"\nlength of class_1_indices = "+str(len
(class_1_indices)))
```

```
length of class_0_indices = 16542
length of class_1_indices = 92706
```

In [10]:

```
# downsample the data by reduce the majority data points equal to minority length
downsample_indices = np.random.choice(class_1_indices,size=len(class_0_indices),replace=False)
```

In [11]:

```
# downsample data and minority class data concatenation
c_0 = data.iloc[class_0_indices]
c_1 = data.iloc[downsample_indices]
data = pd.concat([c_0,c_1],axis=0)
```

In [12]:

```
# data is balanced
data['project_is_approved'].value_counts()
```

Out[12]:

```
1    16542
0    16542
Name: project_is_approved, dtype: int64
```

1. using upsampling we didn't loose data but here in our experiment due to limited resources it tooks more time so I am doing downsampling

In [13]:

```
# # upsample the minority class to make data balance:
# # np.where() returns the indices of elements in input array where the given condition satisfies

# class_0_indices = np.where(data['project_is_approved']==0)[0]
# class_1_indices = np.where(data['project_is_approved']==1)[0]
# print("length of class_0_indices = "+str(len(class_0_indices)),"\nlength of class_1_indices = "+str(l
en(class_1_indices)))
```

In [14]:

```
# # upsample the data by repeating the minority data points equal to majority length
# upsample_indices = np.random.choice(class_0_indices,size=len(class_1_indices),replace=True)
```

In [15]:

```
# # upsample data and majority class data concatenation
# c_0 = data.iloc[upsample_indices]
# c_1 = data.iloc[class_1_indices]
# data = pd.concat([c_0,c_1],axis=0)
```

In [16]:

```
# # data is balanced
# data['project_is_approved'].value_counts()
```

In [17]:

```
data.head(2)
```

Out[17]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	clean_categories	clean_subcategories	project_title	teacher
0	0	mrs	in	grades_prek_2	literacy_language	esl_literacy	educational support for english learners at home	
2	2	ms	az	grades_6_8	health_sports	health_wellness_teamsports	soccer equipment for awesome middle school stu...	

In [18]:

```
data = data.drop(columns=['Unnamed: 0'])
```

In [19]:

```
data.reset_index(drop=True,inplace=True)
```

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

In [20]:

```
# make data into as X and Y
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[20]:

	teacher_prefix	school_state	project_grade_category	clean_categories	clean_subcategories	project_title	teacher_number_of_previo
0	mrs	in	grades_prek_2	literacy_language	esl_literacy	educational support for english learners at home	

In [21]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

1.3 Make Data Model Ready: encoding eassay, and project_title

In [22]:

```
# essay
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=3500)
vectorizer.fit(X_train['essay'].values)                                    # fit has to happen
only on train data

# we use the fitted TfidfVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
# X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
```

```
print(X_train_essay_tfidf.shape, y_train.shape)
# print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22166, 3500) (22166,)
(10918, 3500) (10918,)
====================================================================================================
```

In [23]:

```
# project_title
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=3500)
vectorizer.fit(X_train['project_title'].values)                                    # fit has to happen only on train data

# we use the fitted TfidfVectorizer to convert the text to vector
X_train_title_tfidf = vectorizer.transform(X_train['project_title'].values)
# X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_title_tfidf = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
# print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22166, 2572) (22166,)
(10918, 2572) (10918,)
====================================================================================================
```

1.4 encoding numerical

In [24]:

```
# price
from sklearn.preprocessing import Normalizer
# https://imgur.com/ldZA1zg
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)).reshape(-1,1)
# X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1)).reshape(-1,1)
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1)).reshape(-1,1)

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
# print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22166, 1) (22166,)
(10918, 1) (10918,)
====================================================================================================
```

In [25]:

```
#teacher_number_of_previously_posted_projects

normalizer = Normalizer()
# 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['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_train_teacher_noOf_previous_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)).reshape(-1,1)
# X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1)).reshape(-1,1)
X_test_teacher_noOf_previous_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)).reshape(-1,1)

print("After vectorizations")
print(X_train_teacher_noOf_previous_norm.shape, y_train.shape)
# print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_teacher_noOf_previous_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22166, 1) (22166,)
(10918, 1) (10918,)
====================================================================================================
```

1.5 Encoding categorical features:(using Response coding)

1. school_state
2. teacher_prefix
3. project_grade_category
4. clean_categories
5. clean_subcategories

In [26]:

```
categorical_featues = ['school_state','teacher_prefix','project_grade_category','clean_categories','clean_subcategories']
```

In [27]:

```
def responceTable(feature_df,feature):
  row_list = []
  categories = feature_df[feature].unique()                    # store all unique categories in a categorical feature
  result = feature_df.groupby('class')[feature].value_counts()
  # print(result)
  keys_0=list(result[0].keys())
  # print(keys_0)
  keys_1=list(result[1].keys())
  for category in categories:
    dict1 = {}
    class_0_value = 0
    class_1_value = 0
    dict1[feature] = category
    if (category in keys_0):
      class_0_value = result[0][category]
    if (category in keys_1):
      class_1_value = result[1][category]
    dict1['class_0'] = class_0_value
    dict1['class_1'] = class_1_value
    # print(dict1)
    row_list.append(dict1)
  responce_table = pd.DataFrame(row_list)
  return responce_table
```

In [28]:

```
# function to convert categorical features into responce coding:
def responceCoding(X_train,y_train,categorical_featues):  # return responce coding of train data of all categorical features:
  N = len(np.unique(y_train))                              # each categorical feature has to be converted into 'N' dimentions where (N = distinct class labels)
```

```
  response_dfs = []
  for feature in categorical_featues:

    x_feature = X_train[feature].reset_index()
    x_feature.drop(columns=['index'],inplace=True)

    Y_train = pd.Series(y_train,name='class')

    feature_df = pd.concat([x_feature,Y_train],axis=1)

    response = responceTable(feature_df,feature)

    response_dfs.append(response)
  return response_dfs
```

In [29]:

```
allResponseTables_dfs=responceCoding(X_train,y_train,categorical_featues)
```

In [30]:

```
allResponseTables_dfs[2]
```

Out[30]:

	project_grade_category	class_0	class_1
0	grades_3_5	3614	3703
1	grades_prek_2	4489	4473
2	grades_6_8	1798	1760
3	grades_9_12	1182	1147

In [31]:

```
def encode(feature,feature_df,responce_table):
  row_list = []
  state_0 = str(feature) + '_0'
  state_1 = str(feature) + '_1'

  for each_row in feature_df[feature]:
    dict2 = {}
    value = (responce_table[feature]==each_row).any()
    if (value):
      class_0_prob = (responce_table[responce_table[feature]== each_row]['class_0'].values[0]/(responce
_table[responce_table[feature]== each_row]['class_0'].values[0]+responce_table[responce_table[feature]=
= each_row]['class_1'].values[0]))
      class_1_prob = (responce_table[responce_table[feature]== each_row]['class_1'].values[0]/(responce
_table[responce_table[feature]== each_row]['class_0'].values[0]+responce_table[responce_table[feature]=
= each_row]['class_1'].values[0]))

      dict2[state_0]= class_0_prob
      dict2[state_1]= class_1_prob
    else:
      dict2[state_0]= 1/2
      dict2[state_1]= 1/2

    row_list.append(dict2)
  encoded_df = pd.DataFrame(row_list)
  return encoded_df
```

In [93]:

```
# Encoding Train data:
def encoding(X,y,categorical_featues,allResponseTables_dfs):
  encoded_dfs = []
  for index,feature in tqdm_notebook(enumerate(categorical_featues)):
    response_table = allResponseTables_dfs[index]
    feature_df = 0
```

```
    x_feature = X[feature].reset_index()
    x_feature.drop(columns=['index'],inplace=True)
    Y       = pd.Series(y,name='class')
    feature_df = pd.concat([x_feature,Y],axis=1)
    # pass

    featureEncoded_DF = encode(feature,feature_df,response_table)
    encoded_dfs.append(featureEncoded_DF)

  Encoded_DF = pd.DataFrame()
  for each_featureEncoded_DF in encoded_dfs:
    Encoded_DF = pd.concat([Encoded_DF,each_featureEncoded_DF],axis=1)

  return Encoded_DF
```

In [94]:

```
response_df = encoding(X_train,y_train,categorical_featues,allResponseTables_dfs)
```

In [82]:

```
response_df
```

Out[82]:

	school_state_0	school_state_1	teacher_prefix_0	teacher_prefix_1	project_grade_category_0	project_grade_category_1	clean_categories_0	c
0	0.568987	0.431013	0.510186	0.489814	0.493918	0.506082	0.546422	
1	0.517107	0.482893	0.486534	0.513466	0.493918	0.506082	0.546422	
2	0.516026	0.483974	0.510186	0.489814	0.500893	0.499107	0.452254	
3	0.517107	0.482893	0.510186	0.489814	0.500893	0.499107	0.566871	
4	0.473701	0.526299	0.486534	0.513466	0.505340	0.494660	0.546422	
...	...	...	...	...	...	...	...	
22161	0.473701	0.526299	0.510186	0.489814	0.493918	0.506082	0.467245	
22162	0.517107	0.482893	0.510186	0.489814	0.500893	0.499107	0.467245	
22163	0.471033	0.528967	0.486534	0.513466	0.500893	0.499107	0.554846	
22164	0.478814	0.521186	0.486534	0.513466	0.505340	0.494660	0.506443	
22165	0.523684	0.476316	0.486534	0.513466	0.507514	0.492486	0.467245	

22166 rows × 10 columns

In [34]:

```
response_df_test = encoding(X_test,y_test,categorical_featues,allResponseTables_dfs)
```

1.6 Encoding Text Features: (using Sentiment Score)

In [36]:

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

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

In [37]:

```
# Polarity function to get the sentiment polarity for give sentence.
def Polarity(preprocessed_essays):
  polarity= []                                        # list to store polarity for all sentences.
  sid = SentimentIntensityAnalyzer()
  for sentence in tqdm_notebook(preprocessed_essays):
    scores = sid.polarity_scores(sentence)             # having polarity scores
    l = []                                             # list to store polarity scores for each sente
nce.
    for pol in scores:
      l.append(scores[pol])
    polarity.append(l)
  polarity = np.array(polarity)
  return polarity
```

In [38]:

```
# train and test polarity
train_polarity = Polarity(X_train['essay'])
test_polarity = Polarity(X_test['essay'])
```

1.7 Encoding Text Features : (using TFIDF W2V)

In [39]:

```
# glove vector file
with open('/content/drive/My Drive/13_Apply GBDT on donors choose dataset/glove_vectors','rb') as f:
  model =  pickle.load(f)
  glove_words = set(model.keys())
```

In [40]:

```
tfidf_model = TfidfVectorizer()
# fit using trian essay
tfidf_model.fit(X_train['essay'])
# converting word as key and idf values as value.
idf_dict = dict(zip(tfidf_model.get_feature_names(),list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [41]:

```
# function to compute tfidf_w2v
def tfidfW2v(essays):
  tfidf_w2v_vectors = []
  for sentence in tqdm_notebook(essays):
    vector = np.zeros(300)
    tf_idf_weight = 0
    for word in sentence.split():
      # check word is present in both glove_words and tfidf_words
      if (word in glove_words) and (word in tfidf_words):
        w2v = model[word]
        tf_idf = idf_dict[word]*(sentence.count(word)/len(sentence.split()))
        vector += (w2v * tf_idf)
        tf_idf_weight += tf_idf
    if tf_idf_weight!=0:
      vector/=tf_idf_weight
    tfidf_w2v_vectors.append(vector)
  return tfidf_w2v_vectors
```

In [42]:

```
# computing tfidf_w2v for train and test
tfidfW2v_train = tfidfW2v(X_train['essay'])
tfidfW2v_test = tfidfW2v(X_test['essay'])
```

In [43]:

```
tfidf_model = TfidfVectorizer()
# fit using trian project_title
tfidf_model.fit(X_train['project_title'])
# converting word as key and idf values as value.
idf_dict = dict(zip(tfidf_model.get_feature_names(),list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [44]:

```
# computing tfidf_w2v for train and test
tfidfW2v_title_train = tfidfW2v(X_train['project_title'])
tfidfW2v_title_test = tfidfW2v(X_test['project_title'])
```

1.8 Concatinating all features:

1. set 1
2. set 2

In [103]:

```
# set-1
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_tfidf = hstack((response_df,X_train_teacher_noOf_previous_norm,X_train_price_norm,X_train_essay_tf
idf,X_train_title_tfidf,
               train_polarity)).tocsr()
X_te_tfidf = hstack((response_df_test,X_test_teacher_noOf_previous_norm,X_test_price_norm,X_test_essay_
tfidf,X_test_title_tfidf,
               test_polarity)).tocsr()

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

```
Final Data matrix
(22166, 6088) (22166,)
(10918, 6088) (10918,)
====================================================================================================
```

In [46]:

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

In [47]:

```
# converting essay data into compressed sparse matrix
tfidfW2v_train = csr_matrix(tfidfW2v_train)
tfidfW2v_test  = csr_matrix(tfidfW2v_test)
```

In [48]:

```
tfidfW2v_title_train = csr_matrix(tfidfW2v_title_train)
tfidfW2v_title_test  = csr_matrix(tfidfW2v_title_test)
```

In [102]:

```
# set-2
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_tfidf_w2v = hstack((response_df,X_train_teacher_noOf_previous_norm,X_train_price_norm,tfidfW2v_tra
in,tfidfW2v_title_train)).tocsr()
X_te_tfidf_w2v = hstack((response_df_test,X_test_teacher_noOf_previous_norm,X_test_price_norm,tfidfW2v_
test,tfidfW2v_title_test)).tocsr()

print("Final Data matrix")
print(X_tr_tfidf_w2v.shape, y_train.shape)
print(X_te_tfidf_w2v.shape, y_test.shape )
print("="*100)
```

```
Final Data matrix
(22166, 612) (22166,)
(10918, 612) (10918,)
====================================================================================================
```

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

Model on set-1

In [ ]:

```
parameters = {'max_depth':[5,10,15,20],'n_estimators':[25,50,75,100]}
```

Grid search (Gbdt on set-1)

In [ ]:

```
# initializing decision tree classifier
gbdt = GradientBoostingClassifier(random_state=39)
# using GridSearch with given parameters and "roc_auc" as a metric - 3 fold cross validation.
gbdt_grid = GridSearchCV(gbdt, param_grid=parameters,scoring='roc_auc',n_jobs=-1,
                                cv=3,return_train_score=True)
```

In [ ]:

```
gbdt_grid.fit(X_tr_tfidf,y_train)
results = pd.DataFrame.from_dict(gbdt_grid.cv_results_)
results.head(5)
```

Out[ ]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_estimators	params	split0_test_s
0	96.083489	0.623540	0.034178	0.001308	5	25	{'max_depth': 5, 'n_estimators': 25}	0.69
							{'max_depth':	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_estimators	params	split0_test_s
1	185.121953	0.616929	0.049600	0.001041	5	50	{'max_depth': 5, 'n_estimators': 50}	0.71
2	273.378243	2.490340	0.066682	0.003464	5	75	{'max_depth': 5, 'n_estimators': 75}	0.71
3	359.497307	2.300644	0.084657	0.001641	5	100	{'max_depth': 5, 'n_estimators': 100}	0.72
4	264.569740	6.447698	0.052452	0.001915	10	25	{'max_depth': 10, 'n_estimators': 25}	0.69

In [ ]:

```
results = results.sort_values(['param_n_estimators','param_max_depth'])
```

In [ ]:

```
x_n_estimators = results['param_n_estimators']
y_max_depth    = results['param_max_depth']
z_train_auc    = results['mean_train_score']
z_cv_auc       = results['mean_test_score']
```

Heat Maps:

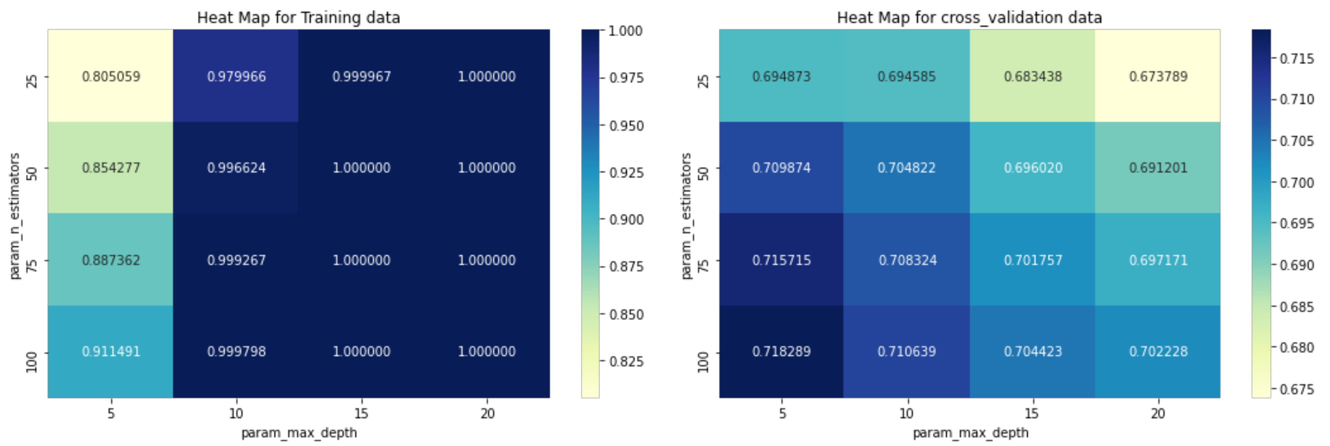
In [ ]:

```
# resource: https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
# pivot_table is used to create a spreadsheet-style table as a DataFrame.

df1 = pd.concat([x_n_estimators,y_max_depth,z_train_auc],axis=1).reset_index()
df1.drop(columns=['index'],inplace=True)
df2 = pd.pivot_table(df1,values='mean_train_score',index='param_n_estimators',columns='param_max_depth'
)

df3 = pd.concat([x_n_estimators,y_max_depth,z_cv_auc],axis=1).reset_index()
df3.drop(columns=['index'],inplace=True)
df4 = pd.pivot_table(df3,values='mean_test_score',index='param_n_estimators',columns='param_max_depth')


fig,ax =plt.subplots(1,2,figsize=(15,5))
sns.heatmap(df2,cmap='YlGnBu',annot=True,fmt='f',ax=ax[0]).set_title('Heat Map for Training data')
sns.heatmap(df4,cmap='YlGnBu',annot=True,fmt='f',ax=ax[1]).set_title('Heat Map for cross_validation dat
a')
plt.tight_layout()
```



In [ ]:

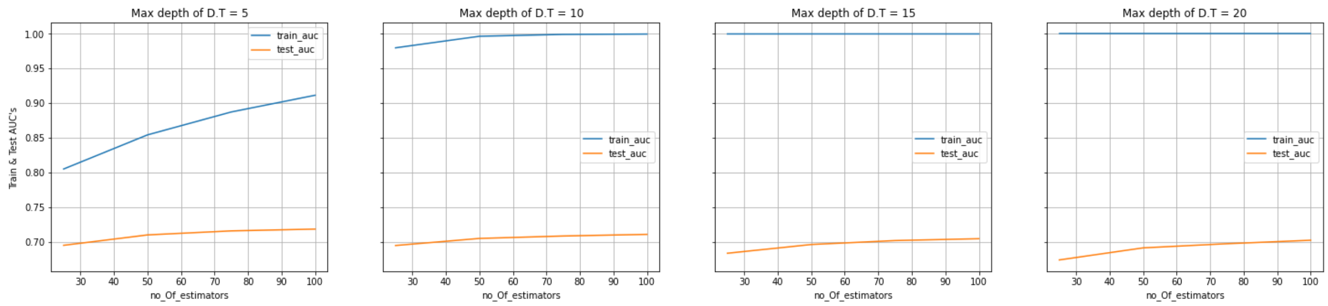
In [ ]:

```
# finding best hyper-parameters using cross validation (if grid search is used):
# results:
x_n_estimators = results['param_n_estimators']
y_max_depth    = results['param_max_depth']
z_train_auc    = results['mean_train_score']
z_cv_auc       = results['mean_test_score']

# sorted unique depths and estimators
depths = np.sort(y_max_depth.unique())
estimators = np.sort(x_n_estimators.unique())

# ploting subplots to get an clear idea to select best hpyer-parameters.
fig,ax = plt.subplots(1,4,figsize=(25,5),sharey=True,sharex=True)
for i,depth in enumerate(depths):
  train_auc = []
  test_auc  = []
  for estimator in estimators:
    train_auc.append(results[(results['param_n_estimators']==estimator) & (results['param_max_depth']==
depth)]['mean_train_score'])
    test_auc.append(results[(results['param_n_estimators']==estimator) & (results['param_max_depth']==d
epth)]['mean_test_score'])

  ax[i].plot(estimators,train_auc,label="train_auc")
  ax[i].plot(estimators,test_auc,label="test_auc")
  ax[i].set_xlabel('no_Of_estimators')
  #ax[i].set_ylabel("Train & Test AUC's ")
  ax[i].set_title("Max depth of D.T = "+ str(depth))
  ax[i].grid()
  ax[i].legend()
fig.text(0.1,0.5,"Train & Test AUC's",va='center',rotation='vertical')
plt.show()
```



Observations:

```
from above subplots, we can see that for the best Hyperparameters are
     max_depth          = 5
     n_estimators       = 100
```

In [105]:

```
# training decison Tree classifier with best hyperparameters:
# https://stackoverflow.com/questions/37522191/how-to-balance-classification-using-decisiontreeclassifi
er/37522252#37522252

clf = GradientBoostingClassifier(max_depth=5,n_estimators=100,random_state=42)
clf.fit(X_tr_tfidf,y_train)
```

Out[105]:

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=5,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=42, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
```

```
                                    warm_start=False)
```

In [106]:

```
# classes present order
print(clf.classes_)
```

```
[0 1]
```

In [107]:

```
# predicted probability scores of test data
proba      = clf.predict_proba(X_tr_tfidf)
# proba contains both classes probabilities, hence we need to pick class-1 proba scores
prob_train = proba[:,1]

# predicted probability scores of train data
proba      = clf.predict_proba(X_te_tfidf)
prob_test = proba[:,1]
```

In [108]:

```
#  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score
from sklearn.metrics import roc_auc_score
auc_train = roc_auc_score(y_train,prob_train)
auc_test = roc_auc_score(y_test,prob_test)
auc_test_model1 = auc_test
print(" Train auc = " + str(auc_train),'\n',"Test auc = "+ str(auc_test))
```

```
 Train auc = 0.8683562795628782
 Test auc = 0.7090304077279308
```

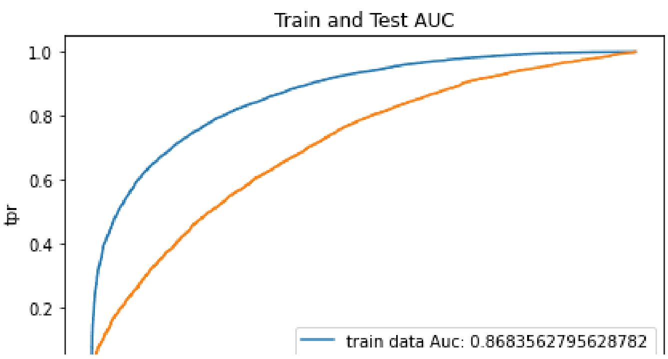
In [110]:

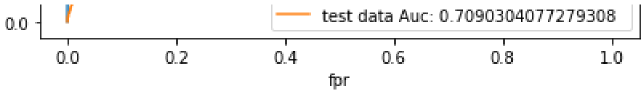
```
#  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
from sklearn import metrics
fpr_tr, tpr_tr, thresholds_tr = metrics.roc_curve(y_train,prob_train, pos_label=1)
fpr_te, tpr_te, thresholds_te = metrics.roc_curve(y_test,prob_test, pos_label=1)
```

AUC -Model-1

In [111]:

```
plt.plot(fpr_tr,tpr_tr,label="train data Auc: "+str(auc_train))
plt.plot(fpr_te,tpr_te,label="test data Auc: "+str(auc_test))
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("Train and Test AUC")
plt.legend()
plt.tight_layout()
plt.show()
```





Confusion matrix:

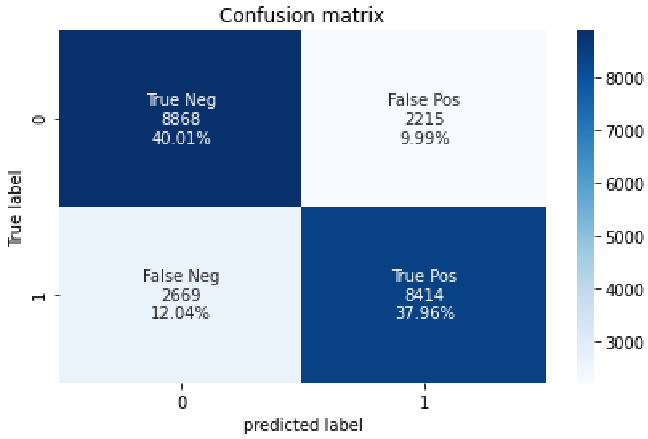
Confusion matrix on train data

In [112]:

```
# predicted
y_predicted = clf.predict(X_tr_tfidf)
mat = confusion_matrix(y_train,y_predicted)
```

In [113]:

```
#  https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                mat.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     mat.flatten()/np.sum(mat)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(mat, annot=labels, fmt='', cmap='Blues').set_title("Confusion matrix")
plt.ylabel("True label")
plt.xlabel("predicted label")
plt.tight_layout()
```



confusion matrix on test data

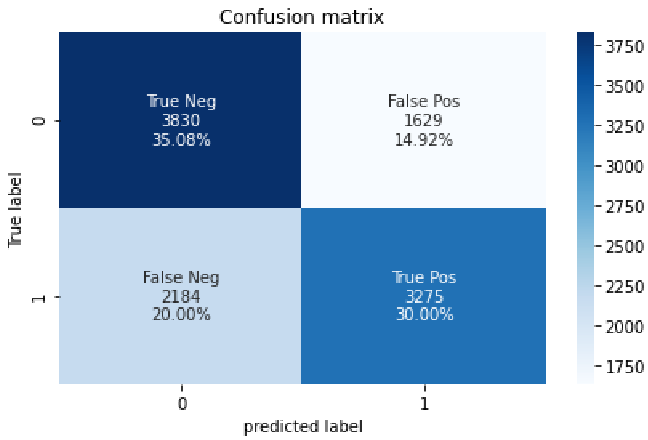
In [114]:

```
# predicted
y_predicted = clf.predict(X_te_tfidf)
mat = confusion_matrix(y_test,y_predicted)
```

In [115]:

```
#  https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                mat.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     mat.flatten()/np.sum(mat)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
```

```
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(mat, annot=labels, fmt='', cmap='Blues').set_title("Confusion matrix")
plt.ylabel("True label")
plt.xlabel("predicted label")
plt.tight_layout()
```



Model on set-2

In [96]:

```
parameters = {'max_depth':[5,10,15],'n_estimators':[25,50,70]}
```

Grid search (Gbdt on set-2)

In [97]:

```
# initializing decision tree classifier
gbdt = GradientBoostingClassifier(random_state=39)
# using GridSearch with given parameters and "roc_auc" as a metric - 3 fold cross validation.
gbdt_grid = GridSearchCV(gbdt, param_grid=parameters,scoring='roc_auc',n_jobs=-1,
                         cv=3,return_train_score=True)
```

In [98]:

```
gbdt_grid.fit(X_tr_tfidf_w2v,y_train)
results = pd.DataFrame.from_dict(gbdt_grid.cv_results_)
results.head(5)
```

Out[98]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_estimators	params	split0_test_s
0	236.550965	0.071208	0.063501	0.005809	5	25	{'max_depth': 5, 'n_estimators': 25}	0.68
1	469.149325	0.476762	0.074977	0.000480	5	50	{'max_depth': 5, 'n_estimators': 50}	0.69
2	657.624416	1.571451	0.093636	0.008709	5	70	{'max_depth': 5, 'n_estimators': 70}	0.70
3	712.868214	2.496825	0.087618	0.002768	10	25	{'max_depth': 10, 'n_estimators': 25}	0.66
4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	{'max_depth': 10,	[illegible]

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_n_estimators	params	split0_test_s
4	1413.128979	9.832241	0.141746	0.005985	10	50	{'n_estimators': 50}	0.67

In [99]:

```
results = results.sort_values(['param_n_estimators','param_max_depth'])
```

In [100]:

```
x_n_estimators = results['param_n_estimators']
y_max_depth   = results['param_max_depth']
z_train_auc   = results['mean_train_score']
z_cv_auc      = results['mean_test_score']
```

Heat Maps:

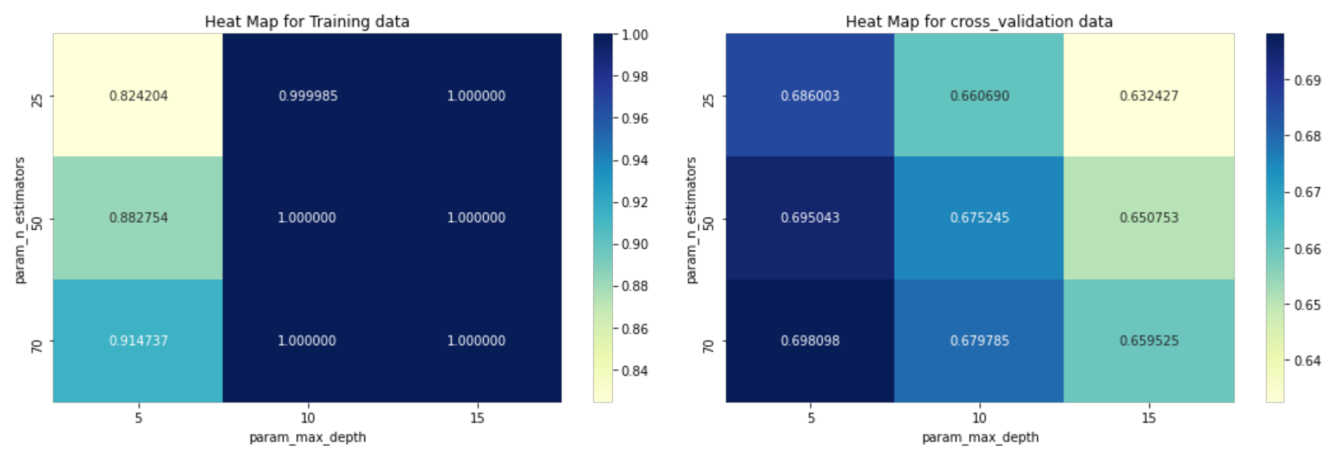
In [101]:

```
# resource: https://cmdlinetips.com/2019/01/how-to-make-heatmap-with-seaborn-in-python/
# pivot_table is used to create a spreadsheet-style table as a DataFrame.

df1 = pd.concat([x_n_estimators,y_max_depth,z_train_auc],axis=1).reset_index()
df1.drop(columns=['index'],inplace=True)
df2 = pd.pivot_table(df1,values='mean_train_score',index='param_n_estimators',columns='param_max_depth'
)

df3 = pd.concat([x_n_estimators,y_max_depth,z_cv_auc],axis=1).reset_index()
df3.drop(columns=['index'],inplace=True)
df4 = pd.pivot_table(df3,values='mean_test_score',index='param_n_estimators',columns='param_max_depth')


fig,ax =plt.subplots(1,2,figsize=(15,5))
sns.heatmap(df2,cmap='YlGnBu',annot=True,fmt='f',ax=ax[0]).set_title('Heat Map for Training data')
sns.heatmap(df4,cmap='YlGnBu',annot=True,fmt='f',ax=ax[1]).set_title('Heat Map for cross_validation dat
a')
plt.tight_layout()
```



In [104]:

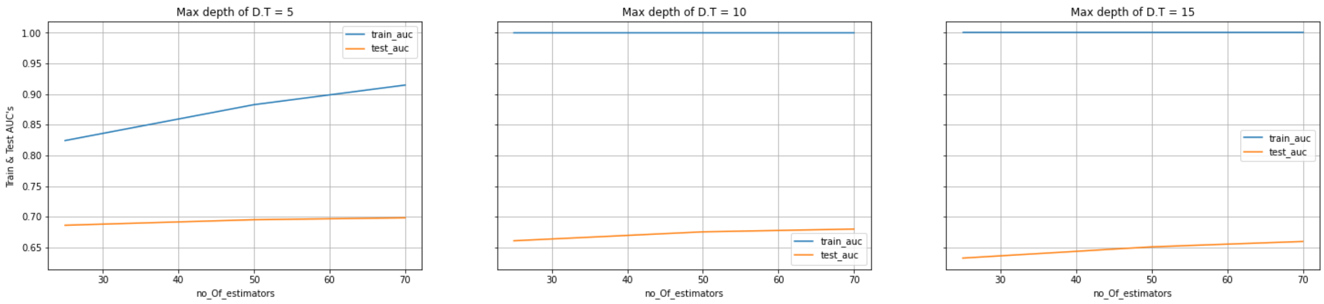
```
# finding best hyper-parameters using cross validation (if grid search is used):
# results:
x_n_estimators = results['param_n_estimators']
y_max_depth   = results['param_max_depth']
z_train_auc   = results['mean_train_score']
z_cv_auc      = results['mean_test_score']

# sorted unique depths and estimators
depths = np.sort(y_max_depth.unique())
estimators = np.sort(x_n_estimators.unique())

# ploting subplots to get an clear idea to select best hpyer-parameters.
fig, ax = plt.subplots(1,3,figsize=(25,5),sharey=True,sharex=True)
```

```
fig,ax = plt.subplots(1,3,figsize=(25,5),sharey=True,sharex=True)
for i,depth in enumerate(depths):
  train_auc = []
  test_auc  = []
  for estimator in estimators:
    train_auc.append(results[(results['param_n_estimators']==estimator) & (results['param_max_depth']==
depth)]['mean_train_score'])
    test_auc.append(results[(results['param_n_estimators']==estimator) & (results['param_max_depth']==d
epth)]['mean_test_score'])

  ax[i].plot(estimators,train_auc,label="train_auc")
  ax[i].plot(estimators,test_auc,label="test_auc")
  ax[i].set_xlabel('no_Of_estimators')
  #ax[i].set_ylabel("Train & Test AUC's ")
  ax[i].set_title("Max depth of D.T = "+ str(depth))
  ax[i].grid()
  ax[i].legend()
fig.text(0.1,0.5,"Train & Test AUC's",va='center',rotation='vertical')
plt.show()
```



Observations:

```
    from above subplots, we can see that for the best Hyperparameters are
        max_depth          = 5
        n_estimators       = 70 here
```

In [125]:

```
# training decison Tree classifier with best hyperparameters:
# https://stackoverflow.com/questions/37522191/how-to-balance-classification-using-decisiontreeclassifi
er/37522252#37522252

clf = GradientBoostingClassifier(max_depth=5,n_estimators=100,random_state=42)
clf.fit(X_tr_tfidf_w2v,y_train)
```

Out[125]:

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=5,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=42, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [126]:

```
# classes present order
print(clf.classes_)
```

```
[0 1]
```

In [127]:

```
# predicted probability scores of test data
proba     = clf.predict_proba(X_tr_tfidf_w2v)
# proba contains both classes probabilities, hence we need to pick class-1 proba scores
prob_train = proba[:,1]

# predicted probability scores of train data
proba     = clf.predict_proba(X_te_tfidf_w2v)
prob_test = proba[:,1]
```

In [128]:

```
#  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score
from sklearn.metrics import roc_auc_score
auc_train = roc_auc_score(y_train,prob_train)
auc_test = roc_auc_score(y_test,prob_test)
auc_test_model2 = auc_test
print(" Train auc = " + str(auc_train),'\n',"Test auc = "+ str(auc_test))
```

```
 Train auc = 0.9047053472787733
 Test auc = 0.6863803213087648
```

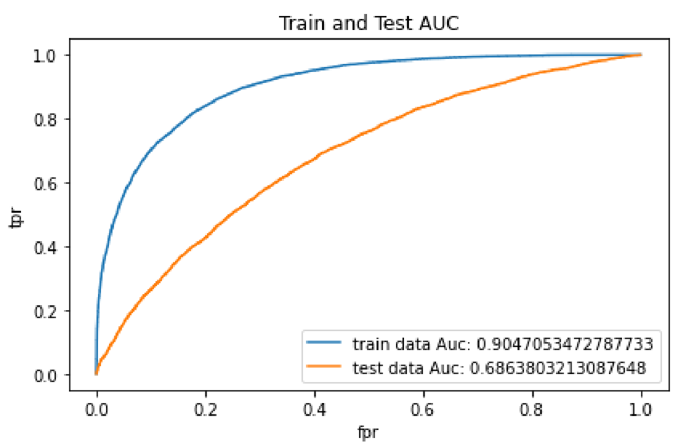
In [130]:

```
#  https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
from sklearn import metrics
fpr_tr, tpr_tr, thresholds_tr = metrics.roc_curve(y_train,prob_train, pos_label=1)
fpr_te, tpr_te, thresholds_te = metrics.roc_curve(y_test,prob_test, pos_label=1)
```

AUC -Model-2

In [131]:

```
plt.plot(fpr_tr,tpr_tr,label="train data Auc: "+str(auc_train))
plt.plot(fpr_te,tpr_te,label="test data Auc: "+str(auc_test))
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("Train and Test AUC")
plt.legend()
plt.tight_layout()
plt.show()
```



Confusion matrix:

confusion matrix on train data

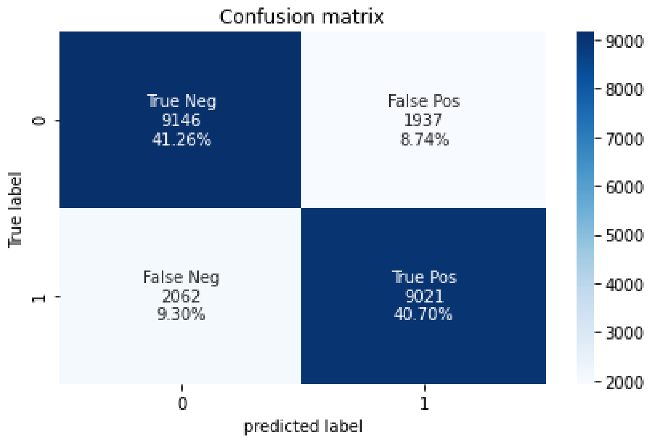
In [132]:

In [132]:

```python
# predicted
y_predicted = clf.predict(X_tr_tfidf_w2v)
mat = confusion_matrix(y_train,y_predicted)
```

In [133]:

```python
#  https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                mat.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     mat.flatten()/np.sum(mat)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(mat, annot=labels, fmt='', cmap='Blues').set_title("Confusion matrix")
plt.ylabel("True label")
plt.xlabel("predicted label")
plt.tight_layout()
```



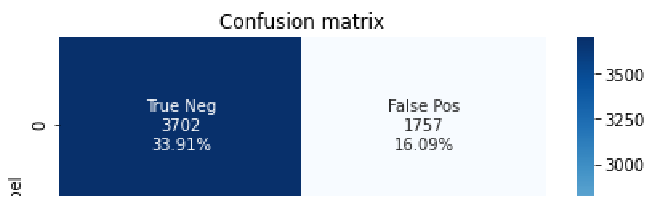
confusion matrix on test data

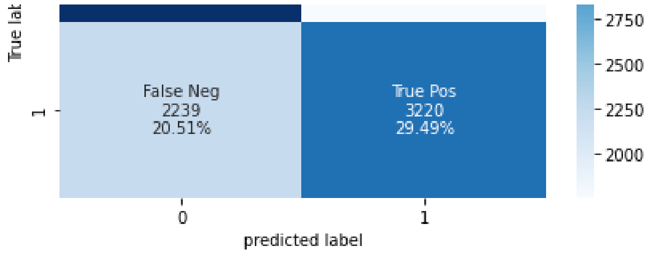
In [134]:

```python
# predicted
y_predicted = clf.predict(X_te_tfidf_w2v)
mat = confusion_matrix(y_test,y_predicted)
```

In [135]:

```python
#  https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                mat.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     mat.flatten()/np.sum(mat)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(mat, annot=labels, fmt='', cmap='Blues').set_title("Confusion matrix")
plt.ylabel("True label")
plt.xlabel("predicted label")
plt.tight_layout()
```





3. Summary

as mentioned in the step 4 of instructions

In [136]:

```
from tabulate import tabulate
table = [["TFIDF","GBDT",100,5,auc_test_model1],["TFIDF_W2v","GBDT",100,5,auc_test_model2]]
headers = ["Vectorizer","Model","No.of estimators (Hyper parameter)","Depth (Hyper parameter)","AUC"]
print(tabulate(table,headers,tablefmt="grid"))
```

```
+--------------+---------+--------------------------------------+---------------------------+---------+
| Vectorizer   | Model   |   No.of estimators (Hyper parameter) |   Depth (Hyper parameter) |     AUC |
+==============+=========+======================================+===========================+=========+
| TFIDF        | GBDT    |                                  100 |                         5 | 0.70903 |
+--------------+---------+--------------------------------------+---------------------------+---------+
| TFIDF_W2v    | GBDT    |                                  100 |                         5 | 0.68638 |
+--------------+---------+--------------------------------------+---------------------------+---------+
```

1. due to limited resources I did downsampling (which contains approximately 32k points)
2. auc can be improved by increasing estimators.

In [ ]: