Assignment_DT_Instructions_final

November 21, 2020

1 Assignment : DT

```
[1]: from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

```
[ ]: import pickle

     !cp '/content/drive/MyDrive/6_Donors_choose_NB/preprocessed_data.csv' '/content/
      ↪sample_data'

     glove_vectors_path = '/content/drive/My Drive/6_Donors_choose_NB/glove_vectors'
     preprocessed_data = '/content/sample_data/preprocessed_data.csv'
```

```
[1]: import nltk
     nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\Baksv\AppData\Roaming\nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
[1]: True
```

Task - 1

1. Decision Tree

```
[2]: from sklearn.metrics import confusion_matrix
     from sklearn.metrics import roc_curve, auc
     from sklearn.metrics import roc_auc_score

     from sklearn.ensemble import RandomForestClassifier
     import pandas as pd
     import numpy as np

     pd.set_option('display.width', 10)
     pd.set_option('display.max_colwidth', 10)
```

```
import warnings
warnings.filterwarnings("ignore")
```

1.1 1.1 Loading Data

[3]:

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

data.info() #basic info about dataset : To know how many categorical and␣
 ↪numeric data point
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 9 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   school_state                                  50000 non-null  object
 1   teacher_prefix                                50000 non-null  object
 2   project_grade_category                        50000 non-null  object
 3   teacher_number_of_previously_posted_projects  50000 non-null  int64
 4   project_is_approved                           50000 non-null  int64
 5   clean_categories                              50000 non-null  object
 6   clean_subcategories                           50000 non-null  object
 7   essay                                         50000 non-null  object
 8   price                                         50000 non-null  float64
dtypes: float64(1), int64(2), object(6)
memory usage: 3.4+ MB
```

[ ]:

```
category =[]
numeric = []

for i in data.columns:

    if data[str(i)].dtype == 'int64' or data[str(i)].dtype == 'float64':
        numeric.append(i)
    else:
        category.append(i)

print(category, "\n", numeric)
```

```
['school_state', 'teacher_prefix', 'project_grade_category', 'clean_categories',
'clean_subcategories', 'essay']
 ['teacher_number_of_previously_posted_projects', 'project_is_approved',
'price']
```

1.2 1.2 Text_features Encoding

[ ]:

```
preprocessed_essays = data.essay.values
```

[4]:

```
############### TFIDF - Vectorizer ######################

from sklearn.feature_extraction.text import TfidfVectorizer

def tfidf():
    tfidf = TfidfVectorizer(min_df=10, max_features=5000, ngram_range=(1,2))
    text_tfidf = tfidf.fit_transform(preprocessed_essays)
    return text_tfidf.toarray()
```

[ ]:

```
################# TFIDF - W2V ###########################
import tqdm

with open(glove_vectors_path, 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())

def tfidf_w2v():
    tfidf = TfidfVectorizer(min_df=10, max_features=5000, ngram_range=(1,2))
    text_tfidf = tfidf.fit_transform(preprocessed_essays)

    dictionary = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
    tfidf_words = set(tfidf.get_feature_names())

    tfidf_w2v_vectors = []

    for sentence in preprocessed_essays:
        vector = np.zeros(300)
        tfidf_weight = 0

        for word in sentence.split():

            if (word in glove_words) and (word in tfidf_words):
                vec = model[word]
                tfidf_vec = dictionary[word] * sentence.count(word) /␣
 ↪len(sentence.split())
                vector += vec * tfidf_vec
                tfidf_weight += tfidf_vec

        if tfidf_weight != 0:
            vector /= tfidf_vec
        tfidf_w2v_vectors.append(vector)

    return np.array(tfidf_w2v_vectors)
```

1.3 1.3 Categorical Feature Encoding

[5]:

```python
from sklearn.preprocessing import Normalizer
from sklearn.feature_extraction.text import CountVectorizer

def ohe_vector(feature, dataset, df_name):
    try:
        if df_name == 'train':
            column = CountVectorizer()
            return column.fit_transform(dataset[feature].values)
            print(column.get_feature_names())
            print("="*100)
        else:
            return column.transform(dataset[feature].values)

    except Exception as e:
        print(e, '\n')
        print("First you should fit with train data")

def normalized(dataset, feature, df_name):
    if df_name == 'train':
        column = Normalizer()
        return column.fit_transform(dataset[feature].values.reshape(-1,1))
    else:
        return column.transform(dataset[feature].values.reshape(-1,1))
```

[ ]:

```python
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
import gc

y = data['project_is_approved']
```

1.4 1.4 Sentiment-Analyser

[6]:

```python
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

def sentiment_anayser(essay):
    sid = SentimentIntensityAnalyzer()
    negative = []
    positive = []
    neutral = []
    for sentence in essay:
        ss = sid.polarity_scores(sentence)
        sentmnt = list(ss.values())
        neg = sentmnt[0]
        neu = sentmnt[1]
        pos = sentmnt[2]
        negative.append(neg)
        neutral.append(neu)
        positive.append(pos)
    return np.column_stack((np.array(negative), np.array(neutral), np.
 ↪array(positive)))
```

1.5 1.5 HyperParameter Tuning

[ ]:

```
from sklearn.model_selection import GridSearchCV

sets = ['tfidf_w2v_vectors', 'tfidf']
scorer = dict()

for index, df_set in enumerate(sets):

    if df_set == 'tfidf_w2v_vectors':
        X = np.column_stack((ohe_vector('school_state', data, 'train').
 ↪toarray(), ohe_vector('teacher_prefix',data, 'train').toarray(), \
                                ohe_vector('project_grade_category', data,␣
 ↪'train').toarray(), ohe_vector('clean_categories', data, 'train').toarray(),␣
 ↪\
                                ohe_vector('clean_subcategories', data,␣
 ↪'train').toarray(), normalized(data, 'price', 'train'), \
                                normalized(data,␣
 ↪'teacher_number_of_previously_posted_projects', 'train'), \
                                tfidf_w2v(),␣
 ↪sentiment_anayser(preprocessed_essays)))

    elif df_set == 'tfidf':
        X = np.column_stack((ohe_vector('school_state', data, 'train').
 ↪toarray(), ohe_vector('teacher_prefix',data, 'train').toarray(), \
                                ohe_vector('project_grade_category', data,␣
 ↪'train').toarray(), ohe_vector('clean_categories', data, 'train').toarray(),␣
 ↪\
                                ohe_vector('clean_subcategories', data,␣
 ↪'train').toarray(), normalized(data, 'price', 'train'), \
                                normalized(data,␣
 ↪'teacher_number_of_previously_posted_projects', 'train'), \
                                tfidf(),␣
 ↪sentiment_anayser(preprocessed_essays)))

    print("Final Data matrix")
    print(X.shape, y.shape)

    parameters={'max_depth' : [1, 5, 10, 50], 'min_samples_split' : [5, 10,␣
 ↪100, 500]}
```

```
    gsc=GridSearchCV(estimator=RandomForestClassifier(random_state=2),
                     param_grid=parameters, scoring='roc_auc', verbose=1,␣
 ↪n_jobs=2, return_train_score=True)

    grid_result = gsc.fit(X, y)
    scorer[df_set] = grid_result.cv_results_

    print("#"*50,"\n\n")
    print("\n", df_set, " : ", "\n")

    best_params=grid_result.best_params_
    print(best_params)

    print(grid_result.best_score_,"\n")
    print("#"*50,"\n\n")

    del X
    gc.collect()
```

```
Final Data matrix
(50000, 404) (50000,)
Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed: 13.8min
[Parallel(n_jobs=2)]: Done  80 out of  80 | elapsed: 42.2min finished

##################################################




 tfidf_w2v_vectors  :

{'max_depth': 10, 'min_samples_split': 500}
0.6521899306254078

##################################################



Final Data matrix
(50000, 5104) (50000,)
Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed: 10.7min
[Parallel(n_jobs=2)]: Done  80 out of  80 | elapsed: 59.2min finished

##################################################
```

```
 tfidf  :

{'max_depth': 50, 'min_samples_split': 500}
0.6684585664197429

##################################################
```

```
[ ]: scorer['tfidf']
```

```
[ ]: {'mean_fit_time': array([ 10.97032423,   8.39581981,   8.2742507 ,   8.08722453,
         30.38114738,  30.65743785,  30.60372534,  32.38107476,
         57.29278274,  59.40535808,  57.22985897,  58.10210152,
        244.64519367, 246.68771806, 240.9268292 , 220.60696707]),
 'mean_score_time': array([0.50487132, 0.37293043, 0.30415392, 0.3109004 ,
0.45341878,
        0.44930696, 0.45625358, 0.47639599, 0.60942841, 0.62102118,
        0.60846624, 0.61170864, 1.70196991, 1.62687926, 1.61287055,
        1.33207645]),
 'mean_test_score': array([0.64199643, 0.64199643, 0.64199643, 0.64199643,
0.65453638,
        0.65321196, 0.65411831, 0.65453691, 0.65760477, 0.65832489,
        0.66059473, 0.66038638, 0.65589103, 0.65968875, 0.66312773,
        0.66845857]),
 'mean_train_score': array([0.66323864, 0.66323864, 0.66323864, 0.66323864,
0.72610265,
        0.72465271, 0.71717892, 0.71048564, 0.81759857, 0.81056051,
        0.78311858, 0.76157495, 0.99899152, 0.99763385, 0.98349227,
        0.95232967]),
 'param_max_depth': masked_array(data=[1, 1, 1, 1, 5, 5, 5, 5, 10, 10, 10, 10,
50, 50, 50, 50],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[5, 10, 100, 500, 5, 10, 100, 500,
5, 10, 100, 500, 5,
                    10, 100, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 1, 'min_samples_split': 5},
  {'max_depth': 1, 'min_samples_split': 10},
  {'max_depth': 1, 'min_samples_split': 100},
  {'max_depth': 1, 'min_samples_split': 500},
  {'max_depth': 5, 'min_samples_split': 5},
  {'max_depth': 5, 'min_samples_split': 10},
  {'max_depth': 5, 'min_samples_split': 100},
  {'max_depth': 5, 'min_samples_split': 500},
  {'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 10, 'min_samples_split': 500},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 500}],
 'rank_test_score': array([13, 13, 13, 13, 10, 12, 11,  9,  7,  6,  3,  4,  8,
5,  2,  1],
       dtype=int32),
 'split0_test_score': array([0.62910563, 0.62910563, 0.62910563, 0.62910563,
0.64444345,
        0.64421938, 0.64539118, 0.6456006 , 0.64637272, 0.64840948,
        0.64998287, 0.65162958, 0.6465921 , 0.64768292, 0.64933588,
        0.65593551]),
 'split0_train_score': array([0.66047639, 0.66047639, 0.66047639, 0.66047639,
0.72340328,
        0.72243571, 0.71374384, 0.70782126, 0.81658166, 0.80724613,
        0.7802566 , 0.75983897, 0.99896089, 0.9972285 , 0.98288046,
        0.95111946]),
 'split1_test_score': array([0.64890046, 0.64890046, 0.64890046, 0.64890046,
0.65799533,
        0.65654644, 0.65685997, 0.65734499, 0.65976193, 0.66280057,
        0.66483027, 0.66475122, 0.66246476, 0.66436722, 0.66931911,
        0.67139763]),
 'split1_train_score': array([0.65988679, 0.65988679, 0.65988679, 0.65988679,
0.72627732,
        0.72488501, 0.71774918, 0.71030597, 0.81641541, 0.81177283,
        0.78260776, 0.76197705, 0.99902579, 0.99776883, 0.9835391 ,
        0.9536554 ]),
 'split2_test_score': array([0.66678081, 0.66678081, 0.66678081, 0.66678081,
0.68373823,
        0.68332006, 0.68315087, 0.68410597, 0.68795525, 0.68307175,
        0.68891916, 0.68683993, 0.68453968, 0.68808763, 0.68944683,
        0.69730666]),
 'split2_train_score': array([0.66134003, 0.66134003, 0.66134003, 0.66134003,
0.72608051,
        0.72372666, 0.71625096, 0.70836713, 0.81372632, 0.80918846,
        0.78170601, 0.75858996, 0.99879556, 0.99749383, 0.9831278 ,
        0.95285083]),
 'split3_test_score': array([0.64196595, 0.64196595, 0.64196595, 0.64196595,
0.64958213,
        0.6481799 , 0.65045661, 0.65026432, 0.65161088, 0.6583225 ,
        0.65592125, 0.65554871, 0.64853267, 0.65385414, 0.6566919 ,
        0.66120816]),
 'split3_train_score': array([0.66947506, 0.66947506, 0.66947506, 0.66947506,
0.7319721 ,
        0.73099257, 0.72326682, 0.71695013, 0.82359753, 0.81411018,
        0.78791626, 0.76721163, 0.99922964, 0.99794845, 0.98494186,
        0.95346483]),
 'split4_test_score': array([0.62322928, 0.62322928, 0.62322928, 0.62322928,
0.63692276,
        0.63379401, 0.6347329 , 0.63536864, 0.64232306, 0.63902015,
        0.64332012, 0.64316247, 0.63732596, 0.64445183, 0.65084491,
        0.65644486]),
 'split4_train_score': array([0.66501491, 0.66501491, 0.66501491, 0.66501491,
0.72278006,
        0.72122358, 0.71488377, 0.70898373, 0.81767194, 0.81048495,
        0.78310628, 0.76025714, 0.99894571, 0.99772965, 0.98297214,
        0.95055783]),
 'std_fit_time': array([ 1.28099792,  0.22897466,  0.19307562,  0.13671441,
0.7612442 ,
         0.73884154,  0.67807887,  1.44100926,  1.24661724,  1.9657247 ,
         1.33603702,  1.85521433,  5.32188227,  6.09215234,  4.68257089,
        17.97481808]),
 'std_score_time': array([0.24656726, 0.09300561, 0.01578863, 0.00798084,
0.00903228,
        0.00942616, 0.0128151 , 0.03591922, 0.0112633 , 0.0084223 ,
        0.00612249, 0.02210068, 0.11272105, 0.04267773, 0.04494722,
        0.28034285]),
 'std_test_score': array([0.01536431, 0.01536431, 0.01536431, 0.01536431,
0.01613297,
        0.01673835, 0.01622203, 0.01641798, 0.01625682, 0.01485548,
        0.01583085, 0.01493713, 0.01642703, 0.01573995, 0.01492335,
        0.01545687]),
 'std_train_score': array([0.00359215, 0.00359215, 0.00359215, 0.00359215,
0.00324985,
        0.00339939, 0.00332615, 0.00333661, 0.00326867, 0.00232134,
        0.00258709, 0.00301974, 0.00014093, 0.00024911, 0.0007591 ,
        0.00125871])}
```

```
[ ]: scorer['tfidf_w2v_vectors']
```

```
[ ]: {'mean_fit_time': array([ 10.98010373,  10.71324081,  10.80231338,  10.7934309 ,
         45.8256465 ,  45.47882285,  45.66940317,  44.84586625,
         79.33226089,  79.129322  ,  78.31976762,  71.95379138,
        126.78118787, 126.54179239, 114.27640448,  83.41435628]),
 'mean_score_time': array([0.11832638, 0.11783423, 0.11904387, 0.1184217 ,
0.24031277,
        0.22881989, 0.23971338, 0.22277851, 0.33646212, 0.34069295,
        0.33634057, 0.32126951, 0.51920495, 0.50864782, 0.45342464,
        0.32545018]),
 'mean_test_score': array([0.60924113, 0.60924113, 0.60924113, 0.60924113,
0.6380502 ,
        0.63815688, 0.63834313, 0.63834759, 0.64921207, 0.64795964,
        0.65203743, 0.65218993, 0.61835853, 0.62031751, 0.63970564,
        0.65180713]),
 'mean_train_score': array([0.62045238, 0.62045238, 0.62045238, 0.62045238,
0.70422043,
        0.7040349 , 0.70145521, 0.69669848, 0.92375603, 0.91370905,
        0.8578653 , 0.78822202, 0.99999141, 0.99992139, 0.98310338,
        0.82978782]),
 'param_max_depth': masked_array(data=[1, 1, 1, 1, 5, 5, 5, 5, 10, 10, 10, 10,
50, 50, 50, 50],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[5, 10, 100, 500, 5, 10, 100, 500,
5, 10, 100, 500, 5,
                    10, 100, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 1, 'min_samples_split': 5},
  {'max_depth': 1, 'min_samples_split': 10},
  {'max_depth': 1, 'min_samples_split': 100},
  {'max_depth': 1, 'min_samples_split': 500},
  {'max_depth': 5, 'min_samples_split': 5},
  {'max_depth': 5, 'min_samples_split': 10},
  {'max_depth': 5, 'min_samples_split': 100},
  {'max_depth': 5, 'min_samples_split': 500},
  {'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 10, 'min_samples_split': 500},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 500}],
 'rank_test_score': array([13, 13, 13, 13, 10,  9,  8,  7,  4,  5,  2,  1, 12,
11,  6,  3],
       dtype=int32),
 'split0_test_score': array([0.54138911, 0.54138911, 0.54138911, 0.54138911,
0.5799381 ,
        0.58074223, 0.58119185, 0.58011427, 0.62372502, 0.61938637,
        0.62056196, 0.6136714 , 0.60831225, 0.6045061 , 0.62226237,
        0.61553363]),
 'split0_train_score': array([0.62976068, 0.62976068, 0.62976068, 0.62976068,
0.69853764,
        0.69875044, 0.69691508, 0.69270545, 0.92567384, 0.9144611 ,
        0.85766341, 0.78445109, 0.99998793, 0.99989566, 0.98222188,
        0.82387828]),
 'split1_test_score': array([0.62817218, 0.62817218, 0.62817218, 0.62817218,
0.65954106,
        0.65991534, 0.6589981 , 0.65846444, 0.66893586, 0.66559521,
        0.67023572, 0.67389161, 0.62717811, 0.63220161, 0.65524438,
        0.6725858 ]),
 'split1_train_score': array([0.61831771, 0.61831771, 0.61831771, 0.61831771,
0.7038289 ,
        0.70389441, 0.7001094 , 0.6955131 , 0.92800778, 0.91560246,
        0.86183166, 0.79169151, 0.99999091, 0.999907  , 0.98383846,
        0.83362805]),
 'split2_test_score': array([0.63491107, 0.63491107, 0.63491107, 0.63491107,
0.65999886,
        0.66066861, 0.65987351, 0.66140901, 0.6531115 , 0.65870591,
        0.6654454 , 0.66318136, 0.62198747, 0.62261974, 0.63797458,
        0.66353624]),
 'split2_train_score': array([0.61683538, 0.61683538, 0.61683538, 0.61683538,
0.70460202,
        0.70421575, 0.70135636, 0.69738798, 0.91792038, 0.90996916,
        0.85285431, 0.78487293, 0.99999054, 0.99990033, 0.98287971,
        0.82891698]),
 'split3_test_score': array([0.62676317, 0.62676317, 0.62676317, 0.62676317,
0.64820651,
        0.64709982, 0.64840244, 0.64963932, 0.65423389, 0.65093712,
        0.65585547, 0.65848294, 0.61781107, 0.62462103, 0.64386942,
        0.65506074]),
 'split3_train_score': array([0.61912941, 0.61912941, 0.61912941, 0.61912941,
0.70781212,
        0.70684308, 0.70369872, 0.69848227, 0.92173683, 0.91253769,
        0.85815245, 0.7909733 , 0.99998843, 0.99991236, 0.9833512 ,
        0.83204798]),
 'split4_test_score': array([0.61497012, 0.61497012, 0.61497012, 0.61497012,
0.64256649,
        0.64235841, 0.64324977, 0.64211089, 0.64605407, 0.6451736 ,
        0.64808862, 0.65172234, 0.61650376, 0.61763907, 0.63917744,
        0.65231924]),
 'split4_train_score': array([0.61821874, 0.61821874, 0.61821874, 0.61821874,
0.70632148,
        0.70647085, 0.7051965 , 0.69940361, 0.92544132, 0.91597482,
        0.85882466, 0.78912127, 0.99999926, 0.99999158, 0.98322564,
        0.83046781]),
 'std_fit_time': array([0.23978535, 0.10691292, 0.12805988, 0.12547474,
0.39813797,
        0.20257136, 0.39199497, 0.52383051, 0.28281827, 0.16142475,
        0.39425746, 0.48615938, 0.78636083, 1.82079421, 1.22847371,
        4.07577088]),
 'std_score_time': array([0.00489127, 0.00671808, 0.00522524, 0.00533572,
0.01152539,
        0.00284742, 0.01612567, 0.01135085, 0.00557289, 0.01032501,
        0.02438532, 0.01114576, 0.00794598, 0.00486425, 0.01757987,
        0.04236648]),
 'std_test_score': array([0.03452845, 0.03452845, 0.03452845, 0.03452845,
0.0298133 ,
        0.02957796, 0.02926335, 0.0298975 , 0.01475815, 0.0158717 ,
        0.0175033 , 0.02056829, 0.00625495, 0.00918942, 0.01064268,
        0.01947275]),
 'std_train_score': array([4.71218271e-03, 4.71218271e-03, 4.71218271e-03,
4.71218271e-03,
        3.15935437e-03, 2.89128800e-03, 2.87973493e-03, 2.38010389e-03,
        3.54196316e-03, 2.22005347e-03, 2.89432450e-03, 3.02826640e-03,
        4.09037567e-06, 3.55558713e-05, 5.37359268e-04, 3.34661190e-03])}
```

1.6 1.6 HyperParameter Representation

[7]:

```python
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

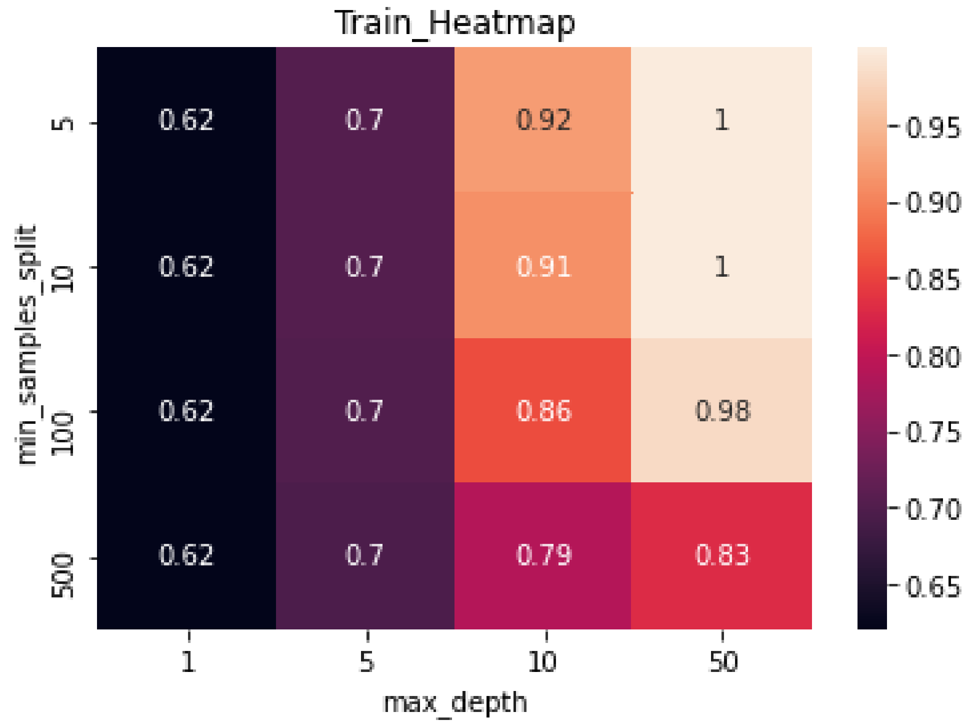
[8]:

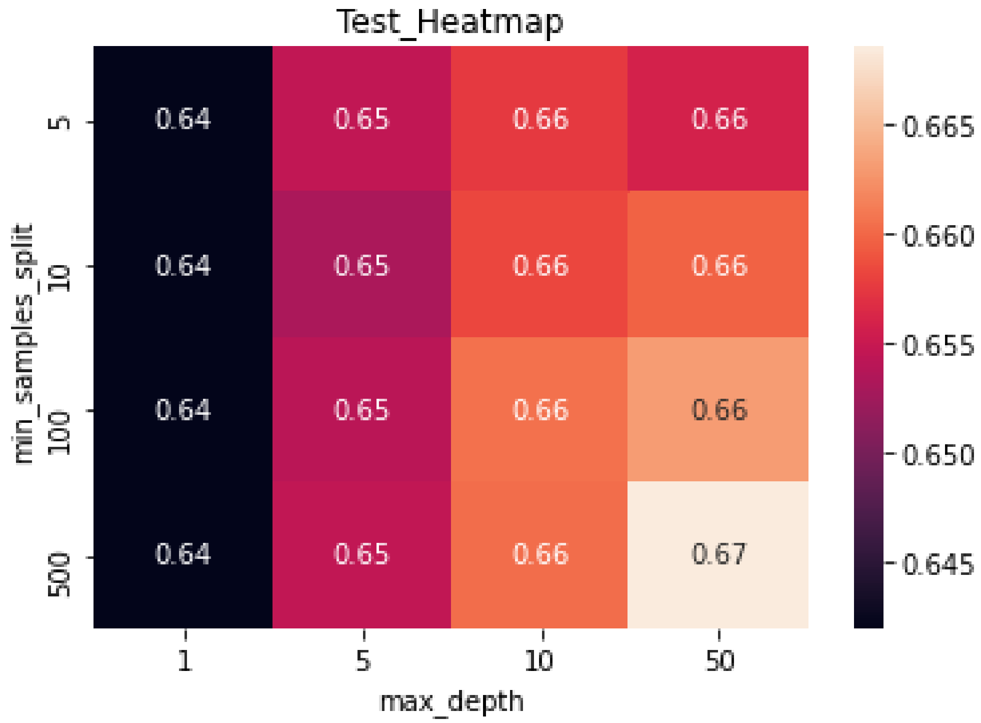
```python
params = [{'max_depth': 1, 'min_samples_split': 5},
  {'max_depth': 1, 'min_samples_split': 10},
  {'max_depth': 1, 'min_samples_split': 100},
  {'max_depth': 1, 'min_samples_split': 500},
  {'max_depth': 5, 'min_samples_split': 5},
  {'max_depth': 5, 'min_samples_split': 10},
  {'max_depth': 5, 'min_samples_split': 100},
  {'max_depth': 5, 'min_samples_split': 500},
  {'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 10, 'min_samples_split': 500},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 500}]



mean_test_score =  pd.Series([0.64199643, 0.64199643, 0.64199643, 0.64199643, 0.
 ↪65453638,0.65321196, 0.65411831,
                    0.65453691, 0.65760477, 0.65832489, 0.66059473, 0.66038638,␣
 ↪0.65589103, 0.65968875, 0.66312773,0.66845857])

mean_train_score = pd.Series([0.6204523844814621, 0.6204523844814621, 0.
 ↪6204523844814621, 0.6204523844814621, 0.704220431794736, 0.7040349049817154,
                     0.701455214619774, 0.6966984822018416, 0.9237560275150418,␣
 ↪0.9137090455849931, 0.8578652964326823, 0.7882220204638741,
                     0.9999914129193224, 0.999921388598592, 0.9831033801694795,␣
 ↪0.8297878200730657])
```

```
[9]: max_depths = []
min_samples_split = []

for parameter in params:
    max_depths.append(parameter['max_depth'])
    min_samples_split.append(parameter['min_samples_split'])
```

```
[10]: df = pd.DataFrame()
df['max_depth'] = pd.Series(max_depths)
df['min_samples_split'] = pd.Series(min_samples_split)
df['mean_test_score'] = mean_test_score
df['mean_train_score'] = mean_train_score

train_heatmap = df.pivot(index='min_samples_split', columns='max_depth',␣
 ↪values='mean_train_score')
test_heatmap = df.pivot(index='min_samples_split', columns='max_depth',␣
 ↪values='mean_test_score')
```

```
[11]: import seaborn as sns
import matplotlib.pyplot as plt

sns.heatmap(train_heatmap, annot=True)
plt.title("Train_Heatmap")
plt.show()

sns.heatmap(test_heatmap, annot=True)
plt.title("Test_Heatmap")
plt.show()
```


Train_Heatmap
min_samples_split
5
10
100
500
max_depth
1
5
10
50
0.62
0.7
0.92
1
0.62
0.7
0.91
1
0.62
0.7
0.86
0.98
0.62
0.7
0.79
0.83
0.95
0.90
0.85
0.80
0.75
0.70
0.65


Test_Heatmap
min_samples_split
5
10
100
500
max_depth
1
5
10
50
0.64
0.65
0.66
0.66
0.64
0.65
0.66
0.66
0.64
0.65
0.66
0.66
0.64
0.65
0.66
0.67
0.665
0.660
0.655
0.650
0.645

[12]:
```
# # https://plot.ly/python/3d-axes/

trace1 = go.Scatter3d(x=min_samples_split,y=max_depths,z=mean_test_score.
 ↪tolist(), name = 'cv_test_score')
trace2 = go.Scatter3d(x=min_samples_split,y=max_depths,z=mean_train_score.
 ↪tolist(), name = 'cv_train_score')

data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC')))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

[ ]:
```
data = pd.read_csv(preprocessed_data)
```

[ ]:
```
preprocessed_essays = data.essay.values

y = data['project_is_approved']
```

[ ]:
```
from sklearn.model_selection import train_test_split

data = data.drop(columns=['project_is_approved'])

X = np.column_stack((ohe_vector('school_state', data, 'train').toarray(),␣
 ↪ohe_vector('teacher_prefix',data, 'train').toarray(), \
                                ohe_vector('project_grade_category', data,␣
 ↪'train').toarray(), ohe_vector('clean_categories', data, 'train').toarray(),␣
 ↪\
                                ohe_vector('clean_subcategories', data,␣
 ↪'train').toarray(), normalized(data, 'price', 'train'), \
                                normalized(data,␣
 ↪'teacher_number_of_previously_posted_projects', 'train'), \
                                tfidf(),␣
 ↪sentiment_anayser(preprocessed_essays)))
```

[ ]:
```
print(X.shape, y.shape)
```

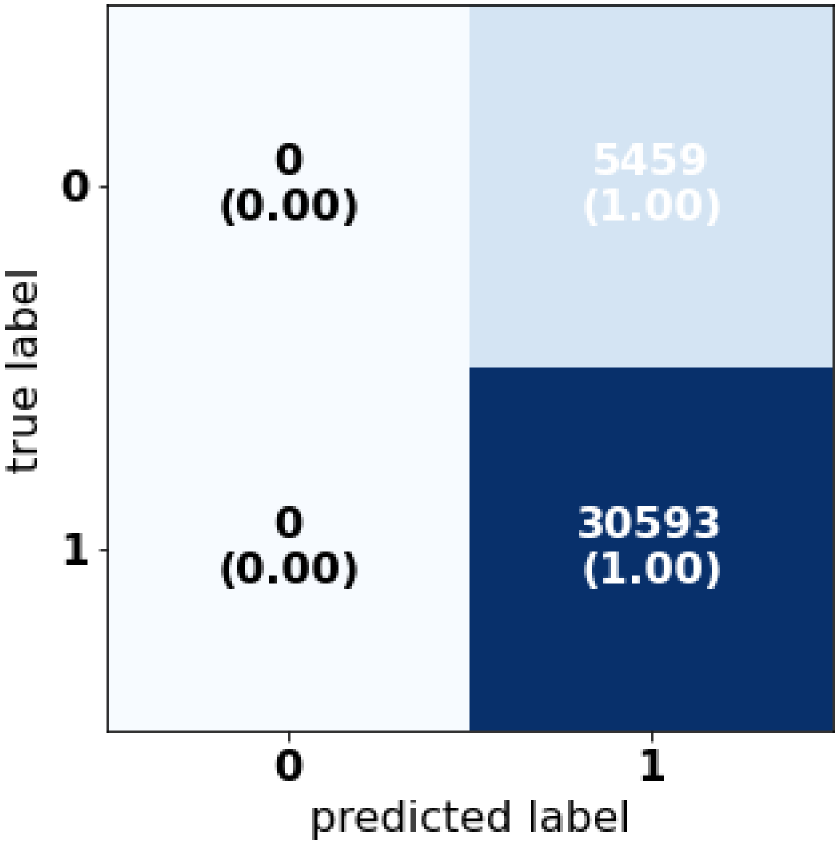
```
(109248, 5104) (109248,)
```

[ ]:
```
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.33,␣
 ↪stratify=y, random_state=2)
```

```
model = RandomForestClassifier(max_depth = 50, min_samples_split= 500,␣
 ↪random_state=2)

model = model.fit(xtrain, ytrain)
Y_pred = model.predict(xtest)
```

1.7 1.7 ConfusionMatrix

[ ]:
```
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
import matplotlib.pyplot as plt

font = {
'family' : 'DejaVu Sans',
'weight' : 'bold',
'size' : '16'
}

plt.rc('font', **font)
mat = confusion_matrix(ytest, Y_pred)
plot_confusion_matrix(conf_mat=mat, figsize=(5,5), show_normed=True);
```



[ ]:

```
from sklearn.metrics import auc

print("train_roc_auc_score : " , roc_auc_score(ytrain, model.predict(xtrain)),␣
 ↪'\n')
print("test_roc_auc_score : ", roc_auc_score(ytest, Y_pred), '\n')

probs = model.predict_proba(xtrain)
probs = probs[:, 1]

train_fpr, train_tpr, train_thresholds = roc_curve(ytrain, probs)

probs = model.predict_proba(xtest)
probs = probs[:, 1]

test_fpr, test_tpr, test_thresholds = roc_curve(ytest, probs)

print("train_auc_score : " , auc(train_fpr, train_tpr), '\n')
print("test_auc_score : ", auc(test_fpr, test_tpr), '\n')
```

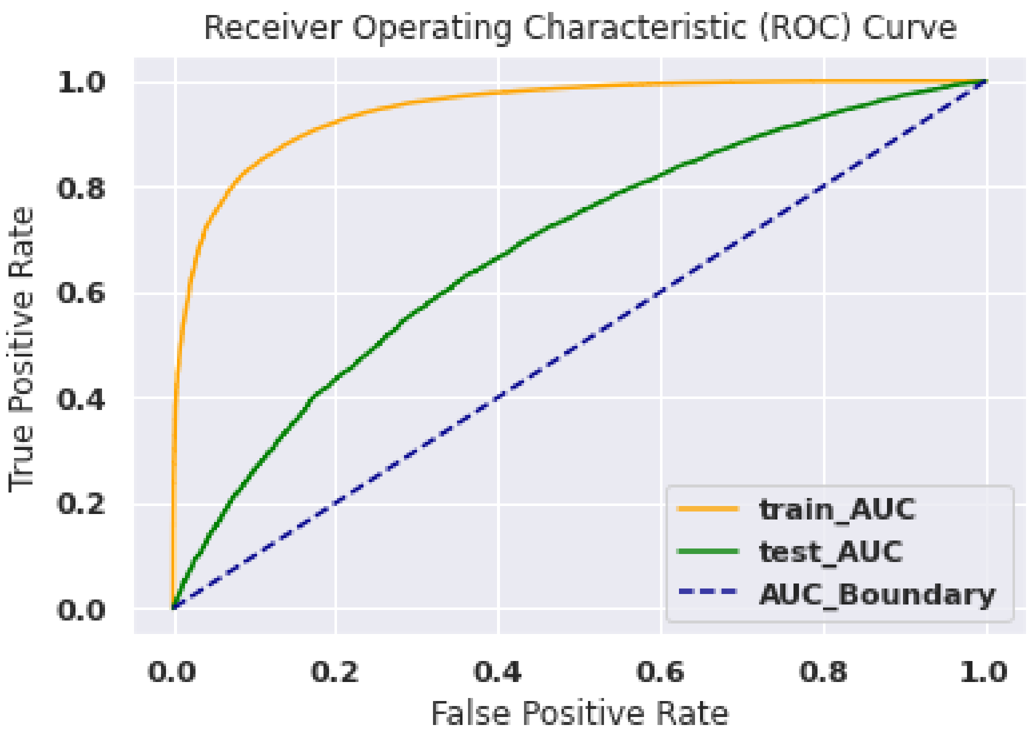
```
train_roc_auc_score :  0.5

test_roc_auc_score :  0.5

train_auc_score :  0.947276327767181

test_auc_score :  0.6814858183318782
```

1.8 1.8 AUC Plot

[ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="ticks")
sns.set(style='darkgrid')

print("train_auc_score : " , auc(train_fpr, train_tpr), "\n\n")
print("test_auc_score : ", auc(test_fpr, test_tpr), "\n\n")

plt.plot(train_fpr, train_tpr, color='orange', label='_train_ROC')
plt.plot(test_fpr, test_tpr, color='green', label='_test_ROC')

plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(['train_AUC', 'test_AUC', 'AUC_Boundary'])
plt.show();
```

```
train_auc_score :  0.947276327767181


test_auc_score :  0.6814858183318782
```



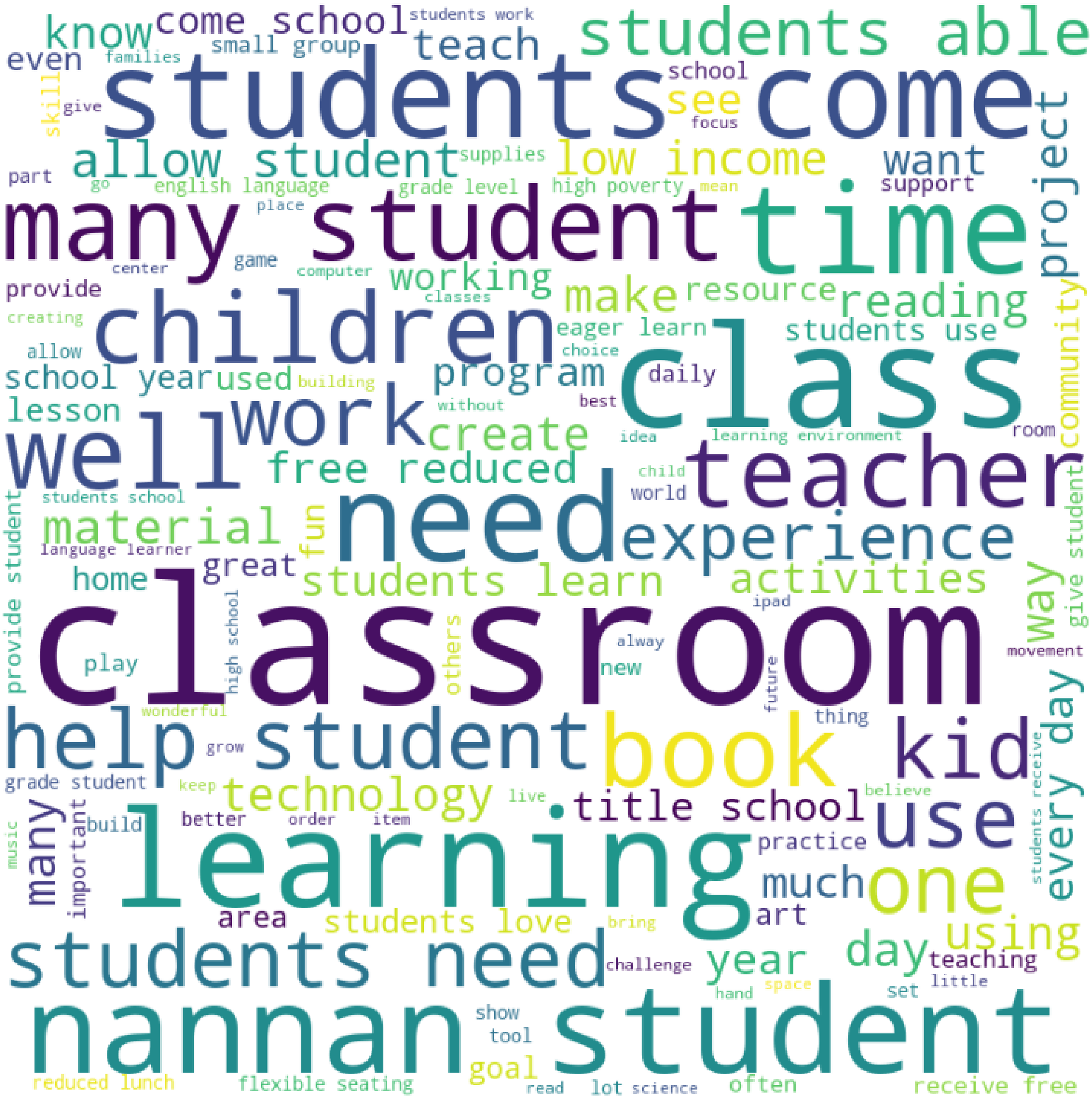
1.9 1.9 WordCloud

[ ]:

```
from wordcloud import WordCloud, STOPWORDS

Y_pred = Y_pred.tolist()

ytest = ytest.tolist()

false_positive = []

for index in range(len(Y_pred)):
    if ytest[index] == 0 and Y_pred[index] == 1:
        false_positive.append(index)
```

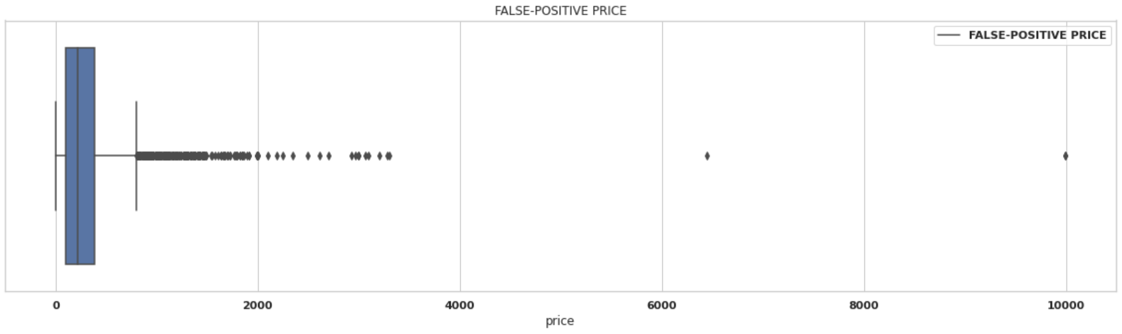
[ ]:

```
fp_essay = data.iloc[false_positive]['essay']
fp_price = data.iloc[false_positive]['price']
fp_teacher_number_of_previously_posted_projects = data.
 ↪iloc[false_positive]['teacher_number_of_previously_posted_projects']

stopwords = set(STOPWORDS)
word_cloud = []
comment_words = ""

for sentence in fp_essay:
    for words in sentence.split():
        word_cloud.append(words.lower())

comment_words += " ".join(word_cloud)+" "

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (10, 10), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
print('#'*50, '\n', 'ESSAYS - WORD_CLOUDS - FOR FALSE-POSITIVE LABELS')
```



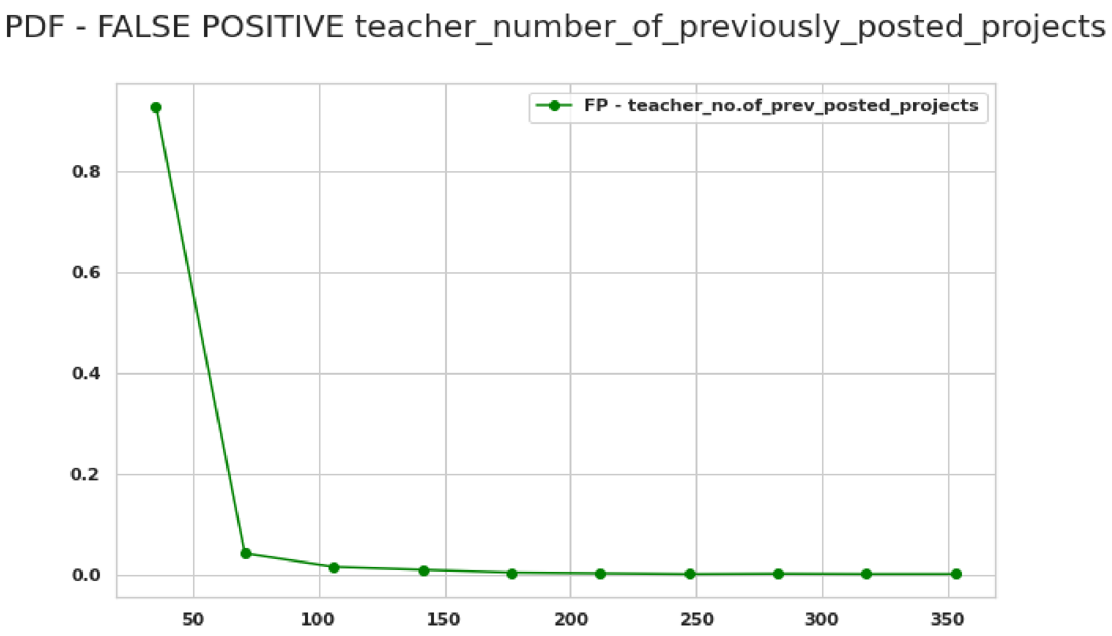
1.10 1.10 Box-Plot & PDF Plot

```
import seaborn as sns

plt.figure(figsize=(20,5))

sns.set_theme(style="whitegrid")
sns.boxplot(fp_price)

plt.title("FALSE-POSITIVE PRICE")
plt.legend(["FALSE-POSITIVE PRICE"])
plt.show()
```



[ ]:

```python
sns.set_style("whitegrid");
plt.figure(figsize=(10,6))

plt.rcParams['axes.titlesize'] = 20
plt.rcParams['axes.titleweight'] = 10

count, bin_edges = np.histogram(fp_teacher_number_of_previously_posted_projects, bins=10, density=True)

nodes_pdf = count / sum(count)
nodes_cdf = np.cumsum(nodes_pdf)

plt.plot(bin_edges[1:],nodes_pdf, color='green', marker='o', linestyle='solid')

plt.title("PDF - FALSE POSITIVE teacher_number_of_previously_posted_projects\n")
plt.legend(['FP - teacher_no.of_prev_posted_projects'])
plt.show();
```



2 TASK - 2:

3 DecisionTree Classifier

[1]:

```
data = pd.read_csv(preprocessed_data, nrows=50000)

preprocessed_essays = data.essay.values

y = data['project_is_approved']

data = data.drop(columns=['project_is_approved'])

data = np.column_stack((ohe_vector('school_state', data, 'train').toarray(),␣
 ↪ohe_vector('teacher_prefix',data, 'train').toarray(), \
                                    ohe_vector('project_grade_category', data,␣
 ↪'train').toarray(), ohe_vector('clean_categories', data, 'train').toarray(),␣
 ↪\
                                    ohe_vector('clean_subcategories', data,␣
 ↪'train').toarray(), normalized(data, 'price', 'train'), \
                                    normalized(data,␣
 ↪'teacher_number_of_previously_posted_projects', 'train'), \
                                    tfidf(),␣
 ↪sentiment_anayser(preprocessed_essays)))

print(data.shape, y.shape)
```

[ ]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

print("Final Data matrix")
print(data.shape, y.shape)

parameters={'min_samples_split' : [5, 10, 100, 500]}

gsc=GridSearchCV(estimator=DecisionTreeClassifier(random_state=2),
                    param_grid=parameters, scoring='roc_auc', verbose=1,␣
 ↪n_jobs=2, return_train_score=True)

grid_result = gsc.fit(data, y)

print("#"*50,"\n\n")
best_params=grid_result.best_params_
print(best_params,'\n')
print(grid_result.best_score_,"\n")
print("#"*50,"\n\n")
```

```
Final Data matrix
(50000, 5104) (50000,)
Fitting 5 folds for each of 4 candidates, totalling 20 fits

[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  20 out of  20 | elapsed: 137.1min finished

##################################################


{'min_samples_split': 500}

0.5668651141964209

##################################################
```

[13]:

```
from sklearn.tree import DecisionTreeClassifier

data = pd.read_csv('preprocessed_data.csv')

preprocessed_essays = data.essay.values

y = data['project_is_approved']

data = data.drop(columns=['project_is_approved'])
data = np.column_stack((ohe_vector('school_state', data, 'train').toarray(),␣
 ↪ohe_vector('teacher_prefix',data, 'train').toarray(), \
                                ohe_vector('project_grade_category', data,␣
 ↪'train').toarray(), ohe_vector('clean_categories', data, 'train').toarray(),␣
 ↪\
                                ohe_vector('clean_subcategories', data,␣
 ↪'train').toarray(), normalized(data, 'price', 'train'), \
                                normalized(data,␣
 ↪'teacher_number_of_previously_posted_projects', 'train'), \
                                tfidf(),␣
 ↪sentiment_anayser(preprocessed_essays)))
```

[20]:
```
from sklearn.feature_selection import SelectFromModel

selector = SelectFromModel(estimator=DecisionTreeClassifier(random_state=2,␣
 ↪min_samples_split=500), threshold=0.1, prefit=True)
X = selector.transform(data)
```

```
(109248, 1679)
```

[26]:
```
importantFeaturesCount = X.shape[1]

print("importantFeaturesCount : ", importantFeaturesCount)
```

```
importantFeaturesCount :  1679
```

[27]:
```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.33,␣
 ↪stratify=y, random_state=2)
```

[29]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

parameters={'max_depth' : [1, 5, 10, 50], 'min_samples_split' : [5, 10, 100,␣
 ↪500]}

gsc=GridSearchCV(estimator=DecisionTreeClassifier(random_state=2),
                    param_grid=parameters, scoring='roc_auc', verbose=1,␣
 ↪n_jobs=4, return_train_score=True)

grid_result = gsc.fit(xtrain, ytrain)

print("#"*50,"\n\n")
best_params=grid_result.best_params_
print(best_params,'\n')
print(grid_result.best_score_,"\n")
print("#"*50,"\n\n")
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:  4.8min
[Parallel(n_jobs=4)]: Done  80 out of  80 | elapsed: 23.5min finished

##################################################


{'max_depth': 10, 'min_samples_split': 500}

0.6268102361721721

##################################################
```

[30]:

```
grid_result.cv_results_
```

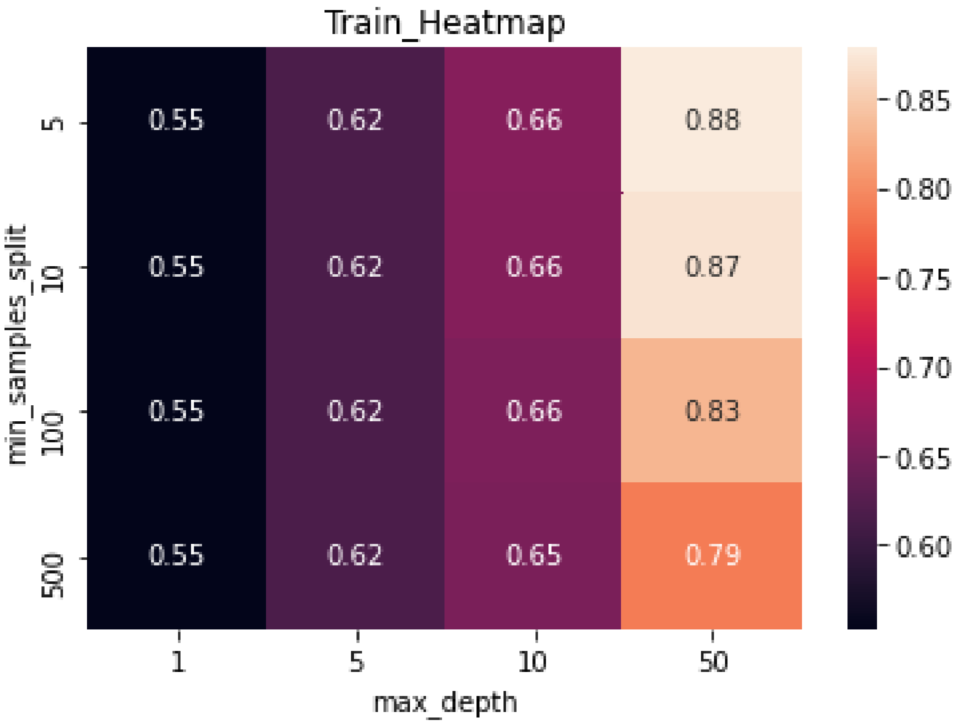
[30]:

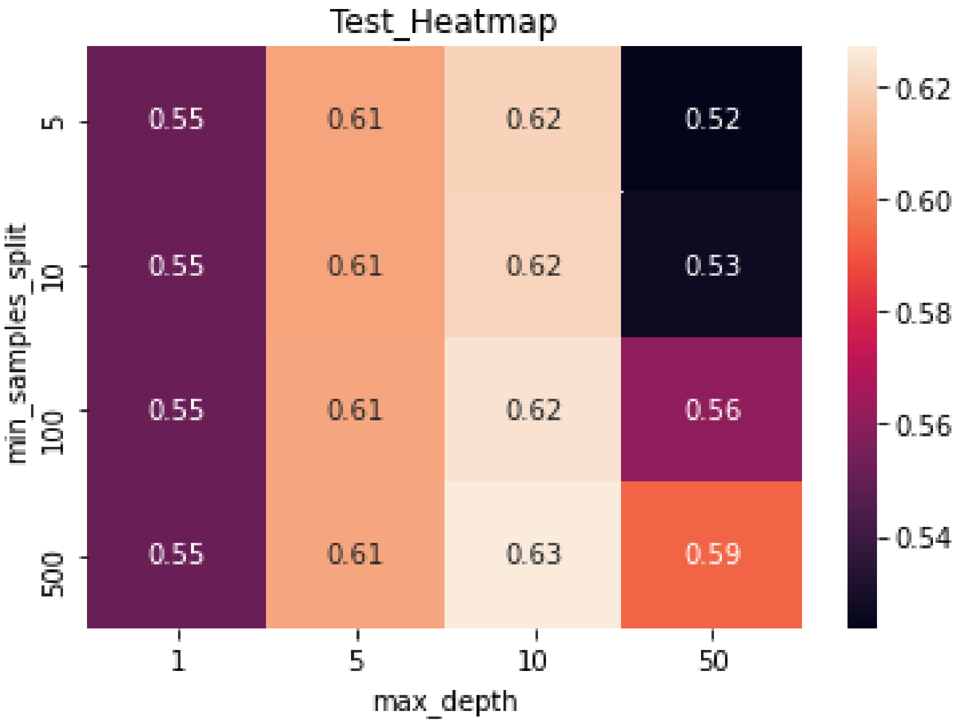
```
{'mean_fit_time': array([ 10.12753882,   8.09999394,   7.29023647,   7.02758694,
         26.93059802,  36.13386941,  33.41459179,  31.29397244,
         50.94235015,  50.73601561,  52.75119977,  51.36398392,
        175.85427504, 167.7394917 , 170.11450949, 172.25959954]),
 'std_fit_time': array([0.64262992, 1.07505015, 1.27339829, 1.20178953,
5.86733469,
        4.55851706, 2.98386299, 2.34795116, 0.61260725, 2.40681244,
        1.08491182, 0.65379627, 2.67186813, 8.9544772 , 9.54550965,
        4.72851698]),
 'mean_score_time': array([0.23497677, 0.24514294, 0.19906778, 0.26389422,
0.35205832,
        0.5702199 , 0.75993152, 0.98833003, 0.42901573, 0.63461351,
        0.36553612, 0.34298768, 0.37032399, 0.3805975 , 0.43811808,
        0.28789291]),
 'std_score_time': array([0.07477189, 0.06412987, 0.04178619, 0.04920263,
0.09344346,
        0.32132917, 0.59647497, 1.19247492, 0.16052255, 0.55539542,
        0.05362303, 0.05643156, 0.17448781, 0.07763461, 0.14805053,
        0.11210356]),
 'param_max_depth': masked_array(data=[1, 1, 1, 1, 5, 5, 5, 5, 10, 10, 10, 10,
50, 50, 50, 50],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_min_samples_split': masked_array(data=[5, 10, 100, 500, 5, 10, 100, 500,
5, 10, 100, 500, 5,
                    10, 100, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_depth': 1, 'min_samples_split': 5},
  {'max_depth': 1, 'min_samples_split': 10},
  {'max_depth': 1, 'min_samples_split': 100},
  {'max_depth': 1, 'min_samples_split': 500},
  {'max_depth': 5, 'min_samples_split': 5},
  {'max_depth': 5, 'min_samples_split': 10},
  {'max_depth': 5, 'min_samples_split': 100},
  {'max_depth': 5, 'min_samples_split': 500},
  {'max_depth': 10, 'min_samples_split': 5},
  {'max_depth': 10, 'min_samples_split': 10},
  {'max_depth': 10, 'min_samples_split': 100},
  {'max_depth': 10, 'min_samples_split': 500},
  {'max_depth': 50, 'min_samples_split': 5},
  {'max_depth': 50, 'min_samples_split': 10},
  {'max_depth': 50, 'min_samples_split': 100},
  {'max_depth': 50, 'min_samples_split': 500}],
 'split0_test_score': array([0.54991669, 0.54991669, 0.54991669, 0.54991669,
0.61953741,
        0.61953741, 0.61956777, 0.61956899, 0.61500572, 0.61483031,
        0.62038044, 0.62564328, 0.52781758, 0.52690655, 0.54825766,
        0.58916461]),
 'split1_test_score': array([0.54699096, 0.54699096, 0.54699096, 0.54699096,
0.60165914,
        0.60165914, 0.60205263, 0.60201729, 0.61608763, 0.61576306,
        0.61664467, 0.61728689, 0.53358103, 0.53565412, 0.56925128,
        0.59903242]),
 'split2_test_score': array([0.55029373, 0.55029373, 0.55029373, 0.55029373,
0.59781901,
        0.59826991, 0.59835224, 0.59839957, 0.61736219, 0.61877309,
        0.62269913, 0.62492027, 0.51774688, 0.52362568, 0.56428371,
        0.58165636]),
 'split3_test_score': array([0.56023942, 0.56023942, 0.56023942, 0.56023942,
0.62277338,
        0.62277338, 0.62294854, 0.62316155, 0.63123346, 0.63028647,
        0.6362719 , 0.63785983, 0.50535887, 0.51467078, 0.54565759,
        0.58685232]),
 'split4_test_score': array([0.5520288 , 0.5520288 , 0.5520288 , 0.5520288 ,
0.5986128 ,
        0.5986181 , 0.59890244, 0.59884134, 0.62123223, 0.62311492,
        0.62514264, 0.62834091, 0.53431841, 0.53591985, 0.57685787,
        0.6093725 ]),
 'mean_test_score': array([0.55189392, 0.55189392, 0.55189392, 0.55189392,
0.60808035,
        0.60817159, 0.60836472, 0.60839775, 0.62018425, 0.62055357,
        0.62422776, 0.62681024, 0.52376456, 0.5273554 , 0.56086162,
        0.59321564]),
 'std_test_score': array([0.00447604, 0.00447604, 0.00447604, 0.00447604,
0.01080103,
        0.01071558, 0.01065667, 0.01072161, 0.0059118 , 0.00565976,
        0.00664139, 0.00663422, 0.01094674, 0.00796531, 0.01206648,
        0.0098529 ]),
 'rank_test_score': array([11, 11, 11, 11,  8,  7,  6,  5,  4,  3,  2,  1, 16,
15, 10,  9]),
 'split0_train_score': array([0.55354557, 0.55354557, 0.55354557, 0.55354557,
0.62294086,
        0.62294086, 0.62293718, 0.62288509, 0.66680549, 0.66604051,
        0.66077618, 0.65694741, 0.88241616, 0.87399606, 0.84352357,
        0.80047516]),
 'split1_train_score': array([0.55496537, 0.55496537, 0.55496537, 0.55496537,
0.62120702,
        0.62120702, 0.62060929, 0.62056851, 0.66511342, 0.66445597,
        0.65789983, 0.65610117, 0.87692563, 0.87072161, 0.82973594,
        0.79512541]),
 'split2_train_score': array([0.55414992, 0.55414992, 0.55414992, 0.55414992,
0.61800868,
        0.61796862, 0.61762189, 0.61746791, 0.664008  , 0.6627672 ,
        0.65696171, 0.65537568, 0.86671582, 0.85921434, 0.81585176,
        0.77868528]),
 'split3_train_score': array([0.54935098, 0.54935098, 0.54935098, 0.54935098,
0.61498007,
        0.61498007, 0.6149201 , 0.61461447, 0.6584266 , 0.65736237,
        0.65283302, 0.64930116, 0.87982448, 0.87453064, 0.83614438,
        0.78888891]),
 'split4_train_score': array([0.55348423, 0.55348423, 0.55348423, 0.55348423,
0.61070205,
        0.61060385, 0.61052222, 0.61025361, 0.66010964, 0.65906282,
        0.65564171, 0.65130145, 0.88362585, 0.87642647, 0.83999114,
        0.7812217 ]),
 'mean_train_score': array([0.55309921, 0.55309921, 0.55309921, 0.55309921,
0.61756773,
        0.61754008, 0.61732214, 0.61715792, 0.66289263, 0.66193778,
        0.65682249, 0.65380537, 0.87790159, 0.87097782, 0.83304936,
        0.78887929]),
 'std_train_score': array([0.0019485 , 0.0019485 , 0.0019485 , 0.0019485 ,
0.00438504,
        0.00441505, 0.00434586, 0.00444224, 0.00313614, 0.00325765,
        0.00261235, 0.00297209, 0.00604847, 0.00616223, 0.00973871,
        0.00819799])}
```

[31]:

```
params = grid_result.cv_results_['params']
```

```
mean_train_score =  pd.Series(grid_result.cv_results_['mean_train_score'])
mean_test_score =  pd.Series(grid_result.cv_results_['mean_test_score'])

min_samples_split = []
max_depth = []

for parameter in params:
    min_samples_split.append(parameter['min_samples_split'])
    max_depth.append(parameter['max_depth'])

df = pd.DataFrame()

df['min_samples_split'] = pd.Series(min_samples_split)
df['mean_test_score'] = mean_test_score
df['mean_train_score'] = mean_train_score
df['max_depth'] = pd.Series(max_depth)

train_heatmap = df.pivot(index='min_samples_split', columns='max_depth',␣
 ↪values='mean_train_score')
test_heatmap = df.pivot(index='min_samples_split', columns='max_depth',␣
 ↪values='mean_test_score')

sns.heatmap(train_heatmap, annot=True)
plt.title("Train_Heatmap")
plt.show()

sns.heatmap(test_heatmap, annot=True)
plt.title("Test_Heatmap")
plt.show()
```


Train_Heatmap
min_samples_split
5
10
100
500
max_depth
1
5
10
50
0.55
0.62
0.66
0.88
0.55
0.62
0.66
0.87
0.55
0.62
0.66
0.83
0.55
0.62
0.65
0.79
0.85
0.80
0.75
0.70
0.65
0.60

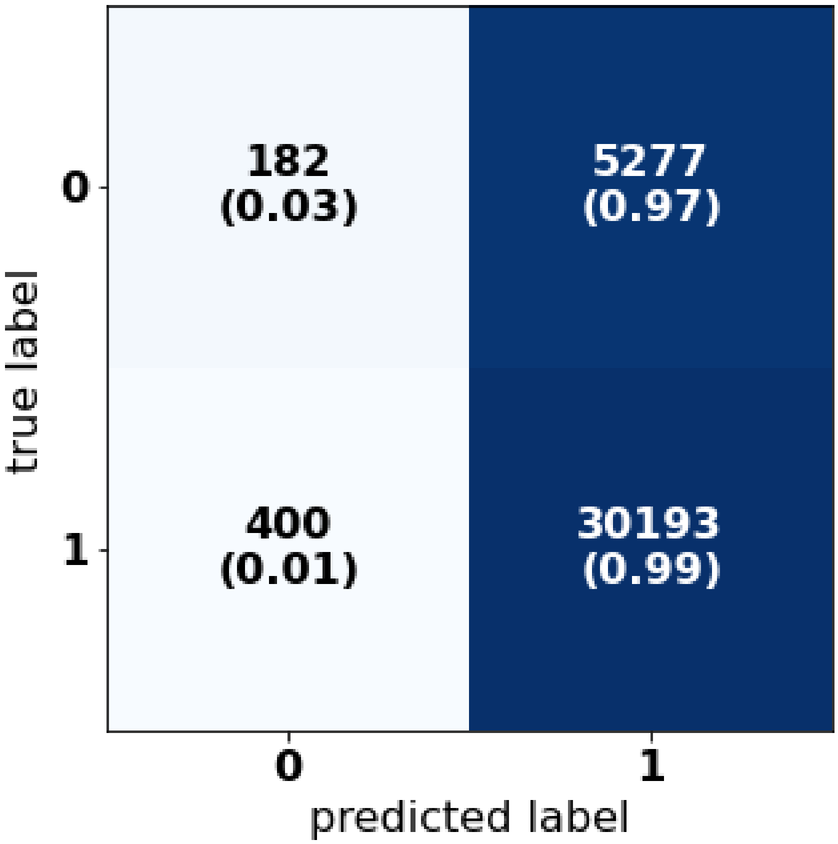

Test_Heatmap
min_samples_split
5
10
100
500
max_depth
1
5
10
50
0.55
0.61
0.62
0.52
0.55
0.61
0.62
0.53
0.55
0.61
0.62
0.56
0.55
0.61
0.63
0.59
0.62
0.60
0.58
0.56
0.54

[32]:

```
# # https://plot.ly/python/3d-axes/

trace1 = go.Scatter3d(x=min_samples_split,y=max_depths,z=mean_test_score.
 ↪tolist(), name = 'cv_test_score')
trace2 = go.Scatter3d(x=min_samples_split,y=max_depths,z=mean_train_score.
 ↪tolist(), name = 'cv_train_score')

data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='n_estimators'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC')))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

[33]:

```
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
import matplotlib.pyplot as plt

xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.33,␣
 ↪stratify=y, random_state=2)

model = DecisionTreeClassifier(max_depth = 10, min_samples_split= 500,␣
 ↪random_state=2)

model = model.fit(xtrain, ytrain)
Y_pred = model.predict(xtest)

font = {
'family' : 'DejaVu Sans',
'weight' : 'bold',
'size' : '16'
}

plt.rc('font', **font)
mat = confusion_matrix(ytest, Y_pred)
plot_confusion_matrix(conf_mat=mat, figsize=(5,5), show_normed=True);
```



[34]:

```
from sklearn.metrics import auc

print("train_roc_auc_score : " , roc_auc_score(ytrain, model.predict(xtrain)),␣
 ↪'\n')
print("test_roc_auc_score : ", roc_auc_score(ytest, Y_pred), '\n')

probs = model.predict_proba(xtrain)
probs = probs[:, 1]

train_fpr, train_tpr, train_thresholds = roc_curve(ytrain, probs)

probs = model.predict_proba(xtest)
probs = probs[:, 1]

test_fpr, test_tpr, test_thresholds = roc_curve(ytest, probs)

print("train_auc_score : " , auc(train_fpr, train_tpr), '\n')
print("test_auc_score : ", auc(test_fpr, test_tpr), '\n')
```

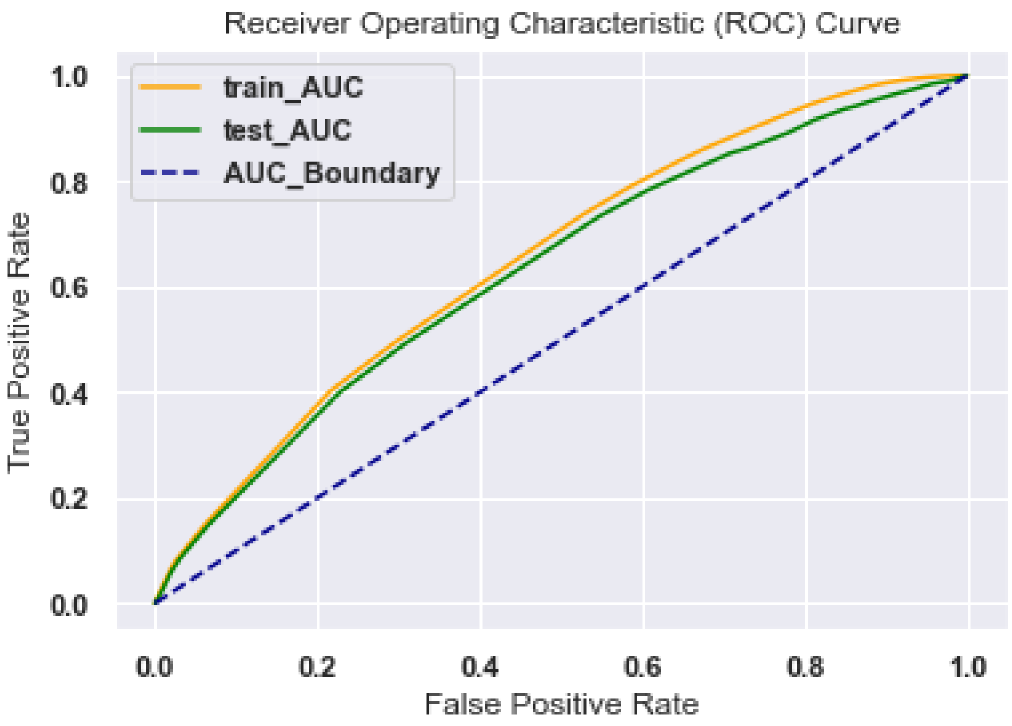
```
train_roc_auc_score :  0.5313425324030288
```

```
test_roc_auc_score :  0.5101322765229259

train_auc_score :  0.6534066126846589

test_auc_score :  0.6324891455120432
```

[35]:

```python
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="ticks")
sns.set(style='darkgrid')

print("train_auc_score : " , auc(train_fpr, train_tpr), "\n\n")
print("test_auc_score : ", auc(test_fpr, test_tpr), "\n\n")

plt.plot(train_fpr, train_tpr, color='orange', label='_train_ROC')
plt.plot(test_fpr, test_tpr, color='green', label='_test_ROC')

plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(['train_AUC', 'test_AUC', 'AUC_Boundary'])
plt.show();
```

```
train_auc_score :  0.6534066126846589


test_auc_score :  0.6324891455120432
```



[37]:

```
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper_Parameter", "Train_AUC",␣
 ↪"Test_AUC"]
x.add_row(["TFIDF-W2V", 'RandomForest', 'max_depth : 50, min_samples_split :␣
 ↪500', 0.95, 0.68])
x.add_row(["TFIDF", 'RandomForest', 'max_depth : 50, min_samples_split : 500',␣
 ↪0.78, 0.65])
x.add_row(["TFIDF", 'DecisionTree', 'max_depth : 10, min_samples_split : 500',␣
 ↪0.65, 0.63])
print(x)
```

```
+------------+--------------+------------------------------------------+---------
--+----------+
| Vectorizer |    Model     |             Hyper_Parameter              |
Train_AUC | Test_AUC |
+------------+--------------+------------------------------------------+---------
--+----------+
| TFIDF-W2V  | RandomForest | max_depth : 50, min_samples_split : 500 |    0.95
|   0.68   |
|   TFIDF    | RandomForest | max_depth : 50, min_samples_split : 500 |    0.78
|   0.65   |
|   TFIDF    | DecisionTree | max_depth : 10, min_samples_split : 500 |    0.65
|   0.63   |
+------------+--------------+-----------------------------------------+---------
--+----------+
```