GBDT_Assignment_v1

December 1, 2020

1 1. GBDT - Assignment (LightGBDT)

[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).
```

[2]:
```
!pip install pandas==1.1.3
!pip install -U lightgbm --install-option=--gpu
```

```
Requirement already satisfied: pandas==1.1.3 in /usr/local/lib/python3.6/dist-
packages (1.1.3)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-
packages (from pandas==1.1.3) (2018.9)
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: six>=1.5 in /usr/local/lib/python3.6/dist-
packages (from python-dateutil>=2.7.3->pandas==1.1.3) (1.15.0)
/usr/local/lib/python3.6/dist-packages/pip/_internal/commands/install.py:283:
UserWarning: Disabling all use of wheels due to the use of --build-options /
--global-options / --install-options.
  cmdoptions.check_install_build_global(options)
Requirement already up-to-date: lightgbm in /usr/local/lib/python3.6/dist-
packages (3.1.0)
Requirement already satisfied, skipping upgrade: numpy in
/usr/local/lib/python3.6/dist-packages (from lightgbm) (1.18.5)
Requirement already satisfied, skipping upgrade: scipy in
/usr/local/lib/python3.6/dist-packages (from lightgbm) (1.4.1)
Requirement already satisfied, skipping upgrade: scikit-learn!=0.22.0 in
/usr/local/lib/python3.6/dist-packages (from lightgbm) (0.22.2.post1)
Requirement already satisfied, skipping upgrade: joblib>=0.11 in
/usr/local/lib/python3.6/dist-packages (from scikit-learn!=0.22.0->lightgbm)
(0.17.0)
```

[3]:
```
import numpy as np
import cupy as cp
import pandas as pd
pd.set_option('display.width', 5)
pd.set_option('display.max_colwidth', 5)

import lightgbm as lgb
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import auc, roc_auc_score, confusion_matrix
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪RandomizedSearchCV

import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.simplefilter("ignore")

import nltk
nltk.download('vader_lexicon')

from nltk.sentiment.vader import SentimentIntensityAnalyzer

from gc import collect

data_path = '/content/drive/MyDrive/6_Donors_choose_NB/preprocessed_data.csv'
w2v_path = '/content/drive/MyDrive/6_Donors_choose_NB/glove_vectors'
train_path = '/content/drive/MyDrive/6_Donors_choose_NB/train_data.csv'
```

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

1.1 1.1 Read Dataset:

[4]:
```
data = pd.read_csv(data_path, nrows=100000)

data['project_title'] = pd.read_csv(train_path, nrows=100000)['project_title']

print(list(data.columns))
```

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

1.1.1 1.1.1 Project-Title Cleaning

[5]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

[6]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',␣
 ↪"you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',␣
 ↪'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its',␣
 ↪'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this',␣
 ↪'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',␣
 ↪'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',␣
 ↪'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',␣
 ↪'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',␣
 ↪'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how',␣
 ↪'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so',␣
 ↪'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should',␣
 ↪"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',␣
 ↪"didn't", 'doesn', "doesn't", 'hadn',\
```

```
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't",␣
↪'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',␣
↪"shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

[7]:
```
print("printing some random reviews")
print(9, data['project_title'].values[9])
print(34, data['project_title'].values[34])
print(147, data['project_title'].values[147])
```

```
printing some random reviews
9 Just For the Love of Reading--\r\nPure Pleasure
34 \"Have A Ball!!!\"
147 Who needs a Chromebook?\r\nWE DO!!
```

[8]:
```
# Combining all the above stundents
from tqdm import tqdm

def preprocess_text(text_data):
    preprocessed_text = []
    # tqdm is for printing the status bar
    for sentance in tqdm(text_data):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\n', ' ')
        sent = sent.replace('\\"', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
        preprocessed_text.append(sent.lower().strip())
    return preprocessed_text
```

[9]:
```
data['project_title'] = preprocess_text(data['project_title'].values)
```

```
100%|      | 100000/100000 [00:02<00:00, 45307.41it/s]
```

1.1.2 1.1.2 Data-Splitting

[10]:
```
xtrain, xtest, ytrain, ytest = train_test_split(data,␣
↪data['project_is_approved'],
                                                test_size=0.40,
                                               ␣
↪stratify=data['project_is_approved'])

print("\nData-Shape : ", xtrain.shape, xtest.shape, ytrain.shape, ytest.shape,␣
↪"\n")
```

```
del data
collect()

categories = ['school_state', 'teacher_prefix', 'project_grade_category',␣
↪'clean_categories', 'clean_subcategories']
numeric = ['teacher_number_of_previously_posted_projects', 'price']
text = ['essay']
```

```
Data-Shape :  (60000, 10) (40000, 10) (60000,) (40000,)
```

1.2 1.2 Feature-Preprocessing

1.2.1 1.2.1 Responsive-Coding

[11]:
```
def getResponseCoding(df, cat):
    df= pd.crosstab(index=df[str(cat)], columns=df['project_is_approved']).
↪reset_index()
    df['total'] = df.sum(axis=1)
    df['neg_prob'] = df[0]/df['total']
    df['pos_prob'] = df[1]/df['total']
    return df

def getTestResponseCoding(df, cat, encode_df):
    negative_prob = []
    positive_prob = []
    for i in encode_df[str(cat)]:
        try:
            negative_prob.append(df[df[str(cat)] == str(i)].neg_prob.values[0])
            positive_prob.append(df[df[str(cat)] == str(i)].pos_prob.values[0])
        except:
            negative_prob.append(0.5)
            positive_prob.append(0.5)
    return np.asarray(negative_prob), np.asarray(positive_prob)
```

[12]:
```
for cat in categories:
    df= getResponseCoding(xtrain, cat)
    xtrain[str(cat) + '_0_prob'], xtrain[str(cat) + '_1_prob'] =␣
↪getTestResponseCoding(df, cat, xtrain)
    xtest[str(cat) + '_0_prob'], xtest[str(cat) + '_1_prob'] =␣
↪getTestResponseCoding(df, cat, xtest)

    xtrain=xtrain.drop(columns=[cat])
    xtest=xtest.drop(columns=[cat])

    collect()
```

1.2.2 1.2.2 TFIDF Vectorizer

[13]:
```
def getFitTFIDF_Vectorizer(preprocessed_data):
    vectorizer = TfidfVectorizer(ngram_range=(2,2), max_features=5000,␣
 ↪min_df=10)
    vectorizer.fit(preprocessed_data)
    return vectorizer

def getTFIDFVectorizeTxtData(preprocessed_data, vectorizer):
    text_tfidf = vectorizer.transform(preprocessed_data)
    print("Shape of matrix after tfidf encodig ",text_tfidf.shape)
    return text_tfidf
```

1.2.3 1.2.3 TFIDF weighted W2V

[35]:
```
import pickle

with open(w2v_path, 'rb') as f:
  model = pickle.load(f)
  glove_words =  set(model.keys())
```

[36]:
```
def TFIDF_W2V(preprocessed_data):
    tfidf_model = TfidfVectorizer()
    tfidf_model.fit(preprocessed_data)
    return tfidf_model

def getTFIDF_W2V(preprocessed_data, tfidf_model):

    # we are converting a dictionary with word as a key, and the idf as a value
    dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.
 ↪idf_)))
    tfidf_words = set(tfidf_model.get_feature_names())

    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in␣
 ↪this list
    for sentence in tqdm(preprocessed_data): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence/
 ↪review
        for word in sentence.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words):
                vec = model[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and the␣
 ↪tf value((sentence.count(word)/len(sentence.split())))
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.
 ↪split())) # getting the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
```

```
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)

    print(len(tfidf_w2v_vectors))
    print(len(tfidf_w2v_vectors[0]))

    return np.asarray(tfidf_w2v_vectors)
```

1.2.4 1.2.4 Standardize Numerical Value

[16]:
```
for num in numeric:
    scaler = StandardScaler()

    scaler = scaler.fit(xtrain[num].values.reshape(-1,1))

    xtrain["std_" +str(num)] = scaler.transform(xtrain[num].values.
 ↪reshape(-1,1))
    xtest["std_" +str(num)]= scaler.transform(xtest[num].values.reshape(-1,1))

    xtrain=xtrain.drop(columns=[num])
    xtest=xtest.drop(columns=[num])

    collect()

xtrain = xtrain.drop(columns=['project_is_approved'])
xtest = xtest.drop(columns=['project_is_approved'])
```

[17]:
```
xtest, xcv, ytest, ycv = train_test_split(xtest, ytest, test_size=0.50,
                                          stratify=ytest)

X_train_essay = xtrain['essay']
X_test_essay = xtest['essay']
X_cv_essay = xcv['essay']
X_train_title = xtrain['project_title']
X_test_title = xtest['project_title']
X_cv_title = xcv['project_title']

xtrain = xtrain.drop(columns=['essay', 'project_title'])
xtest = xtest.drop(columns=['essay', 'project_title'])
xcv = xcv.drop(columns=['essay', 'project_title'])
```

1.3 1.3. Sentiment Intensity Analyser

[18]:
```
def sentiment_anayser(essay):

    sid = SentimentIntensityAnalyzer()
    negative = []
    positive = []
    neutral = []
    compound = []

    for sentence in essay:
        ss = sid.polarity_scores(sentence)
        sentmnt = list(ss.values())
        neg = sentmnt[0]
        neu = sentmnt[1]
        pos = sentmnt[2]
        cmpd = sentmnt[3]

        negative.append(neg)
        neutral.append(neu)
        positive.append(pos)
        compound.append(cmpd)

    return np.column_stack((np.array(negative), np.array(neutral),
                            np.array(positive), np.array(compound)))
```

1.4 1.4 Hyper-Parameter Tuning

[ ]:
```
sets = ['tfidf', 'w2v']
scorer = dict()


for set_ in sets:

    if set_ == 'tfidf':
        vectorizer = getFitTFIDF_Vectorizer(X_train_title)
        temp_1 = getTFIDFVectorizeTxtData(X_train_title, vectorizer).toarray()
        temp_2 = getTFIDFVectorizeTxtData(X_cv_title, vectorizer).toarray()

        vectorizer = getFitTFIDF_Vectorizer(X_train_essay)

        temp_1 = np.column_stack(
            (temp_1, getTFIDFVectorizeTxtData(X_train_essay, vectorizer).
 ↪toarray()))

        temp_2 = np.column_stack(
```

```
            (temp_2, getTFIDFVectorizeTxtData(X_cv_essay, vectorizer).
↪toarray()))


        temp_1 = np.column_stack((temp_1, sentiment_anayser(X_train_essay),
                                  xtrain.values))
        print("Final_Train_Matrix : ", temp_1.shape)

        temp_2 = np.column_stack((temp_2, sentiment_anayser(X_cv_essay),
                                  xcv.values))
        print("Final_CV_Matrix : ", temp_2.shape)

    elif set_ == 'w2v':
        vectorizer1 = TFIDF_W2V(X_train_title)
        temp_1 = getTFIDF_W2V(X_train_title, vectorizer1)

        vectorizer2 = TFIDF_W2V(X_train_essay)
        temp_2 = getTFIDF_W2V(X_train_essay, vectorizer2)

        temp_1 = np.column_stack((temp_1, temp_2,
                                  sentiment_anayser(X_train_essay)))
        temp_1 = np.column_stack((temp_1, xtrain.values))
        print("Final_Train_Matrix : ", temp_1.shape)

        essay_ = getTFIDF_W2V(X_cv_essay, vectorizer2)
        temp_2 = getTFIDF_W2V(X_cv_title, vectorizer1)
        temp_2 = np.column_stack((temp_2, essay_,␣
↪sentiment_anayser(X_cv_essay)))
        temp_2 = np.column_stack((temp_2, xcv.values))
        print("Final_CV_Matrix : ", temp_2.shape)

    parameters={"early_stopping_rounds": 10,
                "eval_metric" : 'auc',
                "eval_set" : [(temp_2,ycv)],
                'eval_names': ['valid'],
                'verbose': False}

    parameter_tuning ={
             'max_depth':  [5, 10, 50, 100, 500],
             'min_child_samples': [5, 10, 100, 500]}

    classifier = lgb.LGBMClassifier(device="gpu", random_state=300,␣
↪silent=True, metric='auc', n_jobs=-1)

    find_parameters = RandomizedSearchCV(
        estimator=classifier, param_distributions=parameter_tuning,
        n_iter=100,
```

```
        scoring='roc_auc',
        cv=5,
        refit=True,
        random_state=300,
        verbose=False,
        return_train_score=True)

    scorer[set_] = find_parameters

    find_parameters.fit(temp_1, ytrain, **parameters)
    print("\n#"*50, set_, " : \n#"*50)
    print('Best score : {} with parameters: {} '.format(find_parameters.
↪best_score_, find_parameters.best_params_), "\n\n")

    del temp_1
    del temp_2
    collect()
```

[20]:
```
scorer['tfidf'].cv_results_
```

[20]:
```
{'mean_fit_time': array([29.37143335, 31.39018006, 27.21542997,  8.12709513,
41.43432875,
        40.39781775, 31.45694046,  9.65661631, 44.139961  , 42.00484724,
        36.59567299, 10.57228518, 43.09559617, 41.77202759, 36.69225626,
        10.745749  , 44.02288251, 41.73093138, 36.49613132, 10.53993392]),
 'mean_score_time': array([0.23542137, 0.24028544, 0.23931074, 0.24317222,
0.25397229,
        0.25247755, 0.25618491, 0.2577342 , 0.25270042, 0.25672455,
        0.26740789, 0.24944429, 0.2450706 , 0.24963655, 0.26820102,
        0.2621449 , 0.25417643, 0.25424833, 0.2626502 , 0.25620427]),
 'mean_test_score': array([0.68654277, 0.68751047, 0.69218654, 0.68715998,
0.68887255,
        0.6892439 , 0.69749857, 0.69014198, 0.69072152, 0.69357233,
        0.70046676, 0.6918717 , 0.69072666, 0.69357233, 0.70046936,
        0.69187132, 0.69072417, 0.69357232, 0.70046815, 0.69187171]),
 'mean_train_score': array([0.81392229, 0.80250993, 0.76146983, 0.73273681,
0.87682745,
        0.86966328, 0.80408965, 0.76242611, 0.88357583, 0.85831355,
        0.82010738, 0.77593928, 0.88357583, 0.85831355, 0.82010737,
        0.77593928, 0.88357583, 0.85831354, 0.82010737, 0.77593928]),
 'param_max_depth': masked_array(data=[5, 5, 5, 5, 10, 10, 10, 10, 50, 50, 50,
50, 100, 100,
                   100, 100, 500, 500, 500, 500],
             mask=[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_child_samples': masked_array(data=[5, 10, 100, 500, 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,
                    False, False, False, False],
       fill_value='?',
            dtype=object),
 'params': [{'max_depth': 5, 'min_child_samples': 5},
  {'max_depth': 5, 'min_child_samples': 10},
  {'max_depth': 5, 'min_child_samples': 100},
  {'max_depth': 5, 'min_child_samples': 500},
  {'max_depth': 10, 'min_child_samples': 5},
  {'max_depth': 10, 'min_child_samples': 10},
  {'max_depth': 10, 'min_child_samples': 100},
  {'max_depth': 10, 'min_child_samples': 500},
  {'max_depth': 50, 'min_child_samples': 5},
  {'max_depth': 50, 'min_child_samples': 10},
  {'max_depth': 50, 'min_child_samples': 100},
  {'max_depth': 50, 'min_child_samples': 500},
  {'max_depth': 100, 'min_child_samples': 5},
  {'max_depth': 100, 'min_child_samples': 10},
  {'max_depth': 100, 'min_child_samples': 100},
  {'max_depth': 100, 'min_child_samples': 500},
  {'max_depth': 500, 'min_child_samples': 5},
  {'max_depth': 500, 'min_child_samples': 10},
  {'max_depth': 500, 'min_child_samples': 100},
  {'max_depth': 500, 'min_child_samples': 500}],
 'rank_test_score': array([20, 18,  8, 19, 17, 16,  4, 15, 14,  6,  3, 10, 12,
5,  1, 11, 13,
         7,  2,  9], dtype=int32),
 'split0_test_score': array([0.68935554, 0.69109645, 0.69548947, 0.69154586,
0.69161979,
        0.69235722, 0.70433714, 0.69557971, 0.69366508, 0.69557146,
        0.70686314, 0.69708845, 0.69366508, 0.69557146, 0.7068742 ,
        0.69708645, 0.69366179, 0.69557141, 0.70686314, 0.69708845]),
 'split0_train_score': array([0.81400249, 0.80036466, 0.75969163, 0.73051334,
0.87109281,
        0.86263119, 0.80801514, 0.76116202, 0.88720719, 0.82635821,
        0.82691397, 0.78053277, 0.88720719, 0.8263582 , 0.82691397,
        0.78053277, 0.88720719, 0.8263582 , 0.82691396, 0.78053276]),
 'split1_test_score': array([0.68848266, 0.69278274, 0.69509169, 0.69140342,
0.69673079,
        0.69275208, 0.70049397, 0.69503378, 0.69843819, 0.70077896,
        0.70287839, 0.69654966, 0.69843814, 0.70077901, 0.70287834,
        0.69654966, 0.69843819, 0.70077901, 0.70287839, 0.69654966]),
```

```
 'split1_train_score': array([0.80475168, 0.80151925, 0.75877813, 0.7319349 ,
0.87108552,
       0.87046027, 0.80247099, 0.76311888, 0.88622569, 0.87680974,
       0.81091563, 0.77528005, 0.88622569, 0.87680974, 0.81091563,
       0.77528005, 0.88622569, 0.87680974, 0.81091564, 0.77528006]),
 'split2_test_score': array([0.69279064, 0.69056254, 0.69597886, 0.68939774,
0.69303653,
       0.69516616, 0.70004836, 0.69322382, 0.69302804, 0.6955981 ,
       0.70493924, 0.69472845, 0.69305371, 0.6955981 , 0.70494123,
       0.69472845, 0.69305371, 0.6955981 , 0.70494123, 0.69472845]),
 'split2_train_score': array([0.81742124, 0.80196067, 0.76147645, 0.73157261,
0.89267687,
       0.86584396, 0.80307133, 0.76100915, 0.89314735, 0.86959977,
       0.83097876, 0.76952648, 0.89314734, 0.86959977, 0.83097875,
       0.76952649, 0.89314735, 0.86959977, 0.83097875, 0.76952649]),
 'split3_test_score': array([0.67631858, 0.67662477, 0.68156094, 0.67651986,
0.67696244,
       0.67895341, 0.68753919, 0.6765363 , 0.67827775, 0.68184106,
       0.68784138, 0.67888732, 0.67827775, 0.68184106, 0.68784138,
       0.67888737, 0.67826848, 0.68184106, 0.68784635, 0.67888737]),
 'split3_train_score': array([0.81847282, 0.80383785, 0.76504282, 0.73580779,
0.88565522,
       0.87600982, 0.80889629, 0.76549331, 0.86298752, 0.85824446,
       0.82848387, 0.77673336, 0.86298751, 0.85824445, 0.82848387,
       0.77673335, 0.86298751, 0.85824446, 0.82848388, 0.77673335]),
 'split4_test_score': array([0.68576641, 0.68648588, 0.69281175, 0.68693302,
0.68601317,
       0.68699061, 0.69507417, 0.69033627, 0.69019855, 0.69407208,
       0.69981163, 0.69210463, 0.6901986 , 0.69407203, 0.69981163,
       0.69210463, 0.69019866, 0.69407203, 0.69981163, 0.69210463]),
 'split4_train_score': array([0.81496323, 0.80486725, 0.76236011, 0.73385543,
0.86362685,
       0.87337115, 0.7979945 , 0.76134718, 0.8883114 , 0.86055555,
       0.80324465, 0.77762374, 0.8883114 , 0.86055556, 0.80324465,
       0.77762373, 0.8883114 , 0.86055555, 0.80324465, 0.77762374]),
 'std_fit_time': array([1.37232884, 0.69561243, 0.23333715, 0.14462753,
0.95892894,
       0.31346547, 0.25882313, 0.13633177, 1.3982609 , 2.33379904,
       2.14613387, 0.29072083, 1.41687467, 2.32868141, 2.19581661,
       0.30192438, 1.22442003, 2.25050509, 2.07906657, 0.22071774]),
 'std_score_time': array([0.00437617, 0.00818969, 0.00686208, 0.00740596,
0.00730339,
       0.00625123, 0.00734555, 0.01442844, 0.01213681, 0.01261423,
       0.01043443, 0.00307986, 0.00468086, 0.01451606, 0.02020671,
       0.00573234, 0.00632294, 0.01367146, 0.01663597, 0.00755575]),
 'std_test_score': array([0.00558296, 0.00582271, 0.00542264, 0.00557699,
0.00687989,
```

```
        0.00579732, 0.00578458, 0.00704484, 0.00676243, 0.0062914 ,
        0.00673145, 0.00672171, 0.00676418, 0.00629141, 0.00673381,
        0.00672138, 0.00676732, 0.0062914 , 0.00672985, 0.00672169]),
 'std_train_score': array([0.00486087, 0.00162516, 0.00218934, 0.00187747,
0.01067001,
        0.00487236, 0.00398299, 0.00171216, 0.01056529, 0.01729545,
        0.01098656, 0.00363667, 0.01056529, 0.01729545, 0.01098656,
        0.00363666, 0.01056529, 0.01729545, 0.01098655, 0.00363666])}
```

[21]:
```
scorer['w2v'].cv_results_
```

[21]:
```
{'mean_fit_time': array([12.45913095, 11.45425334, 11.63125157, 11.0970192 ,
14.04449425,
        13.53438344, 13.66393027, 12.85403032, 14.68655062, 14.40972233,
        13.12716689, 13.6660224 , 15.25874944, 15.00568595, 13.42132525,
        13.76135445, 14.84979615, 14.34800215, 12.93694773, 13.40485387]),
 'mean_score_time': array([0.07960014, 0.06964822, 0.07919188, 0.08095231,
0.07683854,
        0.06996531, 0.07865176, 0.08247004, 0.07735615, 0.07423143,
        0.07247477, 0.08878732, 0.078967  , 0.07787123, 0.07361207,
        0.0891757 , 0.07813673, 0.07357512, 0.0693224 , 0.09025116]),
 'mean_test_score': array([0.71459794, 0.71347811, 0.71464447, 0.71745185,
0.71322064,
        0.71396769, 0.71479425, 0.71657016, 0.71295019, 0.71380615,
        0.71497047, 0.7171578 , 0.71296159, 0.71381891, 0.71495712,
        0.71715777, 0.71294678, 0.71376658, 0.71495382, 0.71715677]),
 'mean_train_score': array([0.87269034, 0.85131101, 0.85777606, 0.83358297,
0.87824685,
        0.86727049, 0.86540386, 0.85122194, 0.88587194, 0.87859269,
        0.8492888 , 0.86344289, 0.88587194, 0.87859269, 0.84928881,
        0.86344289, 0.88587194, 0.87913286, 0.8492888 , 0.86344289]),
 'param_max_depth': masked_array(data=[5, 5, 5, 5, 10, 10, 10, 10, 50, 50, 50,
50, 100, 100,
                   100, 100, 500, 500, 500, 500],
             mask=[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_child_samples': masked_array(data=[5, 10, 100, 500, 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,
                   False, False, False, False],
        fill_value='?',
             dtype=object),
```

```
 'params': [{'max_depth': 5, 'min_child_samples': 5},
  {'max_depth': 5, 'min_child_samples': 10},
  {'max_depth': 5, 'min_child_samples': 100},
  {'max_depth': 5, 'min_child_samples': 500},
  {'max_depth': 10, 'min_child_samples': 5},
  {'max_depth': 10, 'min_child_samples': 10},
  {'max_depth': 10, 'min_child_samples': 100},
  {'max_depth': 10, 'min_child_samples': 500},
  {'max_depth': 50, 'min_child_samples': 5},
  {'max_depth': 50, 'min_child_samples': 10},
  {'max_depth': 50, 'min_child_samples': 100},
  {'max_depth': 50, 'min_child_samples': 500},
  {'max_depth': 100, 'min_child_samples': 5},
  {'max_depth': 100, 'min_child_samples': 10},
  {'max_depth': 100, 'min_child_samples': 100},
  {'max_depth': 100, 'min_child_samples': 500},
  {'max_depth': 500, 'min_child_samples': 5},
  {'max_depth': 500, 'min_child_samples': 10},
  {'max_depth': 500, 'min_child_samples': 100},
  {'max_depth': 500, 'min_child_samples': 500}],
 'rank_test_score': array([11, 16, 10,  1, 17, 12,  9,  5, 19, 14,  6,  2, 18,
13,  7,  3, 20,
        15,  8,  4], dtype=int32),
 'split0_test_score': array([0.72354505, 0.72437285, 0.7225037 , 0.72504373,
0.7228238 ,
        0.72209145, 0.72212429, 0.72250985, 0.72415262, 0.72322376,
        0.72128657, 0.72365845, 0.72419258, 0.72325623, 0.72128748,
        0.72365845, 0.72418967, 0.72324048, 0.72128748, 0.72365845]),
 'split0_train_score': array([0.87801533, 0.84079121, 0.8543511 , 0.83696262,
0.88223308,
        0.85594426, 0.87554345, 0.84686165, 0.90716663, 0.88349549,
        0.81988362, 0.85284242, 0.90716664, 0.88349548, 0.81988363,
        0.85284244, 0.90716663, 0.8834955 , 0.81988362, 0.85284243]),
 'split1_test_score': array([0.71432298, 0.71570113, 0.71713753, 0.72210676,
0.71708593,
        0.71383609, 0.71520718, 0.72255137, 0.71709989, 0.7179731 ,
        0.72101877, 0.72139032, 0.71710194, 0.71799364, 0.72095892,
        0.72139032, 0.71708312, 0.71798345, 0.72095897, 0.72139032]),
 'split1_train_score': array([0.84760596, 0.86524711, 0.84675442, 0.8208661 ,
0.88043284,
        0.8481538 , 0.8707058 , 0.84747798, 0.8724808 , 0.86966277,
        0.85786088, 0.84350694, 0.87248082, 0.86966279, 0.85786089,
        0.84350693, 0.87248081, 0.86966279, 0.85786087, 0.84350694]),
 'split2_test_score': array([0.71649135, 0.71213616, 0.71537893, 0.71633896,
0.70853576,
        0.71400622, 0.71642189, 0.71893787, 0.70907749, 0.70989316,
        0.71337735, 0.71737275, 0.70903602, 0.7098807 , 0.71337735,
```

```
        0.71737281, 0.70902637, 0.70988064, 0.71336058, 0.71737281]),
 'split2_train_score': array([0.87726167, 0.8223475 , 0.84762848, 0.83663605,
0.87238575,
        0.87482399, 0.84842494, 0.86714937, 0.87240842, 0.87269909,
        0.83271633, 0.87531877, 0.87240842, 0.87269909, 0.83271633,
        0.87531877, 0.87240842, 0.87269909, 0.83271633, 0.87531878]),
 'split3_test_score': array([0.70599795, 0.70308298, 0.70620728, 0.70886012,
0.70695581,
        0.70831375, 0.70810501, 0.70688053, 0.70687315, 0.7062481 ,
        0.7072668 , 0.70777806, 0.70691623, 0.70624417, 0.70726971,
        0.7077779 , 0.70687315, 0.70600372, 0.70726523, 0.70777288]),
 'split3_train_score': array([0.88075311, 0.84521835, 0.87198033, 0.83544395,
0.87730559,
        0.8600593 , 0.87633567, 0.87806621, 0.89685137, 0.89346596,
        0.87415038, 0.89001398, 0.89685136, 0.89346596, 0.87415037,
        0.89001398, 0.89685137, 0.8961668 , 0.87415037, 0.89001398]),
 'split4_test_score': array([0.71263238, 0.71209744, 0.71199488, 0.71490968,
0.71070192,
        0.71159093, 0.71211287, 0.71197115, 0.7075478 , 0.71169263,
        0.71190288, 0.7155894 , 0.70756117, 0.71171981, 0.71189215,
        0.7155894 , 0.70756161, 0.71172461, 0.71189684, 0.7155894 ]),
 'split4_train_score': array([0.87981562, 0.88295088, 0.86816594, 0.83800614,
0.87887698,
        0.89737111, 0.85600943, 0.81655447, 0.88045246, 0.87364014,
        0.86183281, 0.85553233, 0.88045246, 0.87364014, 0.86183282,
        0.85553233, 0.88045247, 0.87364013, 0.86183281, 0.85553232]),
 'std_fit_time': array([0.56805255, 0.67413838, 0.26274968, 0.27617122,
0.20501494,
        0.6763349 , 0.51211541, 0.99442321, 0.69136667, 0.65784432,
        0.86645774, 0.80129012, 0.73601096, 0.39340259, 0.8570197 ,
        0.58297902, 0.77660755, 0.50915   , 0.69621483, 0.75983436]),
 'std_score_time': array([0.00645762, 0.00845172, 0.00346555, 0.00454566,
0.00336508,
        0.00538988, 0.00741121, 0.0091677 , 0.00712437, 0.00339897,
        0.00961751, 0.01179595, 0.00467033, 0.00292289, 0.0060649 ,
        0.00881112, 0.00485069, 0.0036865 , 0.00760142, 0.01329356]),
 'std_test_score': array([0.00568392, 0.00686153, 0.00541717, 0.0056714 ,
0.00591103,
        0.00455259, 0.004658  , 0.00619194, 0.00669072, 0.00604909,
        0.00543635, 0.00548923, 0.00669923, 0.00606274, 0.00542367,
        0.00548929, 0.00670478, 0.00611668, 0.0054254 , 0.005491  ]),
 'std_train_score': array([0.01260362, 0.02089004, 0.01044884, 0.00641077,
0.00335572,
        0.01737144, 0.0111979 , 0.0210198 , 0.01388504, 0.0087665 ,
        0.01994112, 0.01684903, 0.01388504, 0.00876649, 0.01994112,
        0.01684903, 0.01388503, 0.00969983, 0.01994112, 0.01684903])}
```

[22]:
```
print("TFIDF BEST PARAMS : " , scorer['tfidf'].best_params_)
print("TFIDF BEST AUC-Score : " , scorer['tfidf'].best_score_, "\n\n")

print("TFIDF_Wheighted_W2V - BEST PARAMS : " , scorer['w2v'].best_params_)
print("TFIDF_Wheighted_W2V - BEST AUC-Score : " , scorer['w2v'].best_score_)
```

```
TFIDF BEST PARAMS :  {'min_child_samples': 100, 'max_depth': 100}
TFIDF BEST AUC-Score :  0.7004693567560001


TFIDF_Wheighted_W2V - BEST PARAMS :  {'min_child_samples': 500, 'max_depth': 5}
TFIDF_Wheighted_W2V - BEST AUC-Score :  0.717451848857623
```


```
TFIDF BEST PARAMS :  {'min_child_samples': 100, 'max_depth': 100}

TFIDF BEST AUC-Score :  0.7004693567560001


TFIDF_Wheighted_W2V - BEST PARAMS :  {'min_child_samples': 500, 'max_depth': 5}

TFIDF_Wheighted_W2V - BEST AUC-Score :  0.717451848857623
```

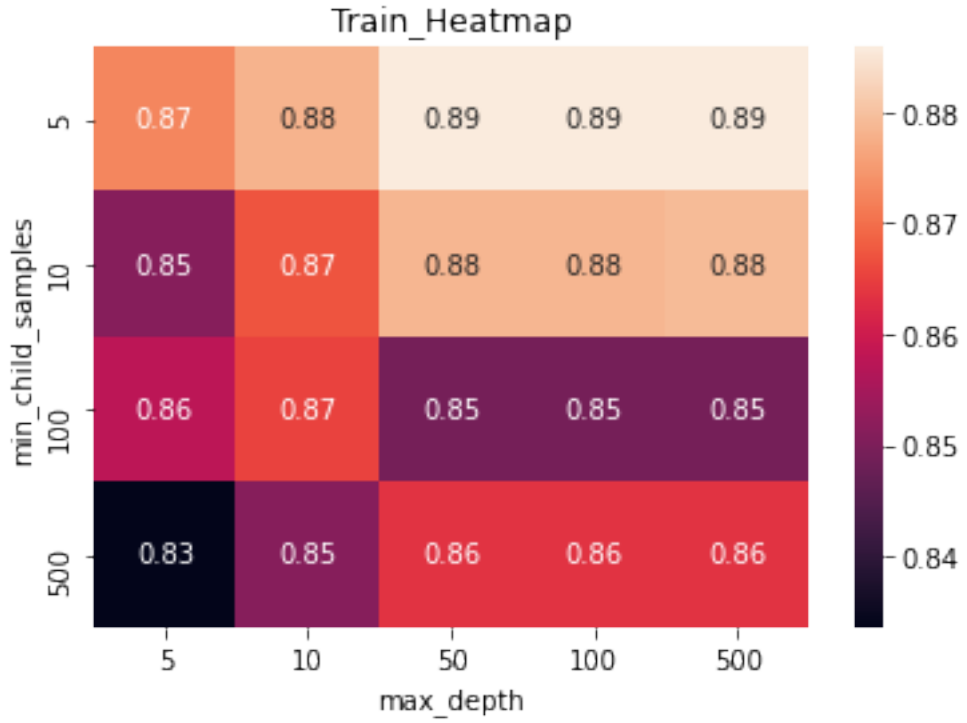

1.5 1.5 CV-Results Representation

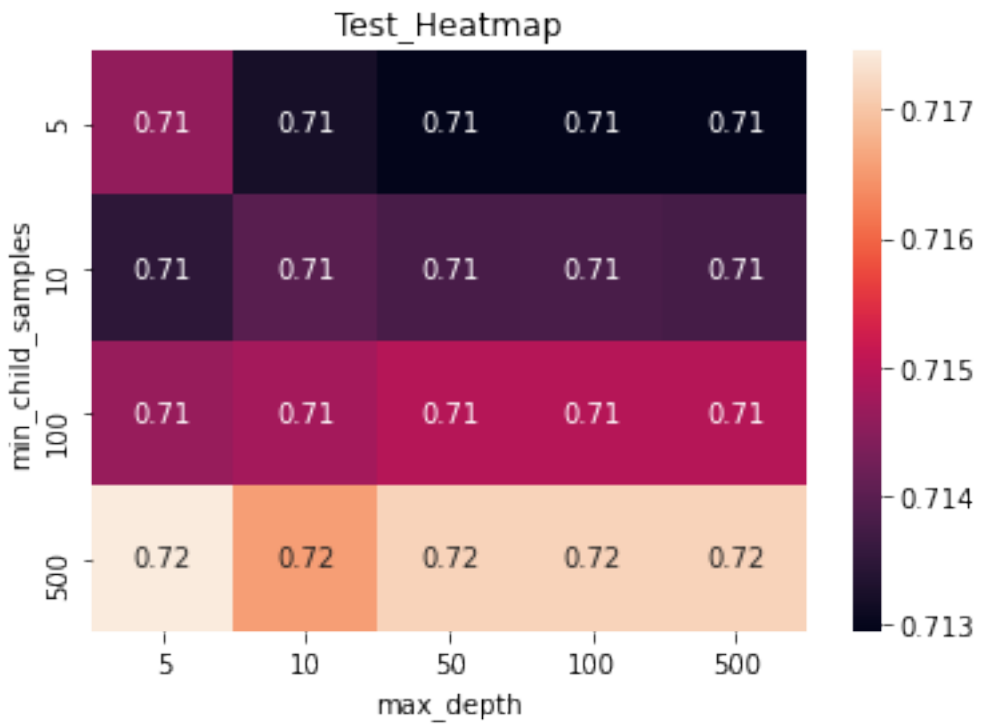
[27]:
```
mean_test_score = scorer['w2v'].cv_results_['mean_test_score']
mean_train_score = scorer['w2v'].cv_results_['mean_train_score']
params = scorer['w2v'].cv_results_['params']

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

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

df.head()
```

[27]:

	min_child_samples	mean_test_score	mean_train_score	max_depth
0	5	0…	0…	5
1	10	0…	0…	5
2	100	0…	0…	5
3	500	0…	0…	5
4	5	0…	0…	10

[28]:
```
train_heatmap = df.pivot(index='min_child_samples', columns='max_depth',
                         values='mean_train_score')
test_heatmap = df.pivot(index='min_child_samples', 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()
```





1.6 1.6 Model-Training

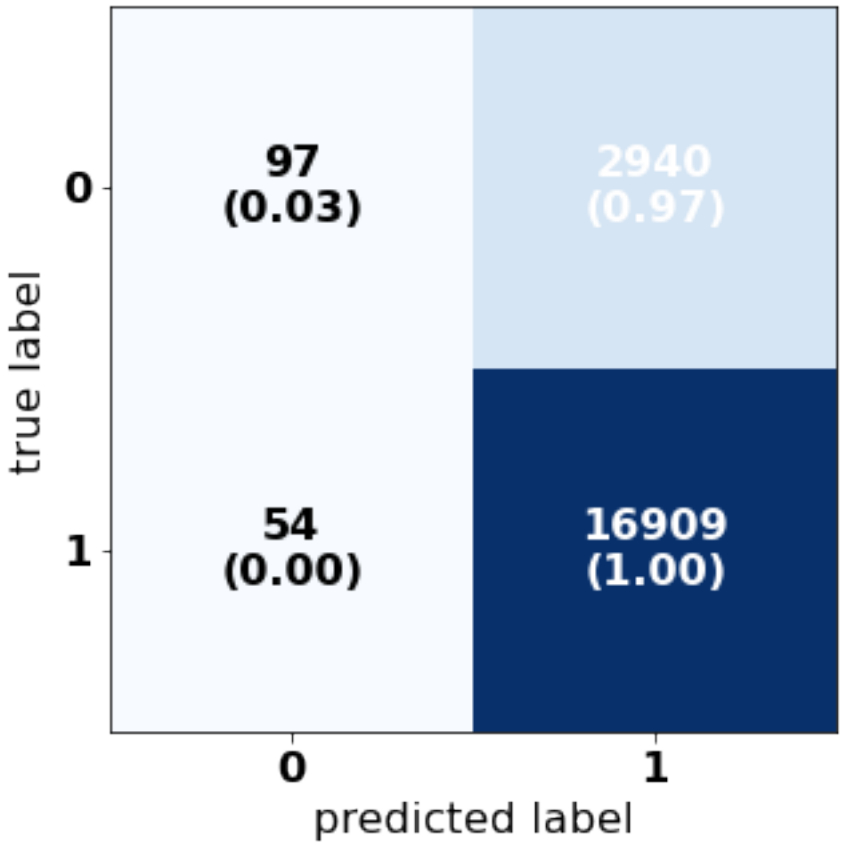
[39]:
```
vectorizer1 = TFIDF_W2V(X_train_title)
temp_1 = getTFIDF_W2V(X_train_title, vectorizer1)

vectorizer2 = TFIDF_W2V(X_train_essay)
temp_2 = getTFIDF_W2V(X_train_essay, vectorizer2)

temp_1 = np.column_stack((temp_1, temp_2,
                          sentiment_anayser(X_train_essay)))
temp_1 = np.column_stack((temp_1, xtrain.values))

print("Final_Train_Matrix : ", temp_1.shape)

essay_ = getTFIDF_W2V(X_test_essay, vectorizer2)
temp_2 = getTFIDF_W2V(X_test_title, vectorizer1)
temp_2 = np.column_stack((temp_2, essay_, sentiment_anayser(X_test_essay)))
temp_2 = np.column_stack((temp_2, xtest.values))
print("Final_Test_Matrix : ", temp_2.shape)
```

```
Final_Train_Matrix :  (60000, 616)
Final_Test_Matrix :  (20000, 616)
```

[40]:
```
clf = lgb.LGBMClassifier(min_child_samples = 500, max_depth = 5, device="gpu",␣
↪random_state=300, silent=True, metric='auc', n_jobs=-1)
clf.fit(temp_1, ytrain)
Y_pred = clf.predict(temp_2)
```

1.7 1.7 Confusion - Matrix

[41]:
```
from mlxtend.plotting import plot_confusion_matrix

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);
```



1.8 1.8 AUC Score

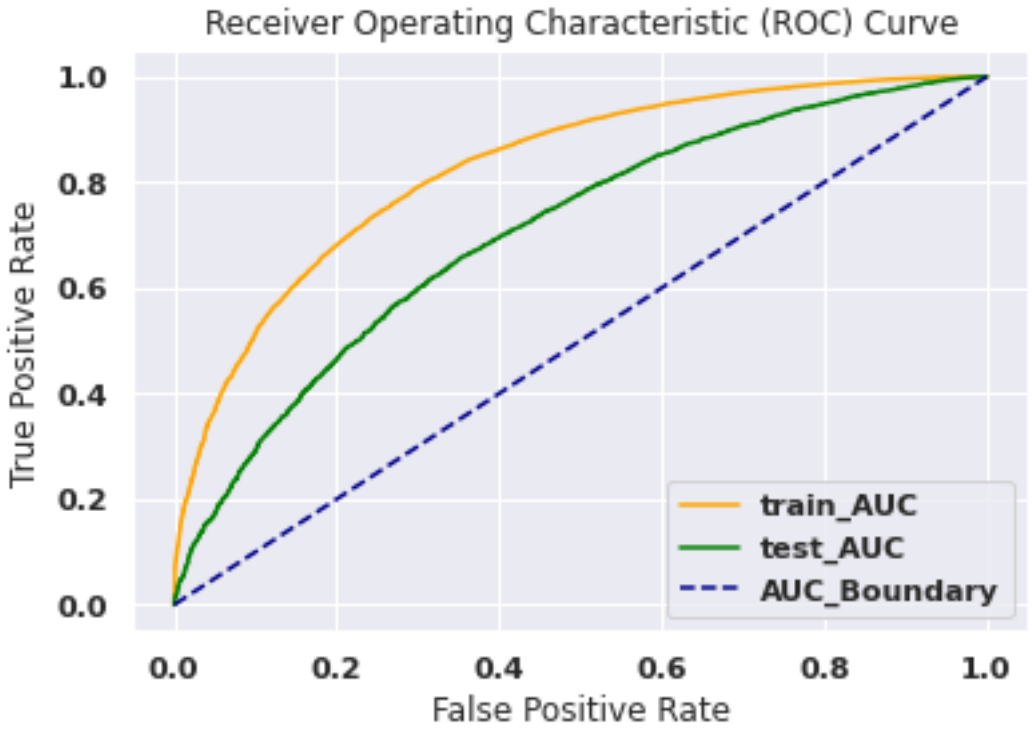
[44]:
```
from sklearn.metrics import auc, roc_curve

print("train_roc_auc_score : " , roc_auc_score(ytrain, clf.predict(temp_1)),
      '\n')
print("test_roc_auc_score : ", roc_auc_score(ytest, Y_pred),
      '\n')

probs = clf.predict_proba(temp_1)
probs = probs[:, 1]
train_fpr, train_tpr, train_thresholds = roc_curve(ytrain, probs)
probs = clf.predict_proba(temp_2)
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 :  0.5288673359535072

test_roc_auc_score :  0.5143780073662038

train_auc_score :  0.8264402979328005

test_auc_score :  0.7057330476443617
```

[45]:
```
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 :  0.8264402979328005
```

```
test_auc_score :  0.7057330476443617
```



2 RESULT

[46]:
```
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper_Parameter", "Train_AUC",
                 "Test_AUC"]

x.add_row(["TFIDF-W2V",'LightGBDT', 'max_depth : 5, min_child_samples :500',
           0.89, '0.71(CV)'])

x.add_row(["TFIDF", 'LightGBDT', 'max_depth : 100, min_child_samples : 100',
           0.82, '0.70(CV)'])

x.add_row(["TFIDF-W2V", 'LightGBDT', 'max_depth : 5, min_child_samples : 500',
           0.82, 0.70])

print(x)
```

Vectorizer	Model	Hyper_Parameter	Train_AUC	Test_AUC
TFIDF-W2V	LightGBDT	max_depth : 5, min_child_samples :500	0.89	0.71(CV)
TFIDF	LightGBDT	max_depth : 100, min_child_samples : 100	0.82	0.70(CV)
TFIDF-W2V	LightGBDT	max_depth : 5, min_child_samples : 500	0.82	0.7