parth.pandey13103447@gmail_13

August 23, 2020

1 Apply GBDT

> The response tabel is built only on train dataset. For a category which is not there in train data and present in test data, we will encode them with default values Ex: in our test data if have State: D then we encode it as [0.5, 0.05]

Apply GBDT on these feature sets

Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check the bellow example, include all 4 values as 4 features)

Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

```
</li>
<li><strong>The hyper paramter tuning (Consider any two hyper parameters)</strong>
    <ul>
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
<li>find the best hyper paramter using k-fold cross validation/simple cross validation data</l
<li>use gridsearch cv or randomsearch cv or you can write your own for loops to do this task</
    </ul>
</li>
<li>
<strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='https://i.imgur.com/Gp2DQmh.jpg' width=500px> with X-axis as <strong>n_estimators</s
        <p style="text-align:center;font-size:30px;color:red;"><strong>or</strong></p> <br>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='https://i.imgur.com/fgN9aUP.jpg' width=300px> <a href='https://seaborn.pydata.org/ge
<li>You choose either of the plotting techniques out of 3d plot or heat map</li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and f
<img src='https://i.imgur.com/wMQDTFe.jpg' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.
<img src='https://i.imgur.com/IdN5Ctv.png' width=300px></li>
        </ul>
<br>
<li>You need to summarize the results at the end of the notebook, summarize it in the table fo
    <img src='http://i.imgur.com/YVpIGGE.jpg' width=400px>
```


1.1 Imports

[1]:
```
!pip install beautifultable
```

```
Collecting beautifultable
  Downloading https://files.pythonhosted.org/packages/00/f8/63a013f19d6b4a2f9cc8
706a98ad6261bff4941de4472a1b5e828803335d/beautifultable-1.0.0-py2.py3-none-
any.whl
Requirement already satisfied: wcwidth in /usr/local/lib/python3.6/dist-packages
(from beautifultable) (0.2.5)
Installing collected packages: beautifultable
Successfully installed beautifultable-1.0.0
```

[50]:
```
import warnings
warnings.filterwarnings('ignore')
import nltk
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from tqdm import tqdm
from scipy.sparse import hstack


import xgboost as xgb
# import lightgbm as lgb
from sklearn.model_selection            import train_test_split
from sklearn.preprocessing              import StandardScaler
from sklearn.feature_extraction.text    import TfidfVectorizer
from sklearn.metrics                    import roc_auc_score, confusion_matrix,␣
↪plot_roc_curve, roc_curve


from beautifultable import BeautifulTable
table = BeautifulTable()


%matplotlib inline
```

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

[3]:
```python
sid = SentimentIntensityAnalyzer()
for_sentiment = 'a person is a person no matter how small dr seuss i teach the␣
 ↪smallest students with the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses␣
 ↪and multiple intelligences i use a wide range\
of techniques to help all my students succeed students in my class come from a␣
 ↪variety of different backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans␣
 ↪our school is a caring community of successful \
learners which can be seen through collaborative student project based learning␣
 ↪in and out of the classroom kindergarteners \
in my class love to work with hands on materials and have many different␣
 ↪opportunities to practice a skill before it is\
mastered having the social skills to work cooperatively with friends is a␣
 ↪crucial aspect of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my␣
 ↪students love to role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try␣
 ↪cooking with real food i will take their idea \
and create common core cooking lessons where we learn important math and␣
 ↪writing concepts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for the work␣
 ↪that went into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for their␣
 ↪bodies this project would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to␣
 ↪make homemade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also␣
 ↪create our own cookbooks to be printed and \
shared with families students will gain math and literature skills as well as a␣
 ↪life long enjoyment for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)
print(ss)
for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
{'neg': 0.01, 'neu': 0.745, 'pos': 0.245, 'compound': 0.9975}
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

[4]:
```python
data = pd.read_csv('drive/My Drive/Colab Notebooks/AppliedAICourse/Assignment/
 ↪preprocessed_data.csv')
data.head()
```

[4]:
```
   school_state  …   price
0            ca  …  725.05
1            ut  …  213.03
2            ca  …  329.00
3            ga  …  481.04
4            wa  …   17.74

[5 rows x 9 columns]
```

1.2 Defining Reusable Functions

[5]:
```python
# Plotting Confusion Matrix
def plot_matrix(y_true, y_pred):
    '''Input: true labes , predicted labels
       Output: None
       Functionality: Plots the confusion, precison and recall matrices
    '''
    conf = confusion_matrix(y_true, y_pred)
    # Column Sum = 1
    precision = conf/conf.sum(0)
    # Row Sum = 1
    recall    = (conf.T/conf.sum(1)).T
    cmap='YlGnBu'
    labels = [1,2,3,4,5,6,7,8,9]
    print('-'*20,'Confusion Matrix','-'*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(conf ,annot=True, fmt='.3f', cmap=cmap, xticklabels=labels,␣
 ↪yticklabels=labels)
    plt.xlabel('Predicted')
    plt.ylabel('Original')
    plt.show()
    print('-'*20,'Precision Matrix ( Columns Sum == 1 )','-'*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(precision ,annot=True, fmt='.3f', cmap=cmap,␣
 ↪xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted')
    plt.ylabel('Original')
    plt.show()
    print('-'*20,'Recall Matrix ( Row Sum == 1 )','-'*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(recall ,annot=True, fmt='.3f', cmap=cmap, xticklabels=labels,␣
 ↪yticklabels=labels)
    plt.xlabel('Predicted')
    plt.ylabel('Original')
    plt.show()

# Defining Functions for Categorical Columns Response Encoding
```

```
def response_encoding_fitting(x_train,y_train,feature,label,alpha):
    '''
    Input   : x_train, y_train, feature, label, alpha
    Output  : dictionary containing response coded features
    Functionality: Encoding a feature using response encoding feature techniques
    '''
    temp = x_train.copy()
    temp[label] = y_train
    temp = temp.groupby([feature,label])[label].agg(Total='count').
 ↪reset_index().sort_values([feature,label])
    response_encoding = {}
    # For evey ele
    for ele in temp[feature].unique():
        response = np.zeros((2,1))
        # Filter DataFrame Values for that ele
        x = temp[temp[feature] == ele]
        total = x['Total'].sum()
        # For each class present for the ele

        for i in temp[label].unique():
            z=x[x[label] == i]['Total']
            if len(z.values) == 0:
                z=0
            numerator   = (z + 10 * alpha)
            denominator = (total +( 20 * alpha))
            response[i-1] = (numerator)/(denominator)
        response_encoding[ele] = response.T[0]
    return response_encoding

def response_encoding_fit_transform(x,y,feature,label,alpha):
    '''
    Input   : x, y, feature, label
    Output  : response encoded data
    Functionality: fit and Transforming the data into response encoded data
    '''

    temp = x.copy()
    temp[label] = y

    response_dictionary = response_encoding_fitting(x, y, feature, label, alpha)

    final_feature = []

    for ind, row in temp.iterrows():
        if row[feature] in response_dictionary.keys():
            final_feature.append(response_dictionary[row[feature]])
        else:
```

```
            feature_count = len(df[feature].unique())
            final_feature.append(np.ones(feature_count)/feature_count)
    return np.array(final_feature)

# Creating Response Encoding Functions for TEXT
def creating_word_count_dict(df):
    return dict(Counter(' '.join(df['Text'].tolist()).split()))

def response_encoding_text(curr_df,label):
    total_wc_dict = creating_word_count_dict(curr_df)
    response_encoding_feature = np.zeros((curr_df.shape[0],2))
    for cls in curr_df.unique():
        # print('calculating for class {}'.format(cls))
        temp_df = curr_df[label['Class'] == cls]
        # print(temp_df.shape)
        per_class_wc_dict = creating_word_count_dict(temp_df)
        index = 0
        for ind, row in curr_df.iterrows():
            sum_prob = 0
            for word in row['Text'].split():
                # print('Calculation for word = {}'.format(word))
                temp = ((per_class_wc_dict.get(word,0)+10)/(total_wc_dict.
 ↪get(word,0) + 20) )
                # print(temp)
                sum_prob += math.log( temp )
            response_encoding_feature[index][cls-1] = math.exp(sum_prob/
 ↪len(row['Text'].split()))
            index += 1
    return response_encoding_feature

def create_tfidf_w2v(df, col):

    with open('drive/My Drive/Colab Notebooks/AppliedAICourse/Assignment/
 ↪glove_vectors', 'rb') as f:
        model = pickle.load(f)
        glove_words =  set(model.keys())

    # Creating TFIDF
    tfidf_vec = TfidfVectorizer()
    tfidf_vec.fit(df[col])
    idf_dict = dict(zip(tfidf_vec.get_feature_names(), list(tfidf_vec.idf_)))
    tfidf_words = set(tfidf_vec.get_feature_names())

    # Creating TVIDF weigthed W2V
    tfidf_w2v_vectors = []
    for sentence in tqdm(df[col].values):
        vector = np.zeros(300)
```

```
        tfidf_val = 0
        for word in sentence.split():
            if word in glove_words and word in tfidf_words:
                temp = model[word]
                # Calcualting the tfidf values
                tf_idf = idf_dict[word] * (sentence.count(word)/len(sentence.
 ↪split()))
                vector += temp * tf_idf
                tfidf_val += tf_idf
        if tfidf_val != 0:
            vector /= tfidf_val
        tfidf_w2v_vectors.append(vector)
    return tfidf_w2v_vectors



def transforming(x_train, y_train, x_test, y_test, x_cv, y_cv, col, alpha):
    # Transforming Integer Fields
    if x_train[col].dtype == np.dtype('int64') or x_train[col].dtype == np.
 ↪dtype('float64'):

        std = StandardScaler()
        x_train_val = std.fit_transform(x_train[col].values.reshape(-1,1))
        x_cv_val    = std.transform(x_cv[col].values.reshape(-1,1))
        x_test_val  = std.transform(x_test[col].values.reshape(-1,1))

   # Transfroming
    if x_train[col].dtype == np.dtype('object'):

        if col == 'essay':
            tf_vec     =␣
 ↪TfidfVectorizer(ngram_range=(1,3),min_df=10,norm='l2',max_features=50000)
            x_train_val = tf_vec.fit_transform(x_train[col].values).tocsr()
            x_test_val    = tf_vec.transform(x_test[col].values).tocsr()
            x_cv_val   = tf_vec.transform(x_cv[col].values).tocsr()

        else:
            x_train_val = response_encoding_fit_transform(x_train, y_train,␣
 ↪col, 'project_is_approved', alpha)
            x_test_val = response_encoding_fit_transform(x_test, y_test, col,␣
 ↪'project_is_approved', alpha)
            x_cv_val = response_encoding_fit_transform(x_cv, y_cv, col,␣
 ↪'project_is_approved', alpha)

    return (x_train_val, x_cv_val, x_test_val)
```

```
def sentiment_score_feature(text):
    sid = SentimentIntensityAnalyzer()
    temp = sid.polarity_scores(text)
    return (temp['neg'], temp['neu'], temp['pos'], temp['compound'])
```

1.3 Splitting the data

[6]:
```
x = data.drop(['project_is_approved'], axis=1)
y = data['project_is_approved']

x_train, x_test, y_train, y_test = train_test_split(x,       y,       ␣
↪stratify=y,       test_size=0.2)
x_train, x_cv,    y_train, y_cv    = train_test_split(x_train, y_train,␣
↪stratify=y_train, test_size=0.2)
```

1.4 Feature Transformation

[7]:
```
x_train_dict = {}
x_cv_dict     = {}
x_test_dict  = {}

for col in x_train.columns:
    print('Transforming', col)
    result = transforming(x_train, y_train, x_test, y_test, x_cv, y_cv, col, 1)
    x_train_dict[col] = result[0]
    x_cv_dict[col]     = result[1]
    x_test_dict[col]   = result[2]

train_essay_tfidf = create_tfidf_w2v(x_train, 'essay')
test_essay_tfidf = create_tfidf_w2v(x_test, 'essay')
cv_essay_tfidf = create_tfidf_w2v(x_cv, 'essay')
```

```
Transforming school_state
Transforming teacher_prefix
Transforming project_grade_category
Transforming teacher_number_of_previously_posted_projects
Transforming clean_categories
Transforming clean_subcategories
Transforming essay
Transforming price
```

```
100%|      | 69918/69918 [02:04<00:00, 562.26it/s]
100%|      | 21850/21850 [00:38<00:00, 567.39it/s]
100%|      | 17480/17480 [00:34<00:00, 500.08it/s]
```

[8]:
```
train_essay_neg     = np.array([])
train_essay_neu     = np.array([])
train_essay_pos     = np.array([])
train_essay_comp    = np.array([])

test_essay_neg     = np.array([])
test_essay_neu     = np.array([])
test_essay_pos     = np.array([])
test_essay_comp    = np.array([])

cv_essay_neg     = np.array([])
cv_essay_neu     = np.array([])
cv_essay_pos     = np.array([])
cv_essay_comp    = np.array([])


for tr in tqdm(x_train['essay']):
    temp = sentiment_score_feature(tr)
    train_essay_neg = np.append(train_essay_neg, temp[0])
    train_essay_neu = np.append(train_essay_neu, temp[1])
    train_essay_pos = np.append(train_essay_pos, temp[2])
    train_essay_comp = np.append(train_essay_comp, temp[3])

for te in tqdm(x_test['essay']):
    temp = sentiment_score_feature(te)
    test_essay_neg = np.append(test_essay_neg, temp[0])
    test_essay_neu = np.append(test_essay_neu, temp[1])
    test_essay_pos = np.append(test_essay_pos, temp[2])
    test_essay_comp = np.append(test_essay_comp, temp[3])

for cv in  tqdm(x_cv['essay']):
    temp = sentiment_score_feature(cv)
    cv_essay_neg = np.append(cv_essay_neg, temp[0])
    cv_essay_neu = np.append(cv_essay_neu, temp[1])
    cv_essay_pos = np.append(cv_essay_pos, temp[2])
    cv_essay_comp = np.append(cv_essay_comp, temp[3])
```

```
100%|     | 69918/69918 [09:54<00:00, 117.64it/s]
100%|     | 21850/21850 [03:04<00:00, 118.12it/s]
100%|     | 17480/17480 [02:26<00:00, 119.33it/s]
```

1.4.1 Set I Feature

[9]:
```
x_train_set_i = hstack((
    x_train_dict['school_state'],
    x_train_dict['teacher_prefix'],
    x_train_dict['project_grade_category'],
```

```
    x_train_dict['teacher_number_of_previously_posted_projects'],
    x_train_dict['clean_categories'],
    x_train_dict['clean_subcategories'],
    x_train_dict['essay'],
    x_train_dict['price'],
    train_essay_neg.reshape(-1,1),
    train_essay_neu.reshape(-1,1),
    train_essay_pos.reshape(-1,1),
    train_essay_comp.reshape(-1,1)
))
print(x_train_set_i.shape)



x_test_set_i = hstack((
    x_test_dict['school_state'],
    x_test_dict['teacher_prefix'],
    x_test_dict['project_grade_category'],
    x_test_dict['teacher_number_of_previously_posted_projects'],
    x_test_dict['clean_categories'],
    x_test_dict['clean_subcategories'],
    x_test_dict['essay'],
    x_test_dict['price'],
    test_essay_neg.reshape(-1,1),
    test_essay_neu.reshape(-1,1),
    test_essay_pos.reshape(-1,1),
    test_essay_comp.reshape(-1,1)
))
print(x_test_set_i.shape)


x_cv_set_i = hstack((
    x_cv_dict['school_state'],
    x_cv_dict['teacher_prefix'],
    x_cv_dict['project_grade_category'],
    x_cv_dict['teacher_number_of_previously_posted_projects'],
    x_cv_dict['clean_categories'],
    x_cv_dict['clean_subcategories'],
    x_cv_dict['essay'],
    x_cv_dict['price'],
    cv_essay_neg.reshape(-1,1),
    cv_essay_neu.reshape(-1,1),
    cv_essay_pos.reshape(-1,1),
    cv_essay_comp.reshape(-1,1)

))
print(x_cv_set_i.shape)
```

```
(69918, 50016)
(21850, 50016)
(17480, 50016)
```

1.4.2 Set II Feature

[10]:
```
x_train_set_ii = np.hstack((
    x_train_dict['school_state'],
    x_train_dict['teacher_prefix'],
    x_train_dict['project_grade_category'],
    x_train_dict['teacher_number_of_previously_posted_projects'],
    x_train_dict['clean_categories'],
    x_train_dict['clean_subcategories'],
    np.array(train_essay_tfidf),
    x_train_dict['price']
))
print(x_train_set_ii.shape)


x_cv_set_ii = np.hstack((
    x_cv_dict['school_state'],
    x_cv_dict['teacher_prefix'],
    x_cv_dict['project_grade_category'],
    x_cv_dict['teacher_number_of_previously_posted_projects'],
    x_cv_dict['clean_categories'],
    x_cv_dict['clean_subcategories'],
    np.array(cv_essay_tfidf),
    x_cv_dict['price']
))
print(x_cv_set_ii.shape)




x_test_set_ii = np.hstack((
    x_test_dict['school_state'],
    x_test_dict['teacher_prefix'],
    x_test_dict['project_grade_category'],
    x_test_dict['teacher_number_of_previously_posted_projects'],
    x_test_dict['clean_categories'],
    x_test_dict['clean_subcategories'],
    np.array(test_essay_tfidf),
    x_test_dict['price']
))
print(x_test_set_ii.shape)
```

```
(69918, 312)
(17480, 312)
```

```
(21850, 312)
```

1.5 Applying XGBoost

1.5.1 Set I

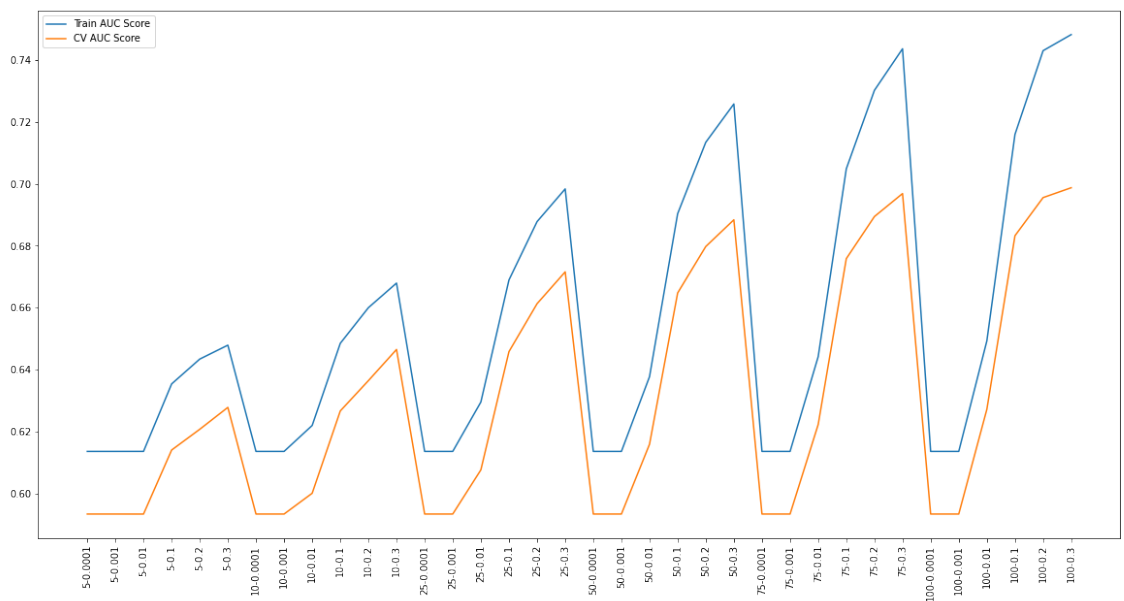
[34]:
```
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
n_estimators=[5, 10, 25, 50, 75, 100]
train_auc_score = {}
cv_auc_score    = {}
for ele in n_estimators:
    for rate  in learning_rate:
        order = '{}-{}'.format(ele, rate)
        param_dist = {
            'objective':'binary:logistic',
            'n_jobs':-1,
            'n_estimators':ele,
            'learning_rate':rate,
            'random_state':42
            }

        clf = xgb.XGBClassifier(**param_dist)

        clf.fit(x_train_set_i, y_train,
                eval_set=[(x_train_set_i, y_train), (x_cv_set_i, y_cv)],
                eval_metric='auc',
                early_stopping_rounds=10,
                verbose=False)

        eval_result = clf.evals_result()
        train_auc = np.mean(eval_result['validation_0']['auc'])
        print('Train AUC Score {} order = {} '.format(train_auc, order))
        cv_auc    = np.mean(eval_result['validation_1']['auc'])
        print('CV AUC Score {} order = {} '.format(cv_auc, order))
        train_auc_score[order] = train_auc
        cv_auc_score[order] = cv_auc
```

```
Train AUC Score 0.613565 order = 5-0.0001
CV AUC Score 0.593334 order = 5-0.0001
Train AUC Score 0.613565 order = 5-0.001
CV AUC Score 0.593334 order = 5-0.001
Train AUC Score 0.6135692 order = 5-0.01
CV AUC Score 0.5933240000000001 order = 5-0.01
Train AUC Score 0.6353658 order = 5-0.1
CV AUC Score 0.6140087999999999 order = 5-0.1
Train AUC Score 0.6433934 order = 5-0.2
CV AUC Score 0.620714 order = 5-0.2
Train AUC Score 0.6478854000000001 order = 5-0.3
```

```
CV AUC Score 0.6277888 order = 5-0.3
Train AUC Score 0.6135650000000001 order = 10-0.0001
CV AUC Score 0.5933340000000001 order = 10-0.0001
Train AUC Score 0.6135650000000001 order = 10-0.001
CV AUC Score 0.5933340000000001 order = 10-0.001
Train AUC Score 0.62195 order = 10-0.01
CV AUC Score 0.6000495 order = 10-0.01
Train AUC Score 0.6484827 order = 10-0.1
CV AUC Score 0.6266406 order = 10-0.1
Train AUC Score 0.6600016 order = 10-0.2
CV AUC Score 0.6364305 order = 10-0.2
Train AUC Score 0.6679421999999999 order = 10-0.3
CV AUC Score 0.6464943 order = 10-0.3
Train AUC Score 0.6135650000000001 order = 25-0.0001
CV AUC Score 0.5933340000000001 order = 25-0.0001
Train AUC Score 0.6135650000000001 order = 25-0.001
CV AUC Score 0.5933340000000001 order = 25-0.001
Train AUC Score 0.62951492 order = 25-0.01
CV AUC Score 0.60759652 order = 25-0.01
Train AUC Score 0.6689549600000001 order = 25-0.1
CV AUC Score 0.64580584 order = 25-0.1
Train AUC Score 0.68779872 order = 25-0.2
CV AUC Score 0.66129092 order = 25-0.2
Train AUC Score 0.69836324 order = 25-0.3
CV AUC Score 0.67157088 order = 25-0.3
Train AUC Score 0.6135650000000001 order = 50-0.0001
CV AUC Score 0.5933340000000001 order = 50-0.0001
Train AUC Score 0.6135650000000001 order = 50-0.001
CV AUC Score 0.5933340000000001 order = 50-0.001
Train AUC Score 0.6377121199999999 order = 50-0.01
CV AUC Score 0.61588404 order = 50-0.01
Train AUC Score 0.69034918 order = 50-0.1
CV AUC Score 0.66474854 order = 50-0.1
Train AUC Score 0.7134200199999999 order = 50-0.2
CV AUC Score 0.6797507600000001 order = 50-0.2
Train AUC Score 0.7258074600000001 order = 50-0.3
CV AUC Score 0.68838396 order = 50-0.3
Train AUC Score 0.6135650000000001 order = 75-0.0001
CV AUC Score 0.5933340000000001 order = 75-0.0001
Train AUC Score 0.6135650000000001 order = 75-0.001
CV AUC Score 0.5933340000000001 order = 75-0.001
Train AUC Score 0.6441840533333334 order = 75-0.01
CV AUC Score 0.6222559733333334 order = 75-0.01
Train AUC Score 0.7048624266666667 order = 75-0.1
CV AUC Score 0.67584108 order = 75-0.1
Train AUC Score 0.7302047600000001 order = 75-0.2
CV AUC Score 0.6894339866666666 order = 75-0.2
Train AUC Score 0.74363872 order = 75-0.3
```

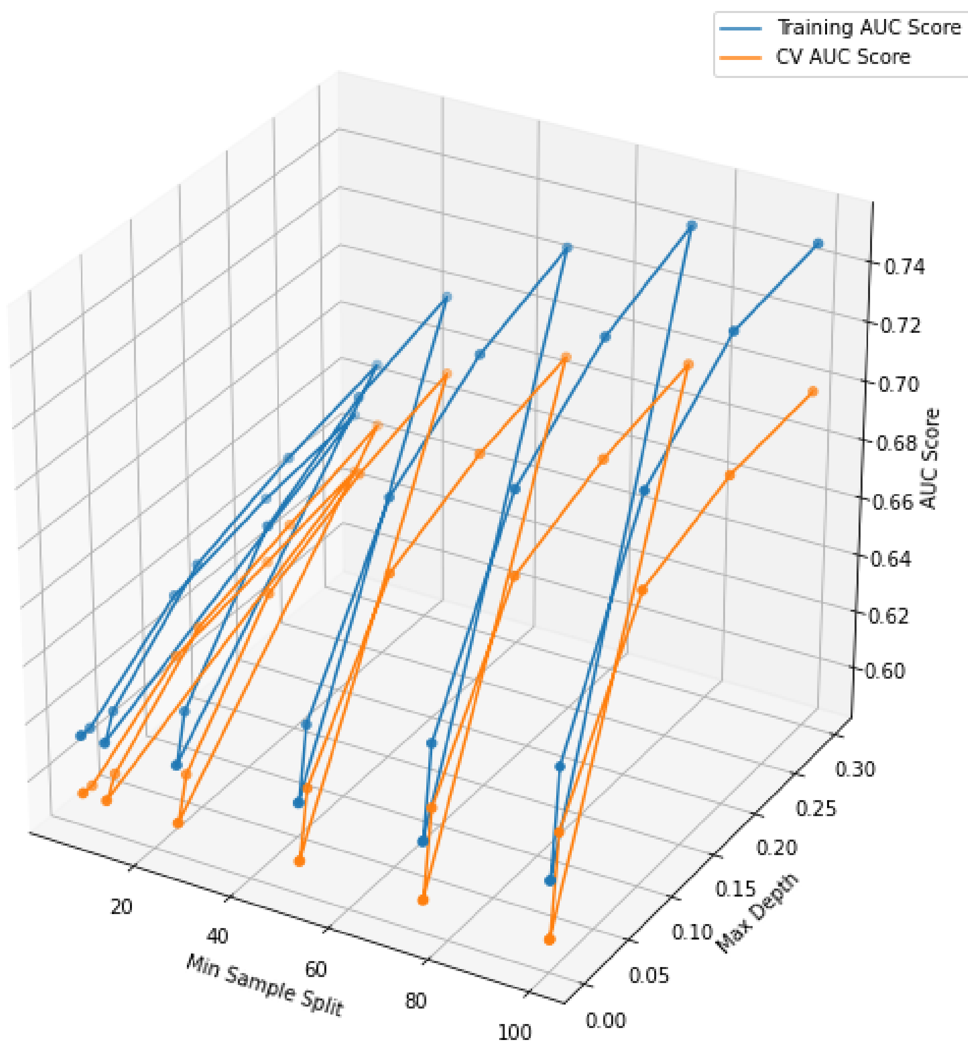
```
CV AUC Score 0.6968640933333334 order = 75-0.3
Train AUC Score 0.6135650000000001 order = 100-0.0001
CV AUC Score 0.5933340000000001 order = 100-0.0001
Train AUC Score 0.6135650000000001 order = 100-0.001
CV AUC Score 0.5933340000000001 order = 100-0.001
Train AUC Score 0.6492686300000001 order = 100-0.01
CV AUC Score 0.6271642400000002 order = 100-0.01
Train AUC Score 0.71597533 order = 100-0.1
CV AUC Score 0.6832450099999998 order = 100-0.1
Train AUC Score 0.7430163799999999 order = 100-0.2
CV AUC Score 0.69556086 order = 100-0.2
Train AUC Score 0.7482200722891568 order = 100-0.3
CV AUC Score 0.6987444216867471 order = 100-0.3
```

[35]:
```python
_,ax = plt.subplots(1,1,figsize=(20,10))
ax.plot(list(train_auc_score.keys()), list(train_auc_score.values()), label='Train AUC Score')
ax.plot(list(cv_auc_score.keys()), list(cv_auc_score.values()), label='CV AUC Score')
for tick in ax.get_xticklabels():
    tick.set_rotation(90)
ax.legend()
plt.show()
```



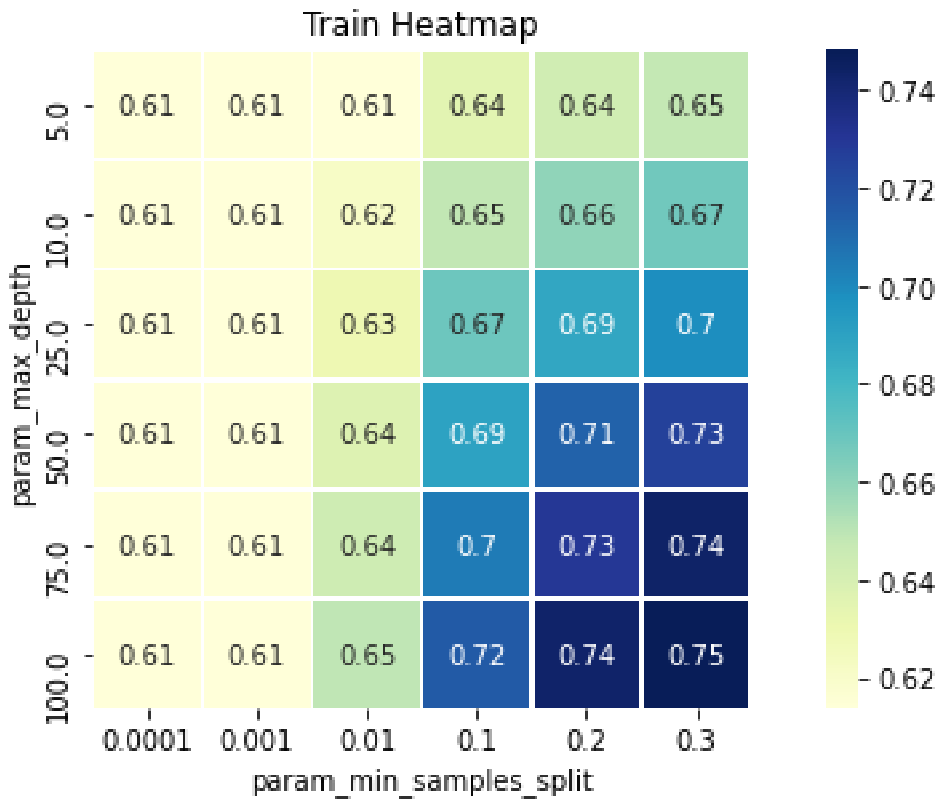
[36]:
```python
x_axis = []
y_axis = []
for key in train_auc_score.keys():
```

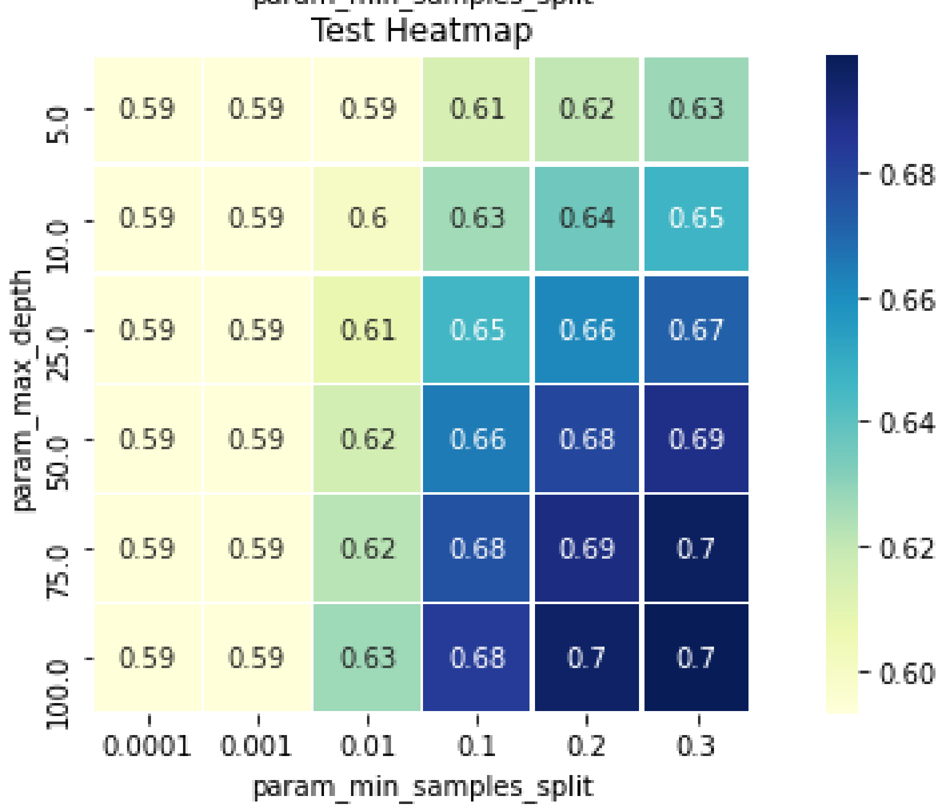
```
    temp = list(map(float, key.split('-')))
    x_axis.append(temp[0])
    y_axis.append(temp[1])
# 3D Plot
subplot_args = {'projection':'3d'}
fig, ax = plt.subplots(1, 1, figsize=(10,10), subplot_kw=subplot_args)
ax.scatter3D(x_axis,y_axis,  list(train_auc_score.values()))
ax.plot3D( x_axis,y_axis, list(train_auc_score.values()), label='Training AUC␣
 ↪Score')
ax.scatter3D(x_axis,y_axis,  list(cv_auc_score.values()))
ax.plot3D(x_axis,y_axis,  list(cv_auc_score.values()), label='CV AUC Score')
ax.legend()
ax.set_xlabel('Min Sample Split')
ax.set_ylabel('Max Depth')
ax.set_zlabel('AUC Score')
plt.show()
```



[37]:

```
temp_df = pd.DataFrame(np.array([x_axis,y_axis,list(train_auc_score.values())]).
↪T, columns=['param_max_depth', 'param_min_samples_split',␣
↪'mean_train_score'])
# Heatmap
_, ax = plt.subplots(1, 2, figsize=(15, 15))
sns.heatmap(data=temp_df.pivot('param_max_depth', 'param_min_samples_split',␣
↪'mean_train_score'),  annot=True, linewidths=.5, square=True, ax =ax[0],␣
↪cmap="YlGnBu")
ax[0].set_title('Train Heatmap')
temp_df = pd.DataFrame(np.array([x_axis,y_axis,list(cv_auc_score.values())]).T,␣
↪columns=['param_max_depth', 'param_min_samples_split', 'mean_cv_score'])
sns.heatmap(data=temp_df.pivot('param_max_depth', 'param_min_samples_split',␣
↪'mean_cv_score'), annot=True, linewidths=.5, square=True, ax =ax[1],␣
↪cmap="YlGnBu")
ax[1].set_title('Test Heatmap')
plt.show()
```


Train Heatmap
param_max_depth
5.0
10.0
25.0
50.0
75.0
100.0
0.61 0.61 0.61 0.64 0.64 0.65
0.61 0.61 0.62 0.65 0.66 0.67
0.61 0.61 0.63 0.67 0.69 0.7
0.61 0.61 0.64 0.69 0.71 0.73
0.61 0.61 0.64 0.7 0.73 0.74
0.61 0.61 0.65 0.72 0.74 0.75
0.0001 0.001 0.01 0.1 0.2 0.3
param_min_samples_split
0.74
0.72
0.70
0.68
0.66
0.64
0.62


Test Heatmap
param_max_depth
5.0
10.0
25.0
50.0
75.0
100.0
0.59 0.59 0.59 0.61 0.62 0.63
0.59 0.59 0.6 0.63 0.64 0.65
0.59 0.59 0.61 0.65 0.66 0.67
0.59 0.59 0.62 0.66 0.68 0.69
0.59 0.59 0.62 0.68 0.69 0.7
0.59 0.59 0.63 0.68 0.7 0.7
0.0001 0.001 0.01 0.1 0.2 0.3
param_min_samples_split
0.68
0.66
0.64
0.62
0.60

1.5.2 Set II

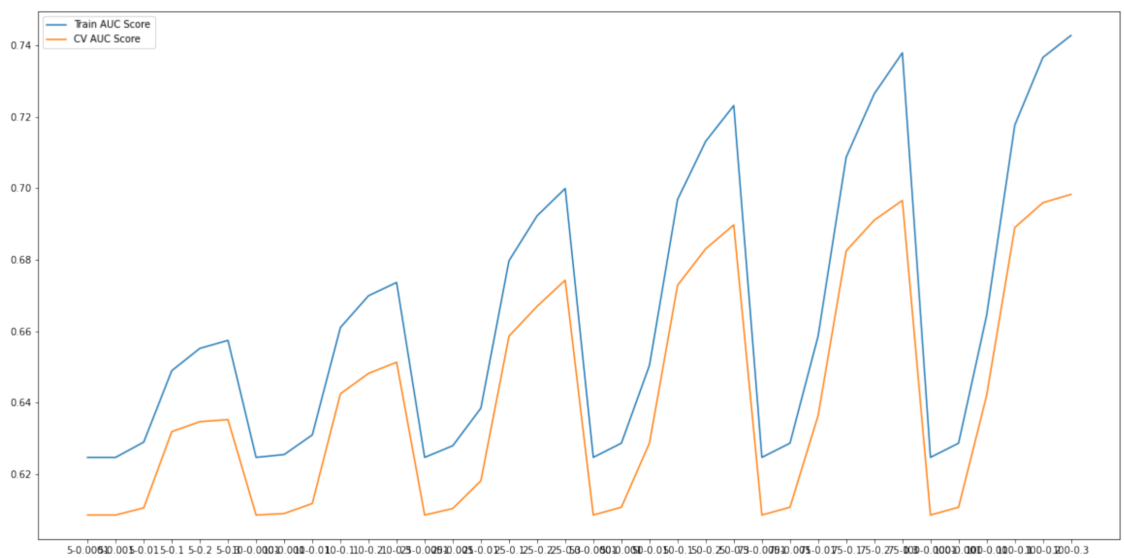
[38]:
```
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
n_estimators=[5, 10, 25, 50, 75, 100]
train_auc_score = {}
cv_auc_score    = {}
for ele in n_estimators:
    for rate  in learning_rate:
        order = '{}-{}'.format(ele, rate)
        param_dist = {
            'objective':'binary:logistic',
            'n_jobs':-1,
            'n_estimators':ele,
            'learning_rate':rate,
            'random_state':42
            }

        clf = xgb.XGBClassifier(**param_dist)

        clf.fit(x_train_set_ii, y_train,
                eval_set=[(x_train_set_ii, y_train), (x_cv_set_ii, y_cv)],
                eval_metric='auc',
                early_stopping_rounds=10,
                verbose=False)

        eval_result = clf.evals_result()
        train_auc = np.mean(eval_result['validation_0']['auc'])
        print('Train AUC Score {} order = {} '.format(train_auc, order))
        cv_auc    = np.mean(eval_result['validation_1']['auc'])
        print('CV AUC Score {} order = {} '.format(cv_auc, order))
        train_auc_score[order] = train_auc
        cv_auc_score[order] = cv_auc
```

```
Train AUC Score 0.624669 order = 5-0.0001
CV AUC Score 0.608529 order = 5-0.0001
Train AUC Score 0.624669 order = 5-0.001
CV AUC Score 0.608529 order = 5-0.001
Train AUC Score 0.6289437999999999 order = 5-0.01
CV AUC Score 0.6105118 order = 5-0.01
Train AUC Score 0.6489600000000001 order = 5-0.1
CV AUC Score 0.6318824 order = 5-0.1
Train AUC Score 0.6551994 order = 5-0.2
CV AUC Score 0.6346706 order = 5-0.2
Train AUC Score 0.6574334000000002 order = 5-0.3
CV AUC Score 0.6352359999999999 order = 5-0.3
Train AUC Score 0.624669 order = 10-0.0001
CV AUC Score 0.608529 order = 10-0.0001
Train AUC Score 0.6254641000000001 order = 10-0.001
```

```
CV AUC Score 0.6089426 order = 10-0.001
Train AUC Score 0.6309579000000001 order = 10-0.01
CV AUC Score 0.6117611999999999 order = 10-0.01
Train AUC Score 0.6610253 order = 10-0.1
CV AUC Score 0.6424689 order = 10-0.1
Train AUC Score 0.6699142 order = 10-0.2
CV AUC Score 0.648174 order = 10-0.2
Train AUC Score 0.6736478 order = 10-0.3
CV AUC Score 0.6513093 order = 10-0.3
Train AUC Score 0.624669 order = 25-0.0001
CV AUC Score 0.608529 order = 25-0.0001
Train AUC Score 0.62793588 order = 25-0.001
CV AUC Score 0.61034264 order = 25-0.001
Train AUC Score 0.6384533600000001 order = 25-0.01
CV AUC Score 0.61811388 order = 25-0.01
Train AUC Score 0.67971964 order = 25-0.1
CV AUC Score 0.6586000799999999 order = 25-0.1
Train AUC Score 0.6923022 order = 25-0.2
CV AUC Score 0.6669679599999999 order = 25-0.2
Train AUC Score 0.6999538 order = 25-0.3
CV AUC Score 0.67427776 order = 25-0.3
Train AUC Score 0.624669 order = 50-0.0001
CV AUC Score 0.608529 order = 50-0.0001
Train AUC Score 0.6286501999999999 order = 50-0.001
CV AUC Score 0.6107094285714285 order = 50-0.001
Train AUC Score 0.65043684 order = 50-0.01
CV AUC Score 0.6286961799999999 order = 50-0.01
Train AUC Score 0.6968785000000001 order = 50-0.1
CV AUC Score 0.6728647 order = 50-0.1
Train AUC Score 0.71315756 order = 50-0.2
CV AUC Score 0.6830491200000001 order = 50-0.2
Train AUC Score 0.7231799800000001 order = 50-0.3
CV AUC Score 0.68976548 order = 50-0.3
Train AUC Score 0.624669 order = 75-0.0001
CV AUC Score 0.608529 order = 75-0.0001
Train AUC Score 0.6286501999999999 order = 75-0.001
CV AUC Score 0.6107094285714285 order = 75-0.001
Train AUC Score 0.6586104666666667 order = 75-0.01
CV AUC Score 0.6363422666666667 order = 75-0.01
Train AUC Score 0.7087200666666668 order = 75-0.1
CV AUC Score 0.6824972266666667 order = 75-0.1
Train AUC Score 0.7264636 order = 75-0.2
CV AUC Score 0.6910851733333334 order = 75-0.2
Train AUC Score 0.7379466266666667 order = 75-0.3
CV AUC Score 0.6965984800000001 order = 75-0.3
Train AUC Score 0.624669 order = 100-0.0001
CV AUC Score 0.608529 order = 100-0.0001
Train AUC Score 0.6286501999999999 order = 100-0.001
```

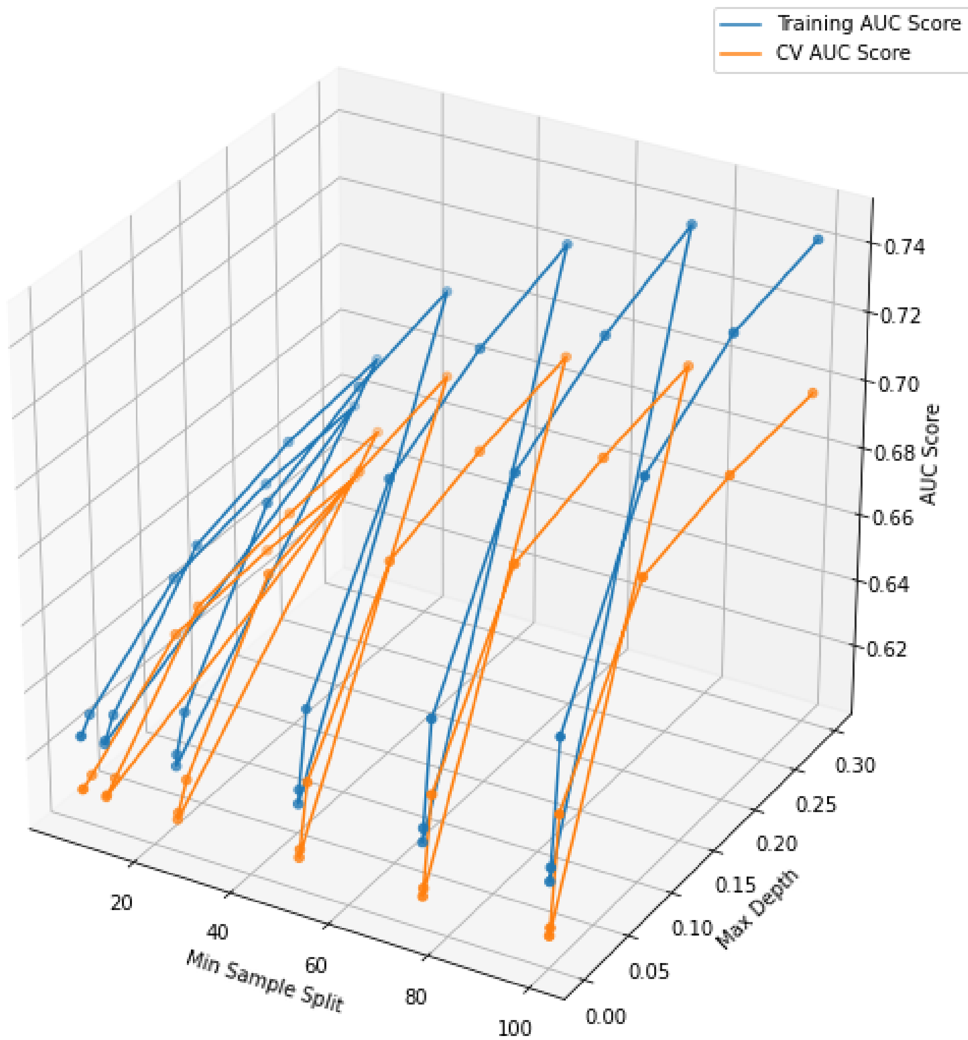
```
CV AUC Score 0.6107094285714285 order = 100-0.001
Train AUC Score 0.6644716599999999 order = 100-0.01
CV AUC Score 0.64204646 order = 100-0.01
Train AUC Score 0.71768578 order = 100-0.1
CV AUC Score 0.6890147600000001 order = 100-0.1
Train AUC Score 0.7366363 order = 100-0.2
CV AUC Score 0.6959687 order = 100-0.2
Train AUC Score 0.7427967294117647 order = 100-0.3
CV AUC Score 0.6983023882352941 order = 100-0.3
```

[39]:
```python
_,ax = plt.subplots(1,1,figsize=(20,10))
ax.plot(list(train_auc_score.keys()), list(train_auc_score.values()), label='Train AUC Score')
ax.plot(list(cv_auc_score.keys()), list(cv_auc_score.values()), label='CV AUC Score')
ax.legend()
plt.show()
```



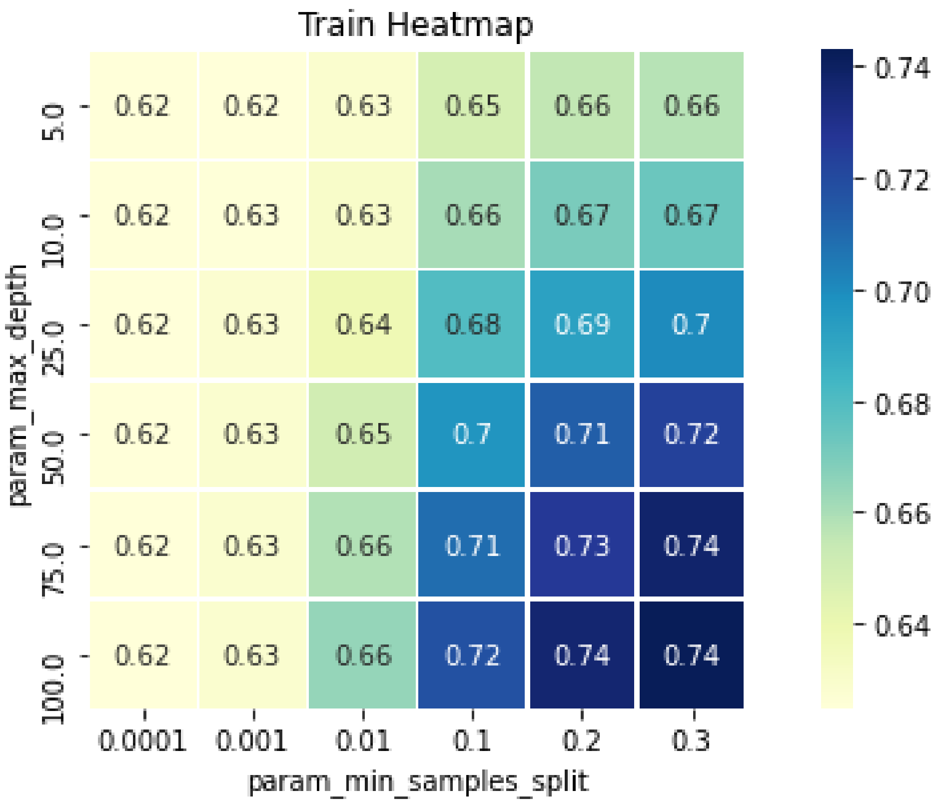
[40]:
```python
x_axis = []
y_axis = []
for key in train_auc_score.keys():
    temp = list(map(float, key.split('-')))
    x_axis.append(temp[0])
    y_axis.append(temp[1])
# 3D Plot
subplot_args = {'projection':'3d'}
fig, ax = plt.subplots(1, 1, figsize=(10,10), subplot_kw=subplot_args)
ax.scatter3D(x_axis,y_axis,  list(train_auc_score.values()))
```

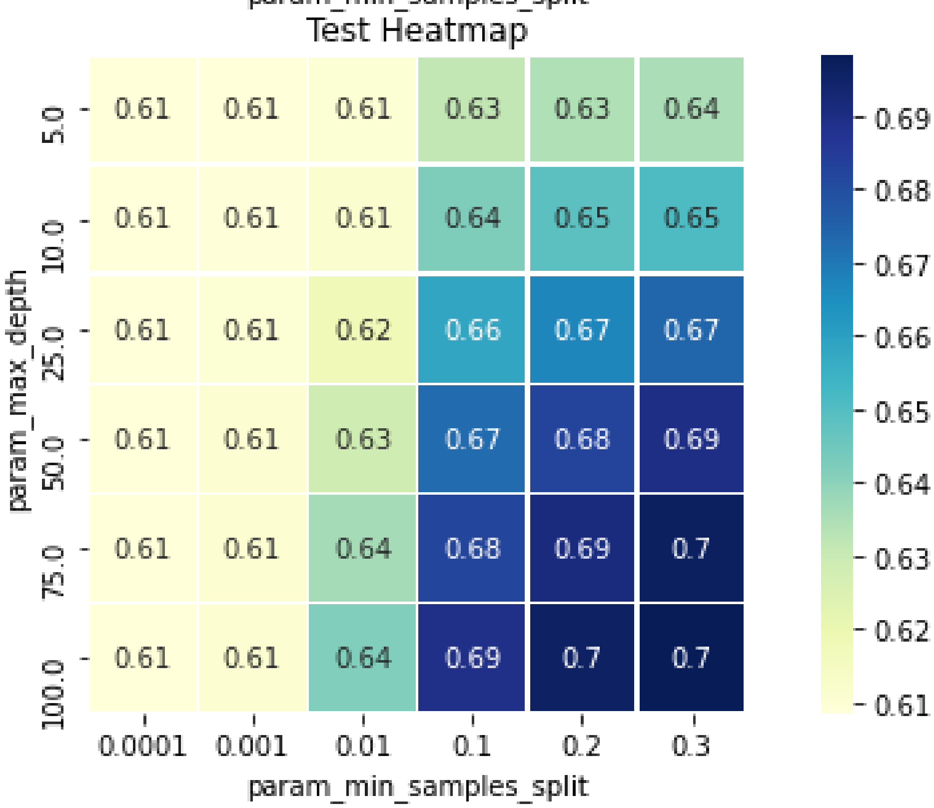
```
ax.plot3D( x_axis,y_axis, list(train_auc_score.values()), label='Training AUC␣
↪Score')
ax.scatter3D(x_axis,y_axis,  list(cv_auc_score.values()))
ax.plot3D(x_axis,y_axis,  list(cv_auc_score.values()), label='CV AUC Score')
ax.legend()
ax.set_xlabel('Min Sample Split')
ax.set_ylabel('Max Depth')
ax.set_zlabel('AUC Score')
plt.show()
```



[41]:
```
# Heatmap
```

```
temp_df = pd.DataFrame(np.array([x_axis,y_axis,list(train_auc_score.values())]).
↪T, columns=['param_max_depth', 'param_min_samples_split',␣
↪'mean_train_score'])
_, ax = plt.subplots(1, 2, figsize=(15, 15))
sns.heatmap(data=temp_df.pivot('param_max_depth', 'param_min_samples_split',␣
↪'mean_train_score'),  annot=True, linewidths=.5, square=True, ax =ax[0],␣
↪cmap="YlGnBu")
ax[0].set_title('Train Heatmap')
temp_df = pd.DataFrame(np.array([x_axis,y_axis,list(cv_auc_score.values())]).T,␣
↪columns=['param_max_depth', 'param_min_samples_split', 'mean_cv_score'])
sns.heatmap(data=temp_df.pivot('param_max_depth', 'param_min_samples_split',␣
↪'mean_cv_score'), annot=True, linewidths=.5, square=True, ax =ax[1],␣
↪cmap="YlGnBu")
ax[1].set_title('Test Heatmap')
plt.show()
```


Train Heatmap
param_max_depth
5.0
10.0
25.0
50.0
75.0
100.0
0.62 0.62 0.63 0.65 0.66 0.66
0.62 0.63 0.63 0.66 0.67 0.67
0.62 0.63 0.64 0.68 0.69 0.7
0.62 0.63 0.65 0.7 0.71 0.72
0.62 0.63 0.66 0.71 0.73 0.74
0.62 0.63 0.66 0.72 0.74 0.74
0.0001 0.001 0.01 0.1 0.2 0.3
param_min_samples_split
0.74
0.72
0.70
0.68
0.66
0.64

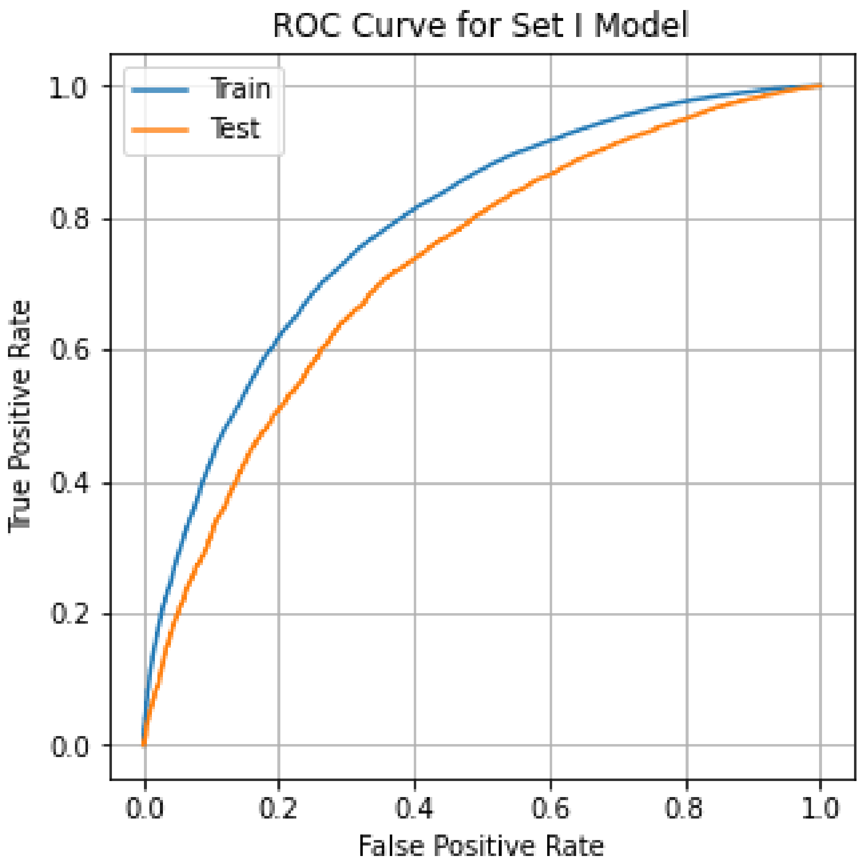

Test Heatmap
param_max_depth
5.0
10.0
25.0
50.0
75.0
100.0
0.61 0.61 0.61 0.63 0.63 0.64
0.61 0.61 0.61 0.64 0.65 0.65
0.61 0.61 0.62 0.66 0.67 0.67
0.61 0.61 0.63 0.67 0.68 0.69
0.61 0.61 0.64 0.68 0.69 0.7
0.61 0.61 0.64 0.69 0.7 0.7
0.0001 0.001 0.01 0.1 0.2 0.3
param_min_samples_split
0.69
0.68
0.67
0.66
0.65
0.64
0.63
0.62
0.61

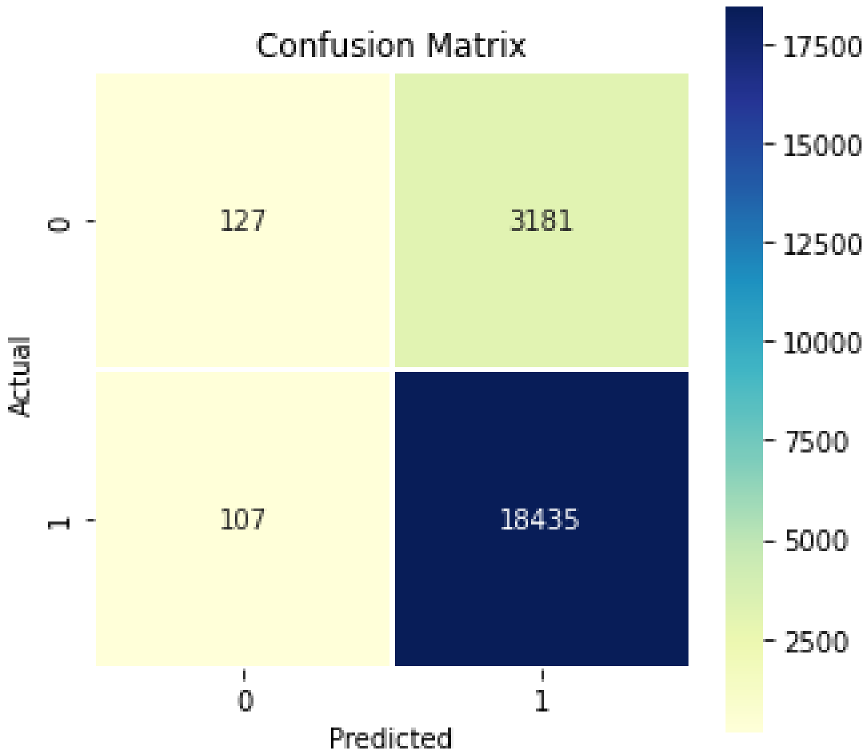
1.6 Training Best Model

1.6.1 Set I

[52]:
```
best_learning_rate = 0.3
best_estimators    = 75
param_dist = {
    'objective':'binary:logistic',
    'n_jobs':-1,
    'n_estimators':best_estimators,
    'learning_rate':best_learning_rate,
    'random_state':42
    }

clf = xgb.XGBClassifier(**param_dist)

clf.fit(x_train_set_i, y_train,
        eval_set=[(x_train_set_i, y_train), (x_test_set_i, y_test)],
        eval_metric='auc',
        early_stopping_rounds=10,
        verbose=False)

eval_result = clf.evals_result()
train_auc = np.mean(eval_result['validation_0']['auc'])
print('Train AUC Score {} '.format(train_auc))
test_auc    = np.mean(eval_result['validation_1']['auc'])
print('Test AUC Score {} '.format(test_auc))
table.append_row(['TFIDF + Sentiment Features','Decision Tree','Learning Rate =␣
 ↪{} Best Estimators = {}'.format(best_learning_rate, best_estimators),␣
 ↪test_auc])

# Plot ROC Curve
_, ax = plt.subplots(1,1,figsize=(5,5))
plot_roc_curve(clf, x_train_set_i, y_train, ax=ax, label='Train')
plot_roc_curve(clf, x_test_set_i, y_test, ax=ax, label='Test')
ax.grid()
ax.set_title('ROC Curve for Set I Model')
ax.legend()
plt.show()

# Confusion Matrix
_, ax = plt.subplots(1,1,figsize=(5,5))
sns.heatmap(confusion_matrix(y_test, y_pred), fmt='.5g',annot=True, linewidths=.
 ↪5, square=True, ax =ax, cmap="YlGnBu")
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
ax.set_title('Confusion Matrix')
plt.show()
```

```
Train AUC Score 0.74363872
Test AUC Score 0.7058297733333334
```



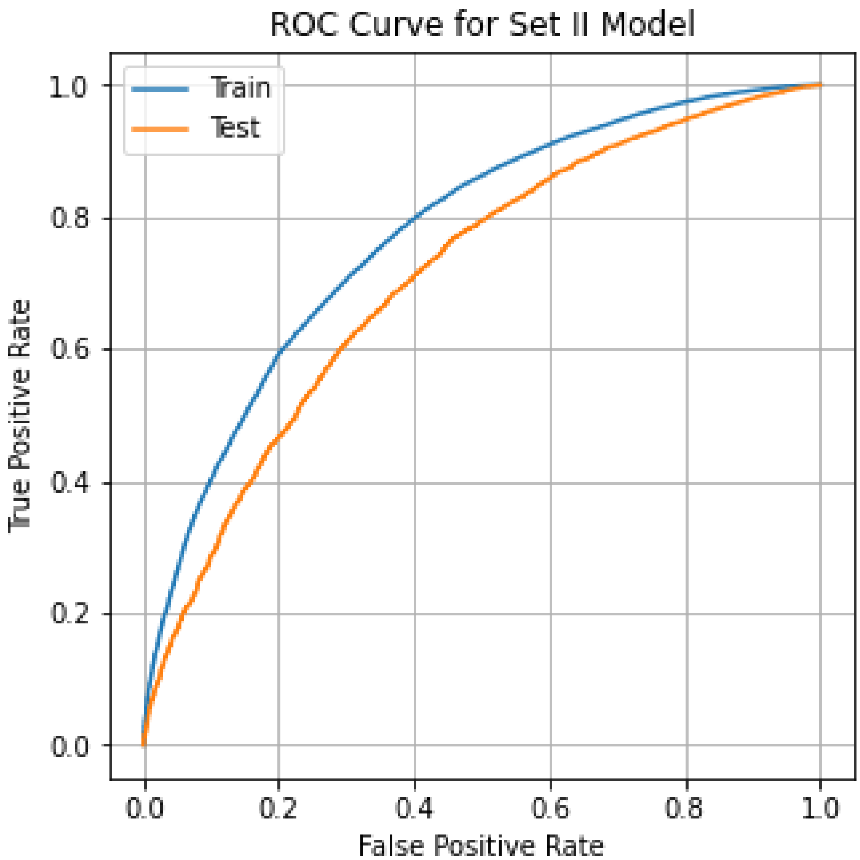


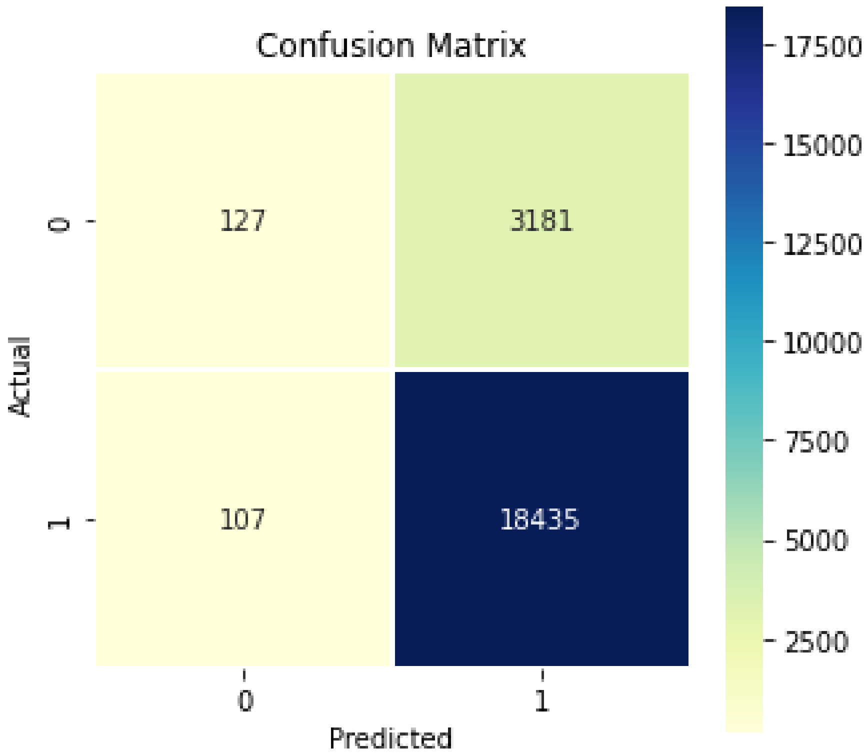
1.6.2 Set II

[53]:
```
best_learning_rate = 0.3
best_estimators    = 75
param_dist = {
    'objective':'binary:logistic',
    'n_jobs':-1,
    'n_estimators':best_estimators,
    'learning_rate':best_learning_rate,
    'random_state':42
    }

clf = xgb.XGBClassifier(**param_dist)

clf.fit(x_train_set_ii, y_train,
        eval_set=[(x_train_set_ii, y_train), (x_test_set_ii, y_test)],
        eval_metric='auc',
        early_stopping_rounds=10,
        verbose=False)

eval_result = clf.evals_result()
train_auc = np.mean(eval_result['validation_0']['auc'])
```

```
print('Train AUC Score {} '.format(train_auc))
test_auc    = np.mean(eval_result['validation_1']['auc'])
print('Test AUC Score {} '.format(test_auc))
table.append_row(['resp encoding + TFIDF','Decision Tree','Learning Rate = {}␣
↪Best Estimators = {}'.format(best_learning_rate, best_estimators), test_auc])

# Plot ROC Curve
_, ax = plt.subplots(1,1,figsize=(5,5))
plot_roc_curve(clf, x_train_set_ii, y_train, ax=ax, label='Train')
plot_roc_curve(clf, x_test_set_ii, y_test, ax=ax, label='Test')
ax.grid()
ax.set_title('ROC Curve for Set II Model')
ax.legend()
plt.show()

# Confusion Matrix
_, ax = plt.subplots(1,1,figsize=(5,5))
sns.heatmap(confusion_matrix(y_test, y_pred), fmt='.5g',annot=True, linewidths=.
↪5, square=True, ax =ax, cmap="YlGnBu")
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
ax.set_title('Confusion Matrix')
plt.show()
```

```
Train AUC Score 0.7379466266666667
Test AUC Score 0.6973729999999999
```


ROC Curve for Set II Model
Train
Test
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate



[54]:
```
table.column_headers=['Method', 'Model', 'Hyper Parameters', 'Auc Score']
print(table)
```

```
+--------------------+----------------+-------------------------------+------------+
|       Method       |     Model      |        Hyper Parameters       | Auc Score  |
+--------------------+----------------+-------------------------------+------------+
| TFIDF + Sentiment  | Decision Tree  | Learning Rate = 0.3 Best Est  |   0.706    |
|      Features      |                |          imators = 75         |            |
+--------------------+----------------+-------------------------------+------------+
| resp encoding + T  | Decision Tree  | Learning Rate = 0.3 Best Est  |   0.697    |
|       FIDF         |                |          imators = 75         |            |
+--------------------+----------------+-------------------------------+------------+
```