RandomForest_Assignment

November 23, 2020

[ ]:
```
!pip install pandas==1.1.3
```

```
Requirement already satisfied: pandas==1.1.3 in /usr/local/lib/python3.6/dist-
packages (1.1.3)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.6/dist-
packages (from pandas==1.1.3) (1.18.5)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.6/dist-packages (from pandas==1.1.3) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-
packages (from pandas==1.1.3) (2018.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-
packages (from python-dateutil>=2.7.3->pandas==1.1.3) (1.15.0)
```

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

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

[1]:
```
import pickle

glove_vectors_path = '/content/drive/MyDrive/6_Donors_choose_NB/glove_vectors'
preprocessed_data = '/content/drive/MyDrive/6_Donors_choose_NB/
↪preprocessed_data.csv'
```

1 TASK 1:

##**RandomForest Classifier**

[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)
```

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

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

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!
```

[2]: True

1.1 1.1 LoadingData

[3]:
```python
data = pd.read_csv(preprocessed_data)
data = pd.concat([data.loc[data['project_is_approved'] == 0], data.
 ↪loc[data['project_is_approved'] == 1].head(33458)], sort=False)

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

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 7 to 39521
Data columns (total 8 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   clean_categories                              50000 non-null  object
 5   clean_subcategories                           50000 non-null  object
 6   essay                                         50000 non-null  object
 7   price                                         50000 non-null  float64
dtypes: float64(1), int64(1), object(6)
memory usage: 3.4+ MB
```

[4]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, ytrain, ytest = train_test_split(data, target, test_size=0.33,␣
 ↪stratify=target)

del data
del target
```

1.2 1.2 TextFeatures Encoding

1.2.1 1.2.1 Tfidf Vectorization

[5]:
```
############### TFIDF - Vectorizer ######################
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(min_df=10, max_features=5000, ngram_range=(1,2))
tfidf.fit(X_train.essay.values)
```

[5]:
```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_features=5000,
                min_df=10, ngram_range=(1, 2), norm='l2', preprocessor=None,
                smooth_idf=True, stop_words=None, strip_accents=None,
                sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, use_idf=True, vocabulary=None)
```

1.2.2 1.2.2 Tfidf-W2V vectorizer

[ ]:
```
################# TFIDF - W2V ###########################
import tqdm

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

def tfidf_w2v(essay):
    tfidf = TfidfVectorizer(min_df=10, max_features=5000, ngram_range=(1,2))
    tfidf.fit(essay.values)

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

    tfidf_w2v_vectors = []

    for sentence in essay:
        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 & Normalize Numeric Data

[6]:
```
from sklearn.preprocessing import Normalizer
from sklearn.feature_extraction.text import CountVectorizer

ohe1 = CountVectorizer()
ohe1.fit(X_train['school_state'].values)
school_state_ohe_train = ohe1.transform(X_train['school_state'].values).
 ↪toarray() # fit has to happen only on train data
school_state_ohe_test = ohe1.transform(X_test['school_state'].values).toarray()

ohe2 = CountVectorizer()
ohe2.fit(X_train['teacher_prefix'].values)
teacher_prefix_ohe_train = ohe2.transform(X_train['teacher_prefix'].values).
 ↪toarray() # fit has to happen only on train data
teacher_prefix_ohe_test = ohe2.transform(X_test['teacher_prefix'].values).
 ↪toarray()

ohe3 = CountVectorizer()
ohe3.fit(X_train['project_grade_category'].values)
project_grade_category_ohe_train = ohe3.
 ↪transform(X_train['project_grade_category'].values).toarray() # fit has to␣
 ↪happen only on train data
project_grade_category_ohe_test = ohe3.
 ↪transform(X_test['project_grade_category'].values).toarray()

ohe4 = CountVectorizer()
ohe4.fit(X_train['clean_categories'].values)
clean_categories_ohe_train = ohe4.transform(X_train['clean_categories'].values).
 ↪toarray() # fit has to happen only on train data
clean_categories_ohe_test = ohe4.transform(X_test['clean_categories'].values).
 ↪toarray()

ohe5 = CountVectorizer()
ohe5.fit(X_train['clean_subcategories'].values)
clean_subcategories_ohe_train = ohe5.transform(X_train['clean_subcategories'].
 ↪values).toarray() # fit has to happen only on train data
clean_subcategories_ohe_test = ohe5.transform(X_test['clean_subcategories'].
 ↪values).toarray()

norm = Normalizer()
```

```
norm.fit(X_train['price'].values.reshape(-1,1))
X_train_price_norm = norm.transform(X_train['price'].values.reshape(-1,1))
X_test_price_norm = norm.transform(X_test['price'].values.reshape(-1,1))

norm.fit(X_train['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(-1,1))
X_train_no_prev_proj = norm.
 ↪transform(X_train['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(-1,1))
X_test_no_prev_proj = norm.
 ↪transform(X_test['teacher_number_of_previously_posted_projects'].values.
 ↪reshape(-1,1))
```

[7]:
```
import gc

X_train_essay = X_train['essay']
X_test_essay = X_test['essay']

X_train = np.column_stack((X_train_no_prev_proj, X_train_price_norm,␣
 ↪school_state_ohe_train, teacher_prefix_ohe_train,␣
 ↪project_grade_category_ohe_train, clean_categories_ohe_train,␣
 ↪clean_subcategories_ohe_train))
X_test = np.column_stack((X_test_no_prev_proj, X_test_price_norm,␣
 ↪school_state_ohe_test, teacher_prefix_ohe_test,␣
 ↪project_grade_category_ohe_test, clean_categories_ohe_test,␣
 ↪clean_subcategories_ohe_test))

gc.collect()

print("Final Data matrix")
print(X_train.shape, ytrain.shape)
print(X_test.shape, ytest.shape)

print("="*100)
```

```
Final Data matrix
(33500, 101) (33500,)
(16500, 101) (16500,)
================================================================================
====================
```

1.4 1.4 Sentiment Analyser

[8]:
```
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 Hyper_Parameter Tuning (SET1 & SET2)

[12]:
```
from sklearn.model_selection import GridSearchCV

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

X_train = np.column_stack((X_train, sentiment_anayser(X_train_essay)))

for index, df_set in enumerate(sets):

    if df_set == 'tfidf':
        temp = tfidf.transform(X_train_essay).toarray()
        X = np.column_stack((X_train, temp))

    elif df_set == 'tfidf_w2v_vectors':
        temp = tfidf_w2v(X_train_essay)
        X = np.column_stack((X_train, temp))

    print("Final Data matrix")
    print(X.shape, ytrain.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, ytrain)
    scorer[df_set] = grid_result.cv_results_
```

```python
    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
    del temp
    gc.collect()
```

```
Final Data matrix
(33500, 404) (33500,)
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:  8.4min
[Parallel(n_jobs=2)]: Done  80 out of  80 | elapsed: 23.9min finished

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



 tfidf_w2v_vectors  :

{'max_depth': 50, 'min_samples_split': 100}
0.6811385798074181

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


Final Data matrix
(33500, 5104) (33500,)
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:  6.2min
[Parallel(n_jobs=2)]: Done  80 out of  80 | elapsed: 27.8min finished

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



 tfidf  :

{'max_depth': 50, 'min_samples_split': 500}
0.7034708676646971
```

##

[13]:
```
scorer['tfidf_w2v_vectors']
```

[13]:
```
{'mean_fit_time': array([ 6.53642168,  6.50008607,  6.446486  ,  6.50317445,
27.50075483,
        27.53363948, 28.96187649, 27.01472712, 48.22469082, 48.23755665,
        46.74311256, 39.58104491, 70.05630026, 69.74470596, 57.81148353,
        40.45910616]),
 'mean_score_time': array([0.08072991, 0.07817049, 0.07889657, 0.07854581,
0.10959311,
        0.11522694, 0.1120954 , 0.1117126 , 0.15585575, 0.15923347,
        0.15560122, 0.13585005, 0.24112983, 0.23229437, 0.18090439,
        0.13065448]),
 'mean_test_score': array([0.64377895, 0.64377895, 0.64377895, 0.64377895,
0.66769097,
        0.66770627, 0.66716528, 0.66538161, 0.68046042, 0.6801951 ,
        0.68093711, 0.67588345, 0.67066628, 0.67177212, 0.68113858,
        0.67665995]),
 'mean_train_score': array([0.6512802 , 0.6512802 , 0.6512802 , 0.6512802 ,
0.70963937,
        0.70960247, 0.70685385, 0.69939117, 0.9209857 , 0.91508418,
        0.84125539, 0.75034393, 0.99999963, 0.99999773, 0.93422663,
        0.7603309 ]),
 '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, 11, 12,  3,  4,  2,  6,  8,
7,  1,  5],
      dtype=int32),
 'split0_test_score': array([0.65225862, 0.65225862, 0.65225862, 0.65225862,
0.67323694,
        0.6737825 , 0.67324238, 0.67329884, 0.68545447, 0.68450917,
        0.68852495, 0.68191914, 0.68022579, 0.67976537, 0.68704446,
        0.68283515]),
 'split0_train_score': array([0.64953667, 0.64953667, 0.64953667, 0.64953667,
0.70905223,
        0.70935749, 0.70632675, 0.69851592, 0.92099028, 0.91735801,
        0.84221141, 0.75080342, 0.99999979, 0.99999875, 0.93579266,
        0.76077153]),
 'split1_test_score': array([0.63748568, 0.63748568, 0.63748568, 0.63748568,
0.6655002 ,
        0.66512119, 0.66639266, 0.6611503 , 0.67913542, 0.67625625,
        0.67656894, 0.67377279, 0.66709789, 0.6726941 , 0.68042249,
        0.67515235]),
 'split1_train_score': array([0.65148669, 0.65148669, 0.65148669, 0.65148669,
0.71027401,
        0.70953038, 0.7080616 , 0.70028313, 0.92186035, 0.91527295,
        0.84298397, 0.75127774, 0.99999981, 0.99999918, 0.93336489,
        0.75986778]),
 'split2_test_score': array([0.64099347, 0.64099347, 0.64099347, 0.64099347,
0.66707164,
        0.66726573, 0.66543936, 0.66326918, 0.6805472 , 0.68111789,
        0.67800323, 0.67475234, 0.66556301, 0.67250489, 0.67705725,
        0.67504936]),
 'split2_train_score': array([0.65183993, 0.65183993, 0.65183993, 0.65183993,
0.71043677,
        0.71064589, 0.7077439 , 0.69932875, 0.92184769, 0.91639154,
        0.84155127, 0.74971237, 0.99999934, 0.99999654, 0.9351048 ,
        0.76099081]),
 'split3_test_score': array([0.64729564, 0.64729564, 0.64729564, 0.64729564,
0.66766442,
        0.66755817, 0.66778345, 0.66578693, 0.68282579, 0.68452157,
        0.68535713, 0.67594992, 0.67536303, 0.67384927, 0.68525898,
        0.6749981 ]),
```

```
 'split3_train_score': array([0.65369477, 0.65369477, 0.65369477, 0.65369477,
0.70892403,
        0.70866214, 0.70713155, 0.69868701, 0.92113169, 0.91314401,
        0.84010287, 0.74955807, 0.99999975, 0.99999661, 0.93385501,
        0.75946172]),
 'split4_test_score': array([0.64086132, 0.64086132, 0.64086132, 0.64086132,
0.66498166,
        0.66480377, 0.66296854, 0.6634028 , 0.67433921, 0.67457063,
        0.67623129, 0.67302306, 0.66508167, 0.66004696, 0.67590972,
        0.67526478]),
 'split4_train_score': array([0.64984297, 0.64984297, 0.64984297, 0.64984297,
0.7095098 ,
        0.70981646, 0.70500546, 0.70014102, 0.91909847, 0.91325439,
        0.83942744, 0.75036805, 0.99999948, 0.99999755, 0.93301579,
        0.76056266]),
 'std_fit_time': array([0.04240422, 0.0282269 , 0.01749522, 0.02833217,
0.09849739,
        0.13856431, 1.88796702, 0.08671036, 0.09547524, 0.1039718 ,
        0.18242358, 0.08356622, 0.2205407 , 0.23923857, 0.08477348,
        1.27085611]),
 'std_score_time': array([0.00251925, 0.00166125, 0.00202288, 0.00262832,
0.00205439,
        0.00710582, 0.00436638, 0.00165003, 0.00270725, 0.00183028,
        0.00790558, 0.00641436, 0.00860267, 0.00127603, 0.0031816 ,
        0.01509835]),
 'std_test_score': array([0.00529431, 0.00529431, 0.00529431, 0.00529431,
0.00294217,
        0.00323245, 0.00342057, 0.00422206, 0.00373545, 0.00413111,
        0.00503876, 0.00317333, 0.00605654, 0.00643542, 0.00439017,
        0.00308895]),
 'std_train_score': array([1.50277183e-03, 1.50277183e-03, 1.50277183e-03,
1.50277183e-03,
        6.18359675e-04, 6.45591870e-04, 1.09748188e-03, 7.24384041e-04,
        1.00906177e-03, 1.67496269e-03, 1.31599579e-03, 6.48098382e-04,
        1.87115789e-07, 1.08274464e-06, 1.05550785e-03, 5.75031309e-04])}
```

[14]:

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

[14]:

```
{'mean_fit_time': array([  5.4144527 ,   4.95163836,   5.29698291,   5.09564219,
         17.84639125,  17.83705859,  17.82873521,  17.61748381,
         33.17415762,  33.25331221,  33.3099905 ,  32.27718129,
        107.95508366, 105.7434628 , 102.11645894,  88.4779398 ]),
 'mean_score_time': array([0.19195576, 0.197367  , 0.20865006, 0.20786195,
0.28999205,
        0.29331164, 0.28994403, 0.28318238, 0.38448644, 0.38372707,
        0.38623033, 0.37853475, 0.94770823, 0.94924035, 0.89747052,
        0.71370134]),
```

```
 'mean_test_score': array([0.64550969, 0.64550969, 0.64550969, 0.64550969,
0.67746338,
        0.6771413 , 0.67621447, 0.6768039 , 0.688689  , 0.68759127,
        0.68912801, 0.68800068, 0.69509522, 0.69683486, 0.70105653,
        0.70347087]),
 'mean_train_score': array([0.65614264, 0.65614264, 0.65614264, 0.65614264,
0.72598749,
        0.72485513, 0.71822928, 0.70981083, 0.82096228, 0.81520093,
        0.78376446, 0.7529182 , 0.99996942, 0.99989844, 0.9897343 ,
        0.91358433]),
 '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,  9, 10, 12, 11,  6,  8,  5,  7,  4,
3,  2,  1],
       dtype=int32),
 'split0_test_score': array([0.65619183, 0.65619183, 0.65619183, 0.65619183,
0.68355844,
        0.68297876, 0.6816136 , 0.68206919, 0.69333407, 0.69280994,
        0.69735539, 0.69575664, 0.69723754, 0.70493398, 0.70518276,
        0.70546948]),
```

```
 'split0_train_score': array([0.65568682, 0.65568682, 0.65568682, 0.65568682,
0.72747326,
       0.72543707, 0.71815314, 0.7100472 , 0.82069607, 0.81679627,
       0.78425842, 0.75411394, 0.99996219, 0.99989973, 0.98982969,
       0.91207297]),
 'split1_test_score': array([0.63798198, 0.63798198, 0.63798198, 0.63798198,
0.67072777,
       0.67105807, 0.67140743, 0.67134041, 0.68257928, 0.68343093,
       0.68101872, 0.6812916 , 0.69229925, 0.69156268, 0.69602293,
       0.69858373]),
 'split1_train_score': array([0.65900488, 0.65900488, 0.65900488, 0.65900488,
0.72579111,
       0.72561733, 0.71900266, 0.71123211, 0.82154784, 0.8136665 ,
       0.78444189, 0.75374008, 0.99998129, 0.99990076, 0.989594  ,
       0.91232937]),
 'split2_test_score': array([0.63827222, 0.63827222, 0.63827222, 0.63827222,
0.67255102,
       0.67152575, 0.6704227 , 0.67104068, 0.68366659, 0.68166745,
       0.68515288, 0.67999563, 0.68912338, 0.692233  , 0.69381267,
       0.69723058]),
 'split2_train_score': array([0.65834003, 0.65834003, 0.65834003, 0.65834003,
0.7268589 ,
       0.7252725 , 0.71852551, 0.71028445, 0.82121949, 0.81623948,
       0.78427716, 0.75378117, 0.99995666, 0.99987615, 0.98971246,
       0.9144533 ]),
 'split3_test_score': array([0.65794183, 0.65794183, 0.65794183, 0.65794183,
0.68579164,
       0.68600907, 0.68498289, 0.68546705, 0.69648165, 0.69837916,
       0.69752624, 0.69655621, 0.70230599, 0.70152491, 0.70902968,
       0.71016417]),
 'split3_train_score': array([0.65569421, 0.65569421, 0.65569421, 0.65569421,
0.72543973,
       0.72412582, 0.71808008, 0.70973387, 0.81852643, 0.81464166,
       0.78404954, 0.75199682, 0.99997737, 0.99990561, 0.98961929,
       0.91269741]),
 'split4_test_score': array([0.63716057, 0.63716057, 0.63716057, 0.63716057,
0.67468805,
       0.67413486, 0.67264575, 0.67410216, 0.68738343, 0.68166886,
       0.68458682, 0.68640333, 0.69450994, 0.69391973, 0.70123458,
       0.70590637]),
 'split4_train_score': array([0.65198726, 0.65198726, 0.65198726, 0.65198726,
0.72437446,
       0.72382291, 0.71738502, 0.7077565 , 0.82282156, 0.81466071,
       0.78179528, 0.75095901, 0.9999696 , 0.99990994, 0.98991604,
       0.91636859]),
 'std_fit_time': array([0.3253636 , 0.12302478, 0.65566589, 0.09141083,
0.22683699,
```

```
        0.03960877, 0.06778919, 0.06272197, 0.18287297, 0.22440627,
        0.17227151, 0.14854239, 0.86539327, 0.51063712, 0.61187103,
        0.97069797]),
 'std_score_time': array([0.01100982, 0.01067072, 0.01392603, 0.00456919,
0.00684582,
        0.00494261, 0.00228153, 0.00325137, 0.00505183, 0.0096278 ,
        0.00169747, 0.01392609, 0.01097866, 0.01293826, 0.02820086,
        0.09488462]),
 'std_test_score': array([0.00945961, 0.00945961, 0.00945961, 0.00945961,
0.00606155,
        0.00616915, 0.00592324, 0.00588451, 0.00541406, 0.00679835,
        0.00693404, 0.00699994, 0.00448124, 0.00538634, 0.0056292 ,
        0.00484875]),
 'std_train_score': array([2.47777955e-03, 2.47777955e-03, 2.47777955e-03,
2.47777955e-03,
        1.08695485e-03, 7.33646501e-04, 5.34173247e-04, 1.14248901e-03,
        1.40500500e-03, 1.14741516e-03, 9.92442774e-04, 1.22779209e-03,
        9.15543034e-06, 1.17285936e-05, 1.22894983e-04, 1.62191152e-03])}
```

1.6 1.6 Cross-Validation Results

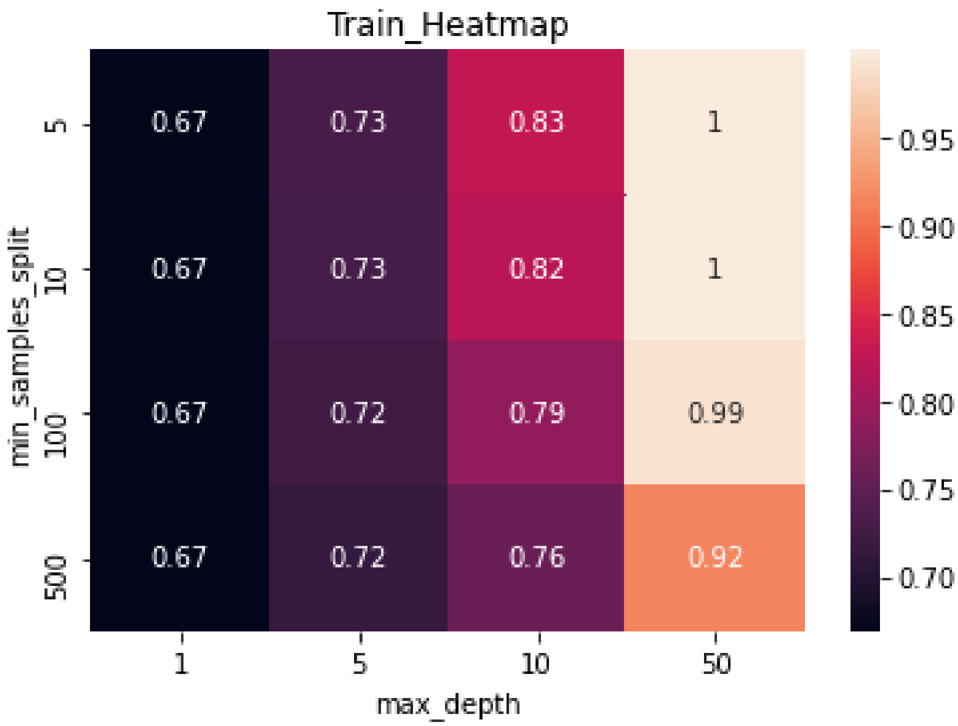
[15]:
```
mean_test_score = [0.65820944, 0.65820944, 0.65820944, 0.65820944, 0.68122492,␣
 ↪0.68113072, 0.68077205, 0.68077527,
                   0.69357459, 0.69372878, 0.69341197, 0.69277492, 0.70191902,␣
 ↪0.70235819, 0.70424156, 0.70644806]

mean_train_score = [0.66895523, 0.66895523, 0.66895523, 0.66895523, 0.73153556,␣
 ↪0.73019983, 0.72432353, 0.71632896,
                    0.82714781, 0.82142404, 0.7903215 , 0.75914939, 0.9999747 ,␣
 ↪0.99989934, 0.99099531, 0.91503008]

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}]
```

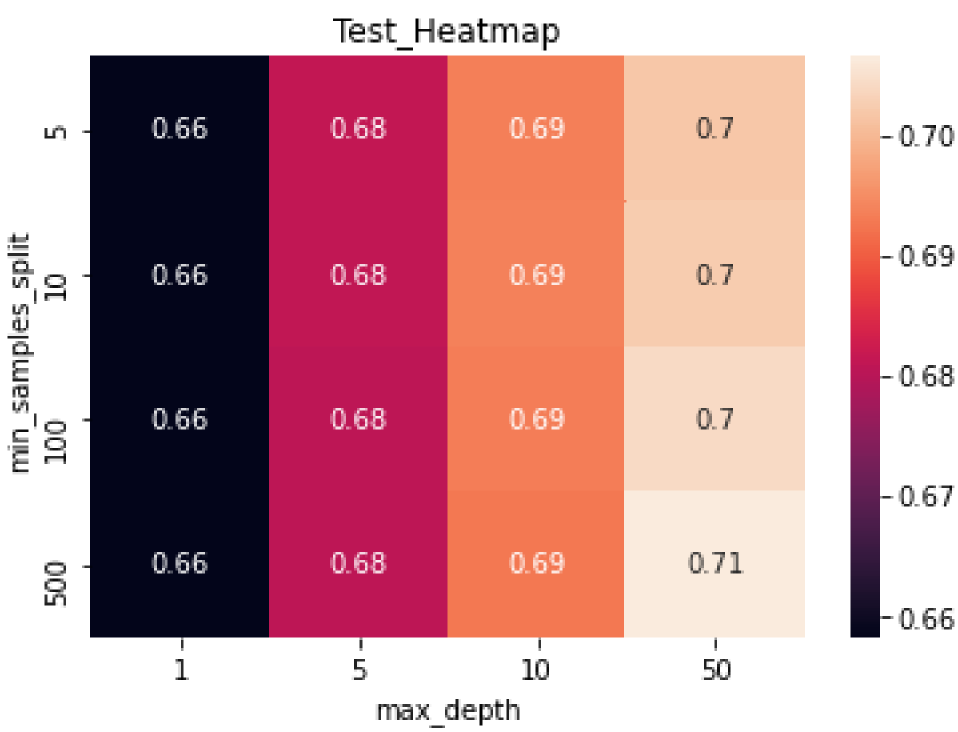
[16]:

```
max_depths = []
min_samples_split = []

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

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')
```

[17]:

```
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
0.67
0.73
0.83
1
0.67
0.73
0.82
1
0.67
0.72
0.79
0.99
0.67
0.72
0.76
0.92
5
10
100
500
min_samples_split
1
5
10
50
max_depth
0.95
0.90
0.85
0.80
0.75
0.70


Test_Heatmap
0.66
0.68
0.69
0.7
0.66
0.68
0.69
0.7
0.66
0.68
0.69
0.7
0.66
0.68
0.69
0.71
5
10
100
500
min_samples_split
1
5
10
50
max_depth
0.70
0.69
0.68
0.67
0.66

1.7 1.7 Model Train & Predict

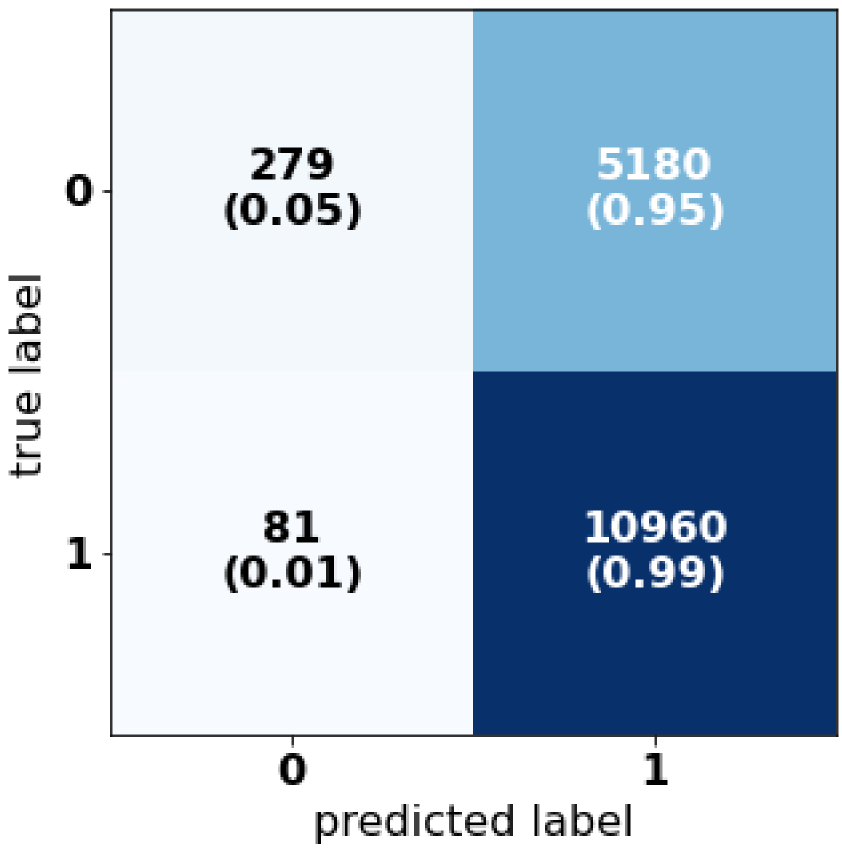
[9]:
```
X_train = np.column_stack((X_train, sentiment_anayser(X_train_essay)))
temp = tfidf.transform(X_train_essay).toarray()
X_train = np.column_stack((X_train, temp))

X_test = np.column_stack((X_test, sentiment_anayser(X_test_essay)))
temp = tfidf.transform(X_test_essay).toarray()
X_test = np.column_stack((X_test, temp))

X_train.shape, X_test.shape
```

[9]:
```
((33500, 5104), (16500, 5104))
```

[10]:
```
model = RandomForestClassifier(max_depth = 50, min_samples_split= 500,␣
 ↪random_state=2)
model = model.fit(X_train, ytrain)
Y_pred = model.predict(X_test)
```

1.8 1.8 ConfusionMatrix

[11]:
```
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);
```



1.9 1.9 AUC-Plot

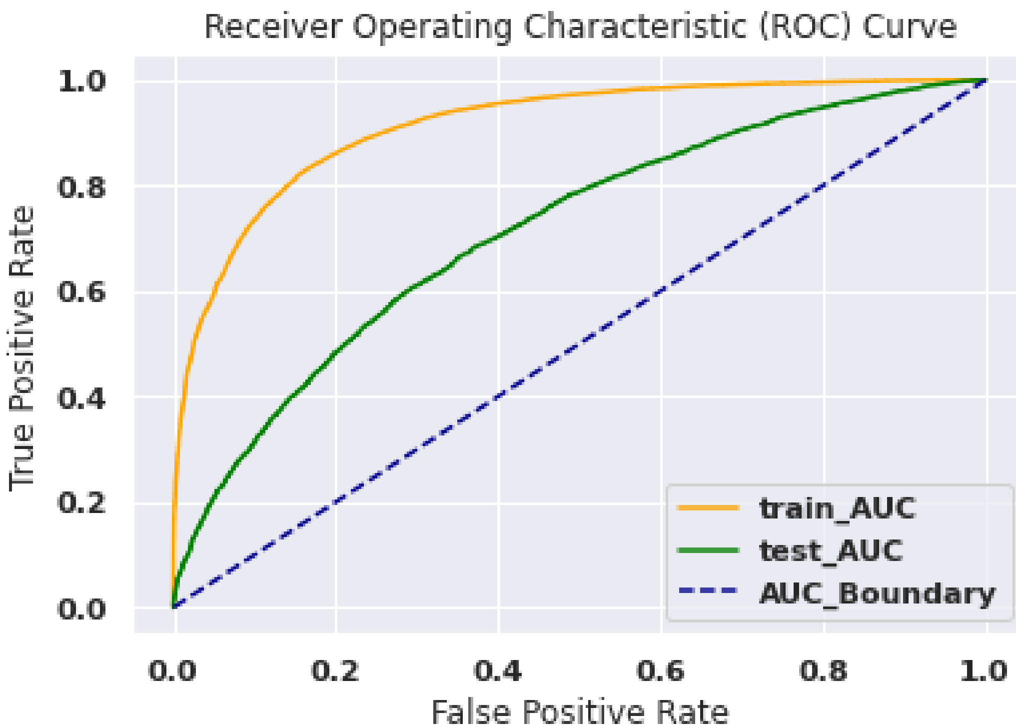
[12]:
```python
from sklearn.metrics import auc

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

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

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

probs = model.predict_proba(X_test)
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 :  0.5670076296286061

test_roc_auc_score :  0.5218859847919176

train_auc_score :  0.9133752024687409

test_auc_score :  0.713452576359503
```

[13]:
```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 :  0.9133752024687409
```

```
test_auc_score :  0.713452576359503
```



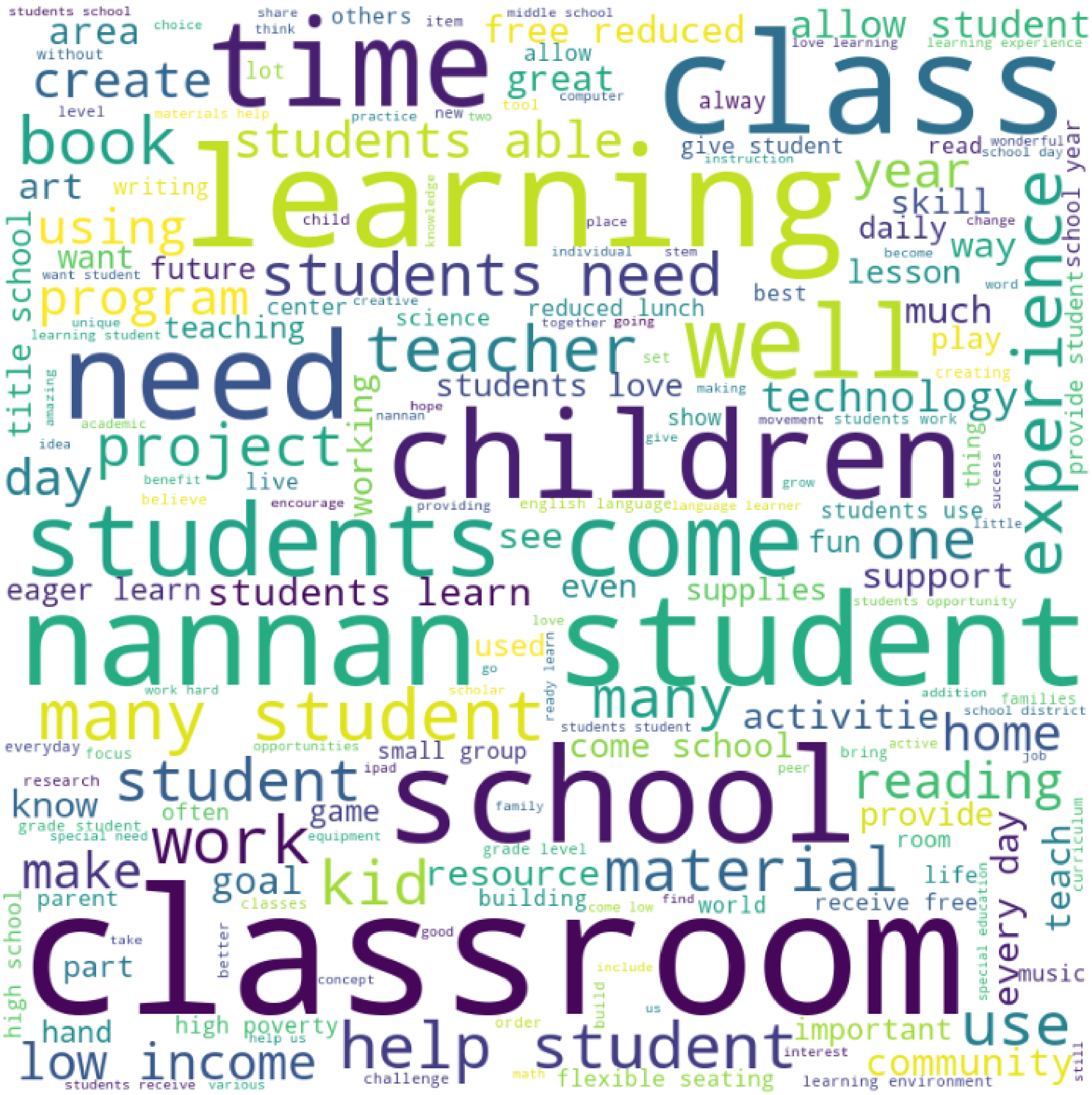
1.10 1.10 False-Positive Prediction WordCloud

[14]:
```
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)
```

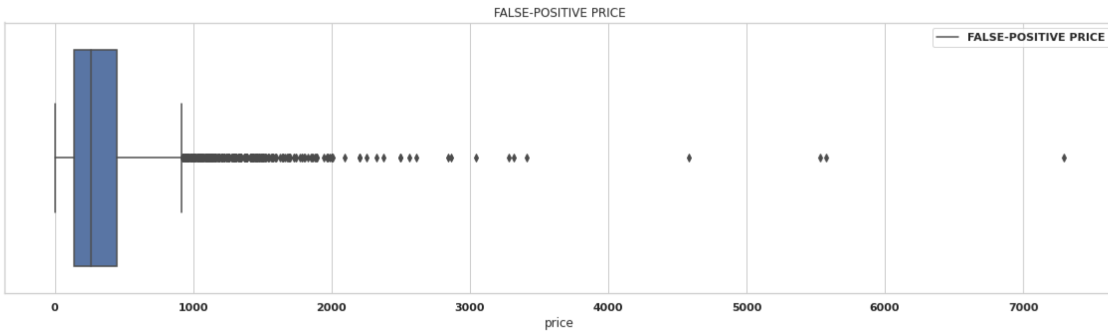
[15]:
```
data = pd.read_csv(preprocessed_data)
```

```
data = pd.concat([data.loc[data['project_is_approved'] == 0], data.
 ↪loc[data['project_is_approved'] == 1].head(33458)], sort=False)
data = data.drop(columns=['project_is_approved'])

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 = []
print('#'*50, '\n', 'WORDS IN ESSAYS - FOR FALSE-POSITIVE PREDICTIONS', '\n',␣
 ↪'#'*50)
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()
```

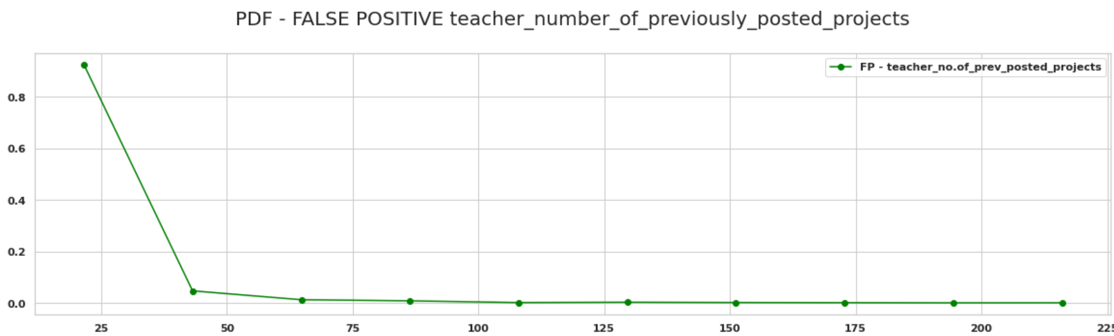
```
##################################################
 WORDS IN ESSAYS - FOR FALSE-POSITIVE PREDICTIONS
 ##################################################
```



[16]:
```python
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()
```



[17]:
```
sns.set_style("whitegrid");
plt.figure(figsize=(20,5))

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:

##DecisionTree Classifier

```
[18]: print(X_train.shape, ytrain.shape, X_test.shape, np.array(ytest).shape) #SET 1␣
      ↪- TFIDF
```

```
(33500, 5104) (33500,) (16500, 5104) (16500,)
```

2.1 2.1 Non-Zero Feature Importance Feature Selection

```
[19]: from sklearn.feature_selection import SelectFromModel
      from sklearn.tree import DecisionTreeClassifier

      model = DecisionTreeClassifier(random_state=2)
      model = model.fit(X_train, ytrain)
```

```
[20]: non_zero_Feature_importance_index = [index for index, value in enumerate(model.
      ↪feature_importances_.tolist()) if value != float(0.0)]
```

```
[21]: print(len(non_zero_Feature_importance_index))

      X = np.take(X_train, non_zero_Feature_importance_index, axis=1)
      X_ = np.take(X_test, non_zero_Feature_importance_index, axis=1)

      print(X.shape, X_.shape)
```

```
1992
(33500, 1992) (16500, 1992)
```

2.2 2.2 Hyper_Parameter Tuning

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

      parameters={'max_depth' : [1, 5, 10, 50, 'None'], 'min_samples_split' : [2, 5,␣
      ↪10, 100, 500]}

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

      grid_result = gsc.fit(X, 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 25 candidates, totalling 125 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:  2.0min
[Parallel(n_jobs=-1)]: Done 125 out of 125 | elapsed: 12.5min finished
```

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


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

0.6416584598675767

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

[23]:
```python
grid_result.cv_results_
```

[23]:
```
{'mean_fit_time': array([ 2.16149755,  2.03612494,  2.04942455,  2.06100979,
2.04422913,
         7.94701281,  7.89494824,  7.89863539,  7.8644557 ,  7.86954598,
        13.73311086, 13.7625011 , 13.75112782, 13.63686953, 13.24367671,
        35.84308181, 35.85975919, 35.53652792, 35.00844431, 28.83373485,
         0.36750345,  0.35105982,  0.34450002,  0.353444  ,  0.33779845]),
 'mean_score_time': array([0.05628324, 0.05030651, 0.05176606, 0.05159874,
0.05081816,
        0.0578064 , 0.06128101, 0.0604774 , 0.05833521, 0.05767779,
        0.05676041, 0.05874224, 0.05812917, 0.05730886, 0.0555511 ,
        0.0597394 , 0.05844855, 0.05873723, 0.06554189, 0.05359812,
        0.        , 0.        , 0.        , 0.        , 0.        ]),
 'mean_test_score': array([0.57006531, 0.57006531, 0.57006531, 0.57006531,
0.57006531,
        0.62769167, 0.62769167, 0.62755632, 0.62757599, 0.6275973 ,
        0.63199893, 0.63202057, 0.63048601, 0.63643141, 0.64165846,
        0.547114  , 0.54951562, 0.55442911, 0.59829398, 0.62486079,
               nan,        nan,        nan,        nan,        nan]),
 'mean_train_score': array([0.57243645, 0.57243645, 0.57243645, 0.57243645,
0.57243645,
        0.64887846, 0.64887846, 0.64881175, 0.64801474, 0.64680202,
        0.72913206, 0.72875749, 0.72711689, 0.71264765, 0.69491694,
        0.98970599, 0.9878969 , 0.98138288, 0.90940004, 0.80865177,
               nan,        nan,        nan,        nan,        nan]),
 'param_max_depth': masked_array(data=[1, 1, 1, 1, 1, 5, 5, 5, 5, 5, 10, 10, 10,
10, 10, 50,
                    50, 50, 50, 50, 'None', 'None', 'None', 'None', 'None'],
              mask=[False, False, False, False, False, False, False, False,
                    False, 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=[2, 5, 10, 100, 500, 2, 5, 10,
100, 500, 2, 5, 10, 100,
                    500, 2, 5, 10, 100, 500, 2, 5, 10, 100, 500],
              mask=[False, False, False, False, False, False, False, False,
                    False, 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': 2},
  {'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': 2},
  {'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': 2},
  {'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': 2},
  {'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},
  {'max_depth': 'None', 'min_samples_split': 2},
  {'max_depth': 'None', 'min_samples_split': 5},
  {'max_depth': 'None', 'min_samples_split': 10},
  {'max_depth': 'None', 'min_samples_split': 100},
  {'max_depth': 'None', 'min_samples_split': 500}],
 'rank_test_score': array([13, 13, 13, 13, 13,  6,  6, 10,  9,  8,  4,  3,  5,
2,  1, 20, 19,
        18, 12, 11, 21, 22, 23, 24, 25], dtype=int32),
 'split0_test_score': array([0.57026829, 0.57026829, 0.57026829, 0.57026829,
0.57026829,
        0.61881893, 0.61881893, 0.61881893, 0.61903384, 0.61947725,
        0.63079281, 0.62716116, 0.62527479, 0.63199091, 0.63958098,
        0.55717818, 0.55721738, 0.55338068, 0.60029121, 0.6132918 ,
               nan,        nan,        nan,        nan,        nan]),
 'split0_train_score': array([0.57267424, 0.57267424, 0.57267424, 0.57267424,
0.57267424,
```

```
        0.65180692, 0.65180692, 0.65180692, 0.65127501, 0.64995106,
        0.7358991 , 0.73549475, 0.73374695, 0.71721691, 0.69886138,
        0.99071143, 0.98705963, 0.98049805, 0.90417812, 0.81433624,
               nan,        nan,        nan,        nan,        nan]),
 'split1_test_score': array([0.56895013, 0.56895013, 0.56895013, 0.56895013,
0.56895013,
        0.62529945, 0.62529945, 0.62529945, 0.62558587, 0.62549207,
        0.6321784 , 0.63282304, 0.63160783, 0.6367018 , 0.64197874,
        0.55510628, 0.54911763, 0.55913173, 0.61154567, 0.62711366,
               nan,        nan,        nan,        nan,        nan]),
 'split1_train_score': array([0.5728859 , 0.5728859 , 0.5728859 , 0.5728859 ,
0.5728859 ,
        0.64896792, 0.64896792, 0.64896792, 0.6486381 , 0.64688276,
        0.72465015, 0.72440649, 0.72277644, 0.70908203, 0.6907956 ,
        0.98887642, 0.98611212, 0.97829248, 0.9071791 , 0.80567445,
               nan,        nan,        nan,        nan,        nan]),
 'split2_test_score': array([0.57582401, 0.57582401, 0.57582401, 0.57582401,
0.57582401,
        0.63605184, 0.63605184, 0.63604238, 0.63500428, 0.63444289,
        0.64602401, 0.64793505, 0.64638542, 0.65319654, 0.65466931,
        0.54337717, 0.55325536, 0.55589099, 0.59455457, 0.63469005,
               nan,        nan,        nan,        nan,        nan]),
 'split2_train_score': array([0.57064167, 0.57064167, 0.57064167, 0.57064167,
0.57064167,
        0.64572019, 0.64572019, 0.6454977 , 0.64407122, 0.64298684,
        0.7239699 , 0.72387573, 0.72184435, 0.70749677, 0.69157198,
        0.98883113, 0.98997835, 0.98103343, 0.91581872, 0.80793902,
               nan,        nan,        nan,        nan,        nan]),
 'split3_test_score': array([0.56492522, 0.56492522, 0.56492522, 0.56492522,
0.56492522,
        0.62074905, 0.62074905, 0.62052629, 0.62093107, 0.62105361,
        0.62353525, 0.62423106, 0.62214213, 0.62659492, 0.63261129,
        0.53849314, 0.54993399, 0.56162372, 0.59109405, 0.62470219,
               nan,        nan,        nan,        nan,        nan]),
 'split3_train_score': array([0.5738891 , 0.5738891 , 0.5738891 , 0.5738891 ,
0.5738891 ,
        0.6480325 , 0.6480325 , 0.647977  , 0.64739588, 0.64593075,
        0.73007518, 0.72986622, 0.72818947, 0.71684301, 0.69462388,
        0.99404764, 0.9919724 , 0.98880236, 0.91666142, 0.79709349,
               nan,        nan,        nan,        nan,        nan]),
 'split4_test_score': array([0.57035892, 0.57035892, 0.57035892, 0.57035892,
0.57035892,
        0.63753909, 0.63753909, 0.63709457, 0.63732488, 0.63752068,
        0.62746419, 0.62795253, 0.62701987, 0.63367288, 0.63945199,
        0.54141522, 0.53805375, 0.54211842, 0.59398438, 0.62450624,
               nan,        nan,        nan,        nan,        nan]),
 'split4_train_score': array([0.57209136, 0.57209136, 0.57209136, 0.57209136,
```

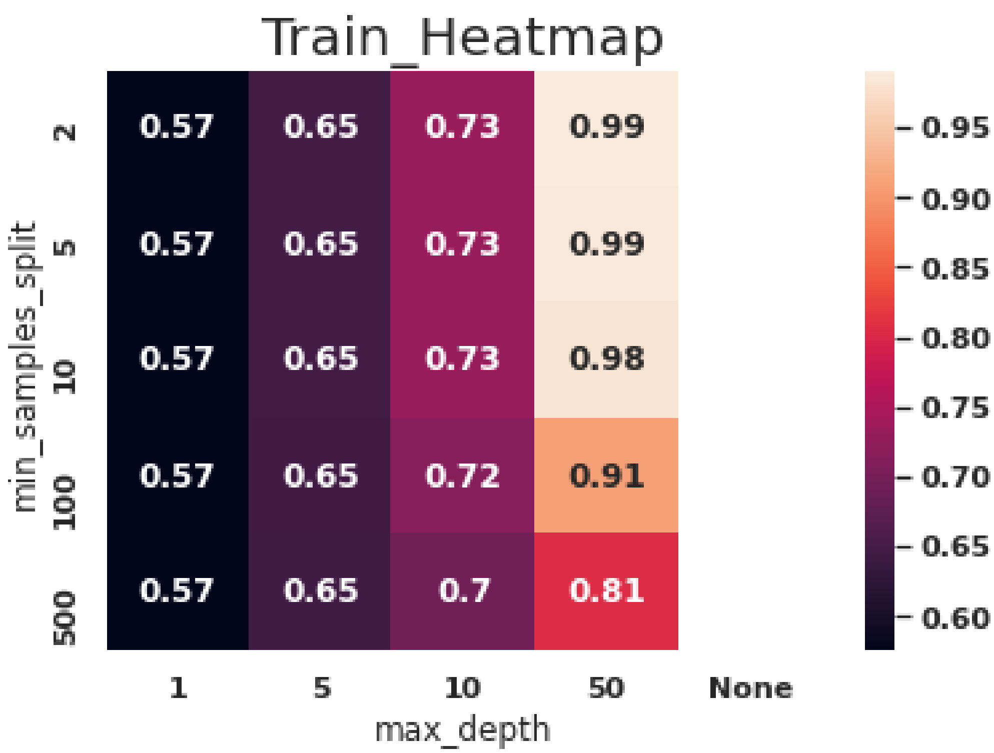
```
0.57209136,
        0.64986476, 0.64986476, 0.64980922, 0.64869347, 0.64825869,
        0.73106597, 0.73014425, 0.72902723, 0.71259951, 0.69873185,
        0.98606334, 0.98436201, 0.97828807, 0.90316286, 0.81821567,
               nan,        nan,        nan,        nan,        nan]),
 'std_fit_time': array([0.17098671, 0.0235819 , 0.02034959, 0.01752512,
0.01694943,
        0.04838963, 0.02643532, 0.06241013, 0.04127965, 0.04970722,
        0.06522703, 0.08155152, 0.08756446, 0.05615827, 0.12068145,
        1.17168742, 1.10436682, 0.99880352, 0.9239368 , 3.13328383,
        0.00483577, 0.00630063, 0.00919723, 0.0074248 , 0.01147554]),
 'std_score_time': array([0.0015313 , 0.00162219, 0.00151464, 0.00284609,
0.00151095,
        0.00036412, 0.00480184, 0.00514208, 0.00053691, 0.00222657,
        0.00303959, 0.00538408, 0.00471291, 0.00255839, 0.00231734,
        0.00328834, 0.00225498, 0.00169684, 0.00487301, 0.00974819,
        0.        , 0.        , 0.        , 0.        , 0.        ]),
 'std_test_score': array([0.00349154, 0.00349154, 0.00349154, 0.00349154,
0.00349154,
        0.00773967, 0.00773967, 0.00766614, 0.0073662 , 0.00719058,
        0.00762086, 0.00842323, 0.0085187 , 0.00900252, 0.00721693,
        0.00756206, 0.00640329, 0.00676264, 0.00726624, 0.0068656 ,
               nan,        nan,        nan,        nan,        nan]),
 'std_train_score': array([0.00106856, 0.00106856, 0.00106856, 0.00106856,
0.00106856,
        0.00201231, 0.00201231, 0.00208183, 0.0023415 , 0.00233921,
        0.00440812, 0.00427331, 0.00436799, 0.00394276, 0.00341679,
        0.00262995, 0.00273218, 0.0038751 , 0.00574509, 0.00730407,
               nan,        nan,        nan,        nan,        nan])}
```

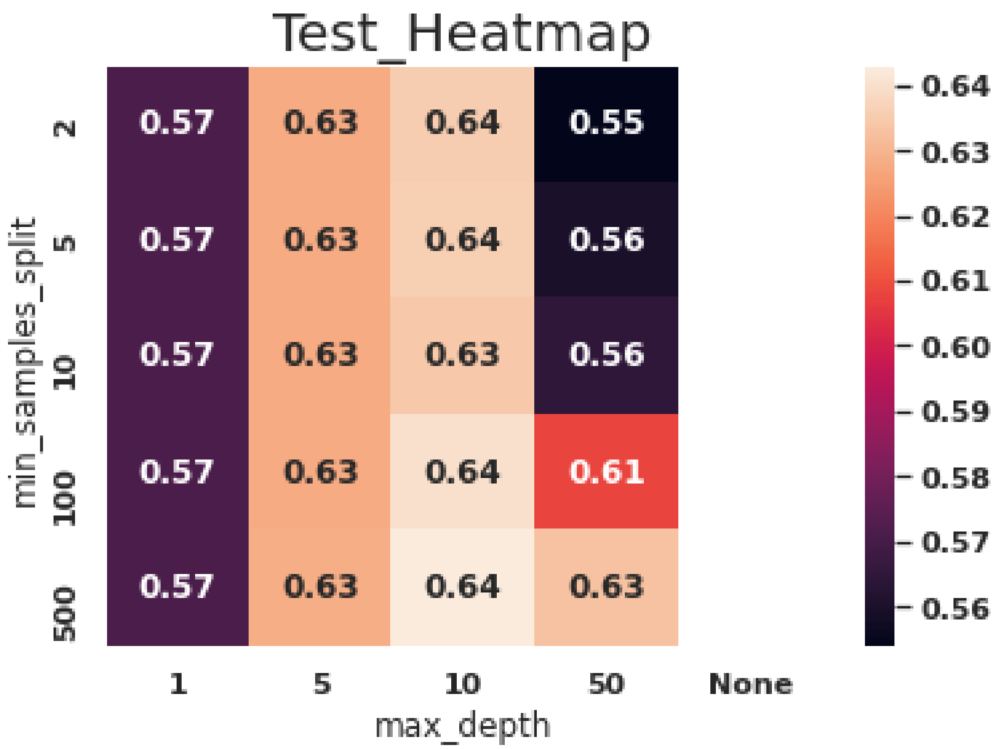
2.3 2.3 Cross-Validation Result

[24]:
```
params = [{'max_depth': 1, 'min_samples_split': 2},
  {'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': 2},
  {'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': 2},
  {'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': 2},
 {'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},
 {'max_depth': 'None', 'min_samples_split': 2},
 {'max_depth': 'None', 'min_samples_split': 5},
 {'max_depth': 'None', 'min_samples_split': 10},
 {'max_depth': 'None', 'min_samples_split': 100},
 {'max_depth': 'None', 'min_samples_split': 500}]

mean_test_score = np.array([0.57130322, 0.57130322, 0.57130322, 0.57130322, 0.
 ↪57130322,
                            0.62814354, 0.62814354, 0.62814354, 0.6280679 , 0.
 ↪62862218,
                            0.63576448, 0.63606226, 0.63465875, 0.64017402, 0.
 ↪64283563,
                            0.55380708, 0.56002823, 0.56476647, 0.60818671, 0.
 ↪63312122,
                            np.nan,     np.nan,     np.nan,     np.
 ↪nan,     np.nan])
mean_train_score = np.array([0.57493185, 0.57493185, 0.57493185, 0.57493185, 0.
 ↪57493185,
                             0.64836227, 0.64836227, 0.64834003, 0.64771022, 0.
 ↪64598223,
                             0.73365531, 0.73309553, 0.73175414, 0.71574084, 0.
 ↪69606365,
                             0.99068627, 0.98902076, 0.98316671, 0.91041784, 0.
 ↪80675576,
                             np.nan,     np.nan,     np.nan,     np.
 ↪nan,     np.nan])

mean_train_score = pd.Series(mean_train_score)
mean_test_score = pd.Series(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()
```



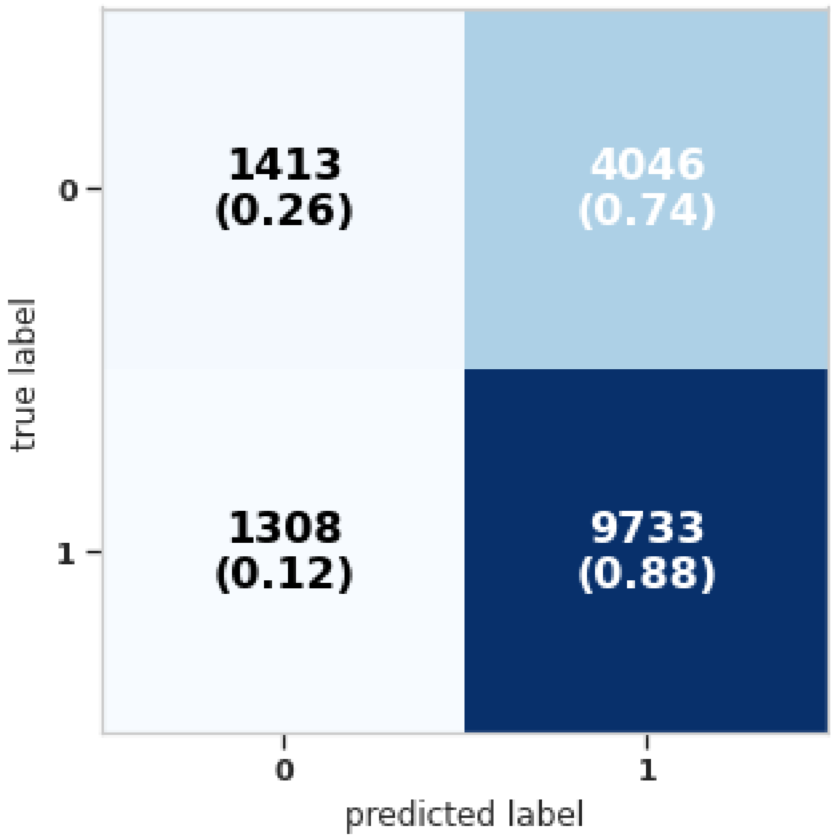


2.4 2.4 Model Train

[25]:
```
model = DecisionTreeClassifier(random_state=2, max_depth=10,␣
 ↪min_samples_split=500)
model.fit(X, ytrain)
Y_pred = model.predict(X_)
```

2.5 2.5 ConfusionMatrix

[26]:
```
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);
```



2.6 2.6 AUC-Plot

[27]:
```
from sklearn.metrics import auc

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

probs = model.predict_proba(X)
probs = probs[:, 1]
train_fpr, train_tpr, train_thresholds = roc_curve(ytrain, probs)

probs = model.predict_proba(X_)
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')
```

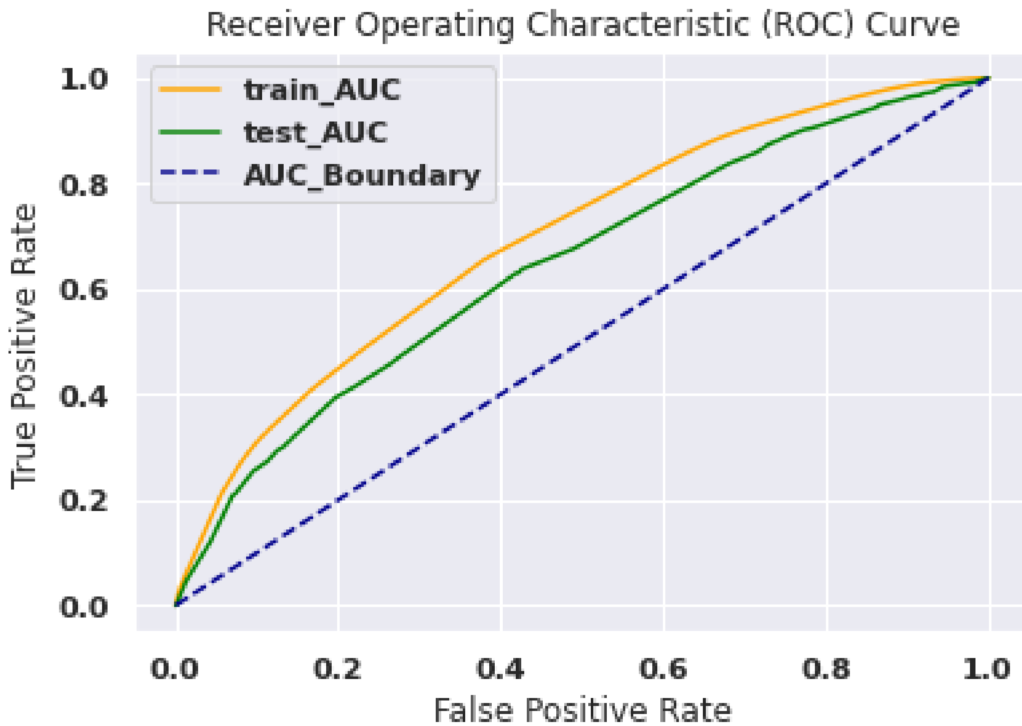
```
train_roc_auc_score :  0.6014207619005844

test_roc_auc_score :  0.5701855425079753
```

```
train_auc_score :  0.6959339045526182

test_auc_score :  0.6463583111982867
```

[28]:

```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 :  0.6959339045526182


test_auc_score :  0.6463583111982867
```



3 RESULT

---

[29]:
```
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 :␣
↪100', 0.93, 0.68])
x.add_row(["TFIDF", 'RandomForest', 'max_depth : 50, min_samples_split : 500',␣
↪0.92, 0.71])
x.add_row(["TFIDF", 'DecisionTree', 'max_depth : 10, min_samples_split : 500',␣
↪0.69, 0.64])

print(x)
```

```
+------------+--------------+------------------------------------------+---------
--+----------+
| Vectorizer |    Model     |             Hyper_Parameter              |
Train_AUC | Test_AUC |
+------------+--------------+------------------------------------------+---------
```

```
--+----------+
| TFIDF-W2V  | RandomForest | max_depth : 50, min_samples_split : 100 |    0.93
|   0.68   |
|   TFIDF    | RandomForest | max_depth : 50, min_samples_split : 500 |    0.92
|   0.71   |
|   TFIDF    | DecisionTree | max_depth : 10, min_samples_split : 500 |    0.69
|   0.64   |
+------------+--------------+-----------------------------------------+---------
--+----------+
```