Assignment11

July 19, 2020

1 Assignment 8: DT

Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets

Set 1: categorical, numerical features + preprocessed_eassay (TFIDF)

Set 2: categorical, numerical features + preprocessed_eassay (TFIDF W2V)

The hyper paramter tuning (best `depth` in range [1, 5, 10, 50], and the best `min_samples_split` in range [5, 10, 100, 500])

Find the best hyper parameter which will give the maximum AUC value

find the best hyper paramter using k-fold cross validation(use gridsearch cv or randomsearch cv)/simple cross validation data(you can write your own for loops refer sample solution)

```
</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>min_sample_spli
        <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/ger
<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 fi
<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.c
<img src='https://i.imgur.com/IdN5Ctv.png' width=300px></li>
<li>Once after you plot the confusion matrix with the test data, get all the `false positive da
    <ul>
        <li> Plot the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) wi
        <li> Plot the box plot with the `price` of these `false positive data points`</li>
        <li> Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `fal
    </ul>
    </ul>
</li>
```

Task 2: For this task consider set-1 features. Select all the features which are having non-zero feature importance.You can get the feature importance using 'feature_importances_' (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html), discard the all other remaining features and then apply any of the model of you choice i.e. (Dession tree, Logistic Regression, Linear SVM), you need to do hyperparameter tuning corresponding to the model you selected and procedure in step 2 and step 3 Note: when you want to find the feature importance make sure you don't use max_depth parameter keep it None.

You need to summarize the results at the end of the notebook, summarize it in the table format

1.1 Task 1

1.1.1 Imports

[11]:
```
!pip install beautifultable
```

```
Requirement already satisfied: beautifultable in /usr/local/lib/python3.6/dist-
packages (0.8.0)
```

[12]:
```
import pandas as pd
import numpy as np
from scipy.sparse import hstack
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score, confusion_matrix,plot_roc_curve
from sklearn.model_selection import GridSearchCV

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

import pickle
import seaborn as sns
from tqdm import tqdm
from wordcloud import WordCloud
from beautifultable import BeautifulTable
table = BeautifulTable()
```

1.1.2 Loading Data

[13]:
```
data = pd.read_csv('drive/My Drive/Colab Notebooks/AppliedAICourse/Assignment/
 ↪preprocessed_data.csv')
data.head()
```

[13]:
```
  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]
```

[14]:
```
data.shape
```

[14]:
```
(109248, 9)
```

[15]:
```
', '.join(data.columns.tolist())
```

[15]:
```
'school_state, teacher_prefix, project_grade_category,
teacher_number_of_previously_posted_projects, project_is_approved,
clean_categories, clean_subcategories, essay, price'
```

[16]:
```
y = data['project_is_approved']
x = data.drop(['project_is_approved'], axis=1)
```

1.1.3 Splitting Data

[17]:
```
# please write all the code with proper documentation, and proper titles for␣
 ↪each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in␣
 ↪debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the␣
 ↪reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

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)

print(x_train.shape)
print(x_cv.shape)
print(x_test.shape)
print(y_train.shape)
print(y_cv.shape)
print(y_test.shape)
```

```
(69918, 8)
```

```
(17480, 8)
(21850, 8)
(69918,)
(17480,)
(21850,)
```

1.1.4 Defining Reusable Functions

[18]:
```
x_train_dict = {}
x_cv_dict = {}
x_test_dict = {}

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, x_test, x_cv, col):
    # 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=(1,5),min_df=10,norm='l2')
            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:
            vec = CountVectorizer(lowercase=False)
            x_train_val = vec.fit_transform(x_train[col]).toarray()
            x_cv_val    = vec.transform(x_cv[col]).toarray()
            x_test_val  = vec.transform(x_test[col]).toarray()

    return (x_train_val, x_cv_val, x_test_val)
```

1.1.5 Preparing Set I Data

[19]:
```
for col in x_train.columns:
    print('Transforming', col)
    result = transforming(x_train, x_test, x_cv, col)
    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 [02:14<00:00, 521.01it/s]
100%|      | 21850/21850 [00:42<00:00, 513.67it/s]
```

```
100%|          | 17480/17480 [00:34<00:00, 512.22it/s]
```

[20]:
```
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']
))
print(x_train_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']
))
print(x_cv_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']
))
print(x_test_set_i.shape)
```

```
(69918, 259833)
(17480, 259833)
(21850, 259833)
```

[21]:
```
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, 401)
(17480, 401)
(21850, 401)
```

1.1.6 Modeling

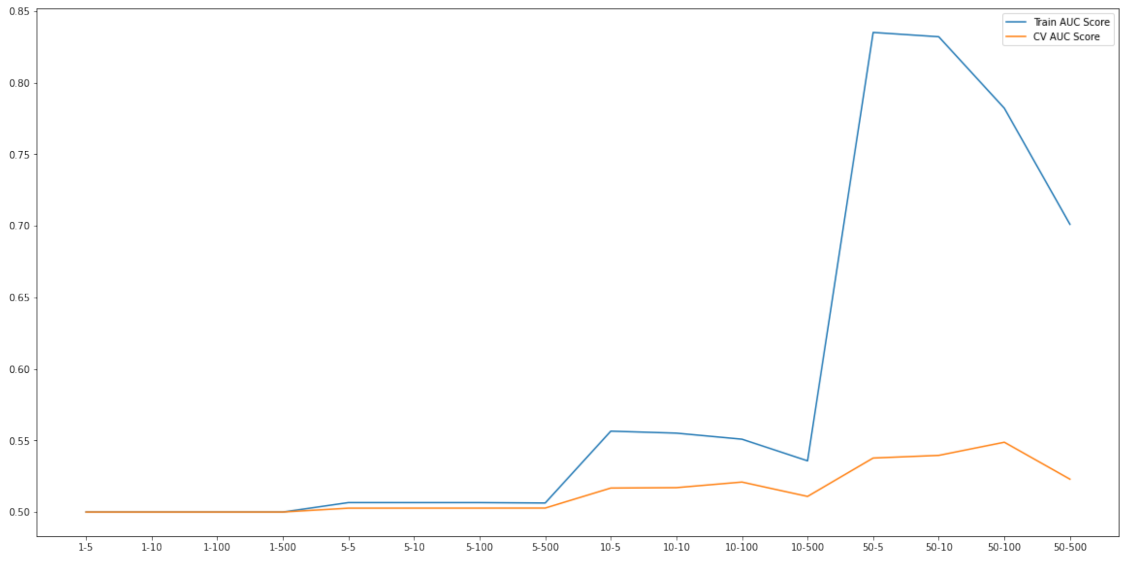
Set I

[22]:
```
# Hyperparameter Tuning
train_auc_score = {}
cv_auc_score = {}
max_depth=[1, 5, 10, 50]
min_sample_split= [5, 10, 100, 500]
```

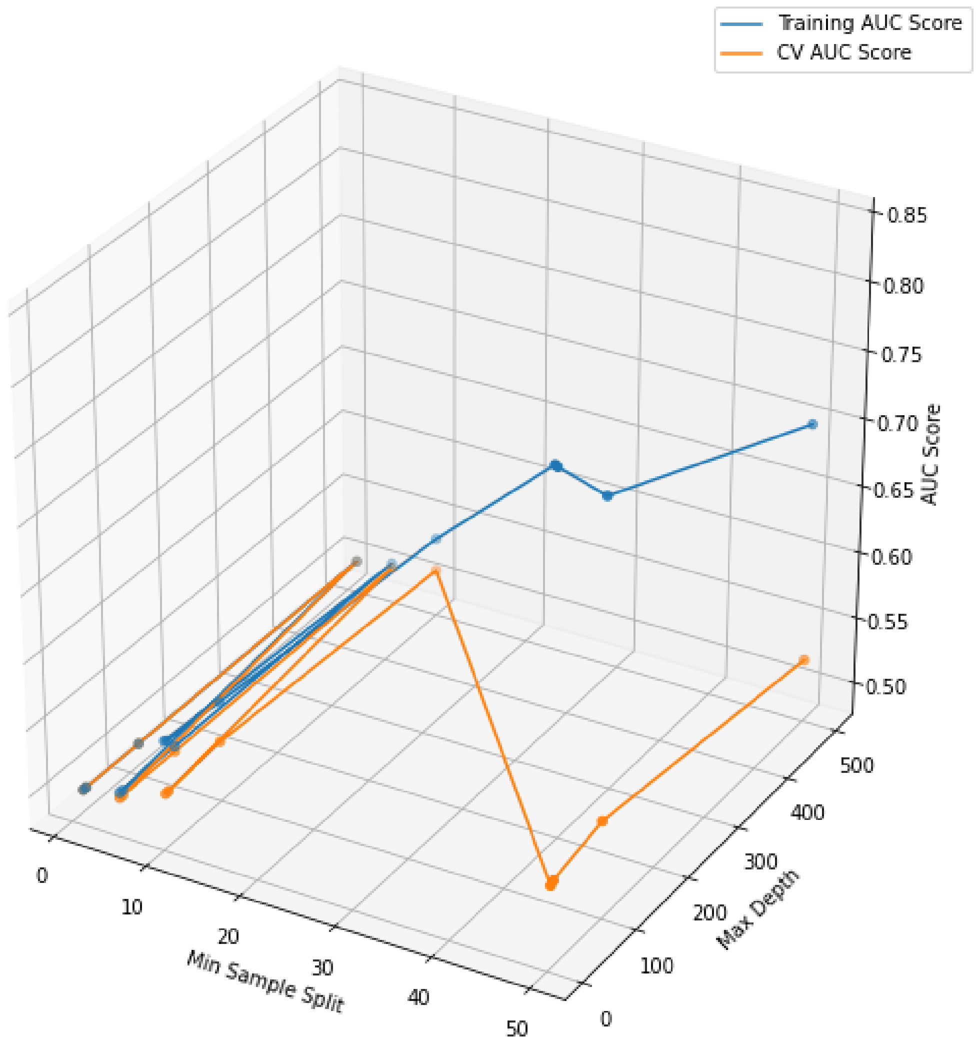
```python
for depth in max_depth:
    for sample in min_sample_split:
        order = '{}-{}'.format(depth, sample)
        clf = DecisionTreeClassifier(max_depth=depth, min_samples_split=sample)
        clf.fit(x_train_set_i, y_train)

        # Predict Train
        y_pred = clf.predict(x_train_set_i)
        auc_score = roc_auc_score(y_train, y_pred)
        print('For',order,'Training AUC Score',auc_score)
        train_auc_score[order] = auc_score
        # Predict CV
        y_pred = clf.predict(x_cv_set_i)
        auc_score = roc_auc_score(y_cv, y_pred)
        cv_auc_score[order] = auc_score
        print('For',order,'CV AUC Score',auc_score)
```

```
For 1-5 Training AUC Score 0.5
For 1-5 CV AUC Score 0.5
For 1-10 Training AUC Score 0.5
For 1-10 CV AUC Score 0.5
For 1-100 Training AUC Score 0.5
For 1-100 CV AUC Score 0.5
For 1-500 Training AUC Score 0.5
For 1-500 CV AUC Score 0.5
For 5-5 Training AUC Score 0.5065955508768332
For 5-5 CV AUC Score 0.5027061771286626
For 5-10 Training AUC Score 0.5065786962814278
For 5-10 CV AUC Score 0.5027398857513282
For 5-100 Training AUC Score 0.5065786962814278
For 5-100 CV AUC Score 0.5027398857513282
For 5-500 Training AUC Score 0.506295329888682
For 5-500 CV AUC Score 0.502773594373994
For 10-5 Training AUC Score 0.5564978742219154
For 10-5 CV AUC Score 0.5167566747593679
For 10-10 Training AUC Score 0.555109834530332
For 10-10 CV AUC Score 0.5170263437406933
For 10-100 Training AUC Score 0.5508626627531719
For 10-100 CV AUC Score 0.5209199379842845
For 10-500 Training AUC Score 0.5357622610650359
For 10-500 CV AUC Score 0.5109067324053151
For 50-5 Training AUC Score 0.8350112252608365
For 50-5 CV AUC Score 0.5377252973165465
For 50-10 Training AUC Score 0.8319931947935262
For 50-10 CV AUC Score 0.5395690329033088
For 50-100 Training AUC Score 0.78204569979547
For 50-100 CV AUC Score 0.5487433942497089
For 50-500 Training AUC Score 0.7009597409720443
For 50-500 CV AUC Score 0.5229389533150475
```

[23]:
```
_,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()
```

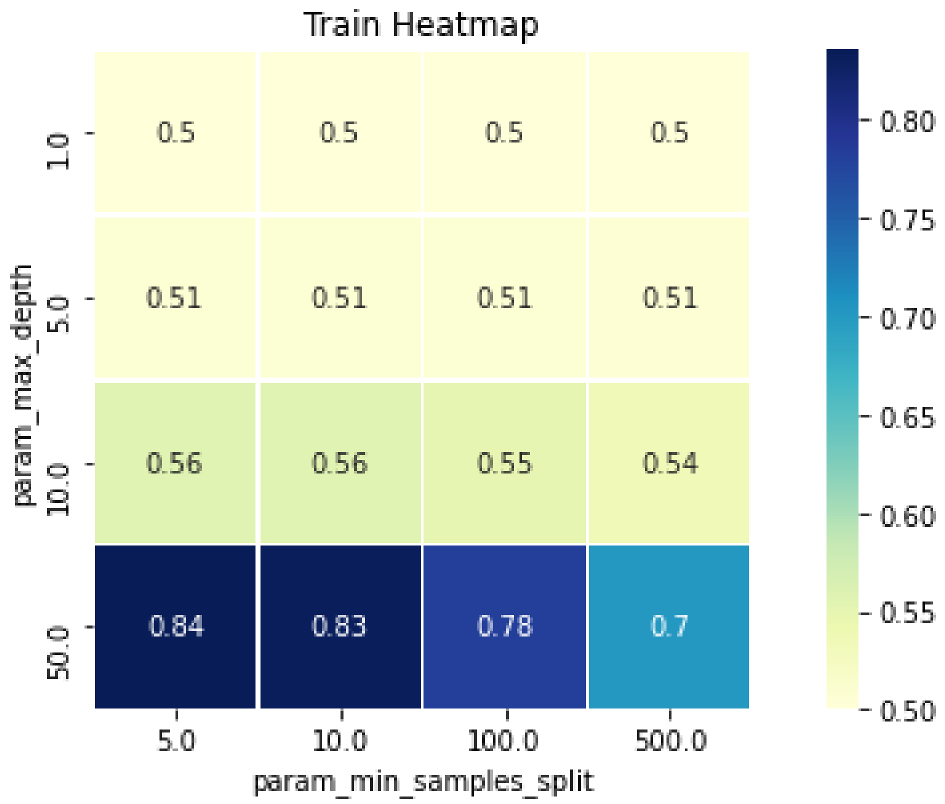


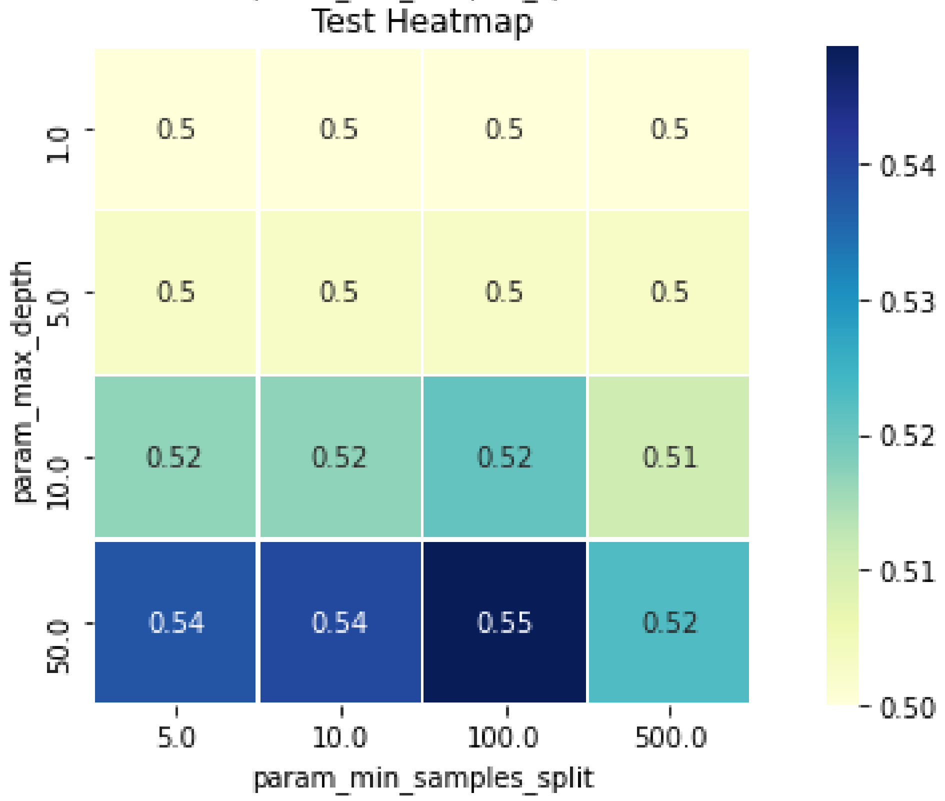
[24]:
```
x_axis = []
y_axis = []
for key in train_auc_score.keys():
    temp = list(map(int, key.split('-')))
    x_axis.append(temp[0])
    y_axis.append(temp[1])
# 3D Plot
subplot_args = {'projection':'3d'}
fig, ax = plt.subplots(1, 1, figsize=(10,10), subplot_kw=subplot_args)
ax.scatter3D(x_axis,y_axis,  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()
```



[25]:
```
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(2, 1, figsize=(10, 10))
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
1.0
5.0
10.0
50.0
0.5
0.5
0.5
0.5
0.51
0.51
0.51
0.51
0.56
0.56
0.55
0.54
0.84
0.83
0.78
0.7
5.0
10.0
100.0
500.0
param_min_samples_split
0.80
0.75
0.70
0.65
0.60
0.55
0.50

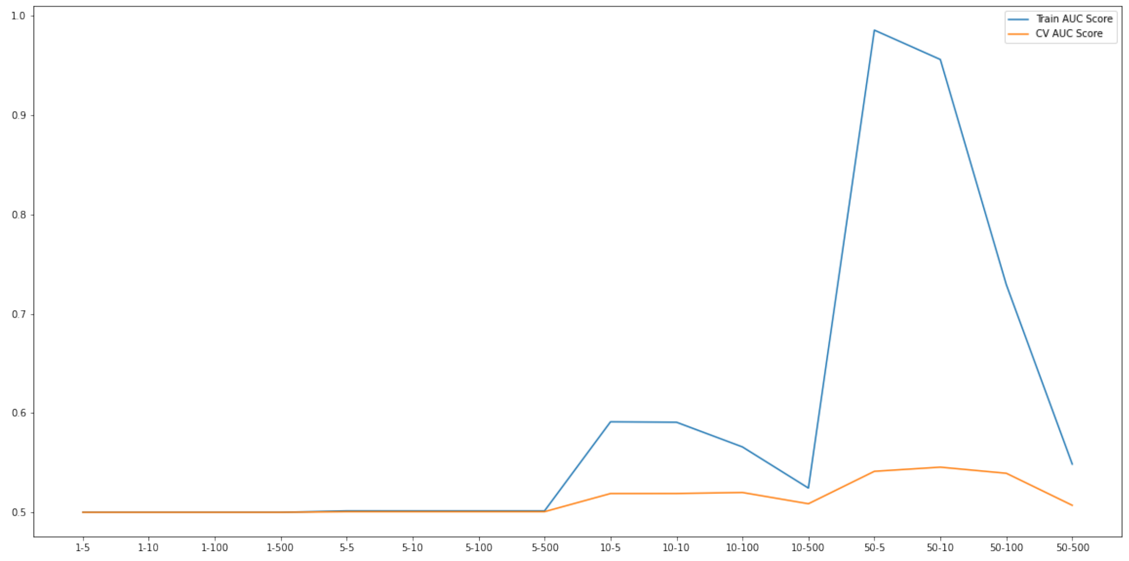

Test Heatmap
param_max_depth
1.0
5.0
10.0
50.0
0.5
0.5
0.5
0.5
0.5
0.5
0.5
0.5
0.52
0.52
0.52
0.51
0.54
0.54
0.55
0.52
5.0
10.0
100.0
500.0
param_min_samples_split
0.54
0.53
0.52
0.51
0.50

Set II

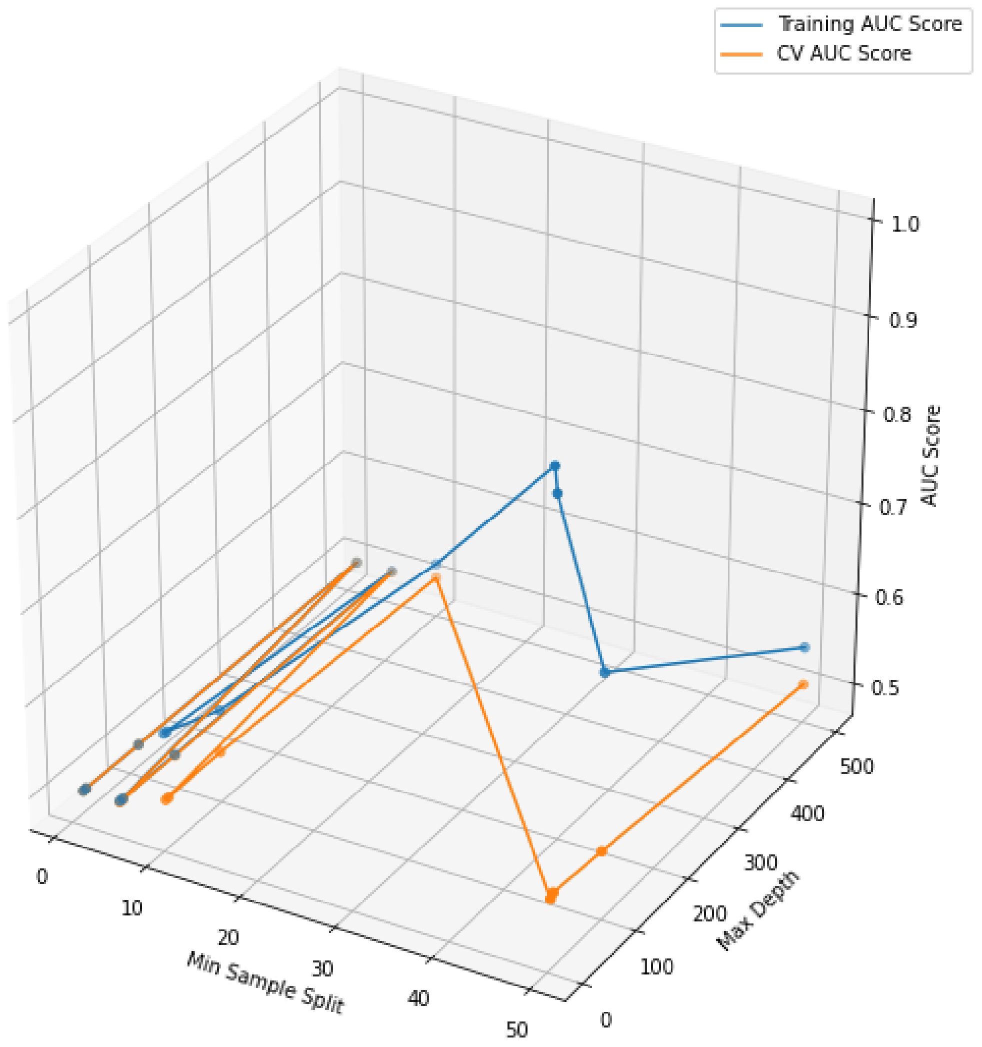
[26]:
```python
# Hyperparameter Tuning
train_auc_score = {}
cv_auc_score = {}
max_depth=[1, 5, 10, 50]
min_sample_split= [5, 10, 100, 500]

for depth in max_depth:
    for sample in min_sample_split:
        order = '{}-{}'.format(depth, sample)
        clf = DecisionTreeClassifier(max_depth=depth, min_samples_split=sample)
        clf.fit(x_train_set_ii, y_train)

        # Predict Train
        y_pred = clf.predict(x_train_set_ii)
        auc_score = roc_auc_score(y_train, y_pred)
        print('For',order,'Training AUC Score',auc_score)
        train_auc_score[order] = auc_score
        # Predict CV
        y_pred = clf.predict(x_cv_set_ii)
        auc_score = roc_auc_score(y_cv, y_pred)
        cv_auc_score[order] = auc_score
        print('For',order,'CV AUC Score',auc_score)
```

```
For 1-5 Training AUC Score 0.5
For 1-5 CV AUC Score 0.5
For 1-10 Training AUC Score 0.5
For 1-10 CV AUC Score 0.5
For 1-100 Training AUC Score 0.5
For 1-100 CV AUC Score 0.5
For 1-500 Training AUC Score 0.5
For 1-500 CV AUC Score 0.5
For 5-5 Training AUC Score 0.5014573875876694
For 5-5 CV AUC Score 0.5006073792059084
For 5-10 Training AUC Score 0.5014489602899667
For 5-10 CV AUC Score 0.5006073792059084
For 5-100 Training AUC Score 0.5014489602899667
For 5-100 CV AUC Score 0.5006073792059084
For 5-500 Training AUC Score 0.5014489602899667
For 5-500 CV AUC Score 0.5006073792059084
For 10-5 Training AUC Score 0.5911217527973028
For 10-5 CV AUC Score 0.5188738233150126
For 10-10 Training AUC Score 0.590713732128535
For 10-10 CV AUC Score 0.518914523261382
For 10-100 Training AUC Score 0.565822252869662
For 10-100 CV AUC Score 0.5199245339454999
For 10-500 Training AUC Score 0.5243651038922467
For 10-500 CV AUC Score 0.508584708775456
For 50-5 Training AUC Score 0.9856697826367091
For 50-5 CV AUC Score 0.5412736806257888
For 50-10 Training AUC Score 0.9560827176928486
For 50-10 CV AUC Score 0.5454439351744091
For 50-100 Training AUC Score 0.7293182663853186
For 50-100 CV AUC Score 0.5393011085692463
For 50-500 Training AUC Score 0.5484812598224047
For 50-500 CV AUC Score 0.5070244974709109
```

[27]:
```
_,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()
```

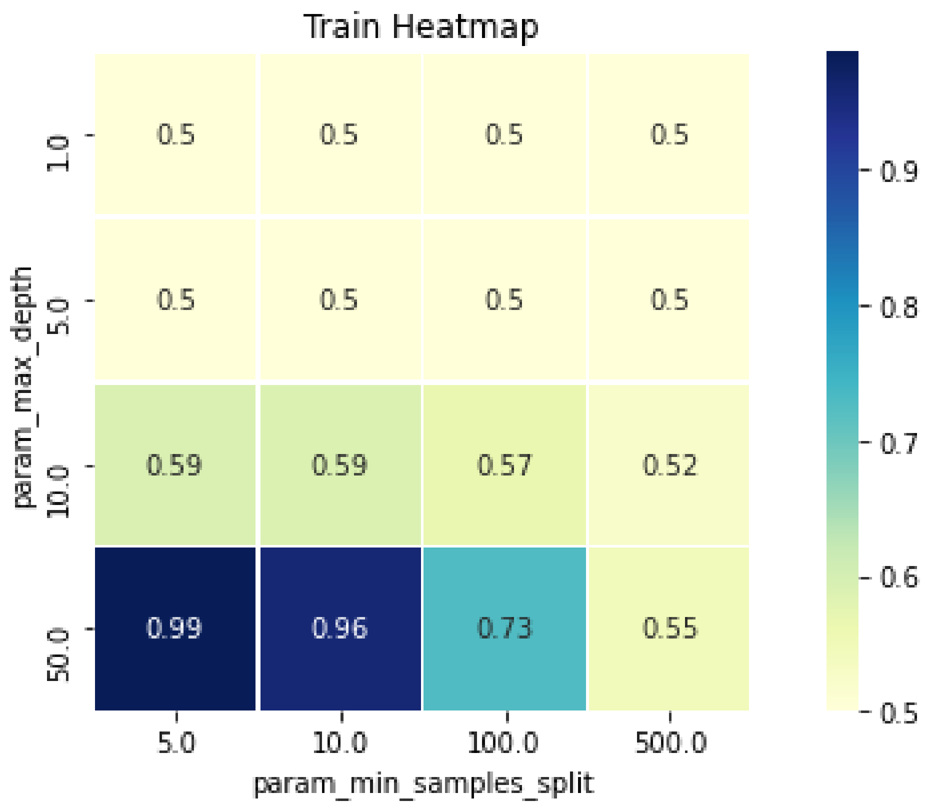


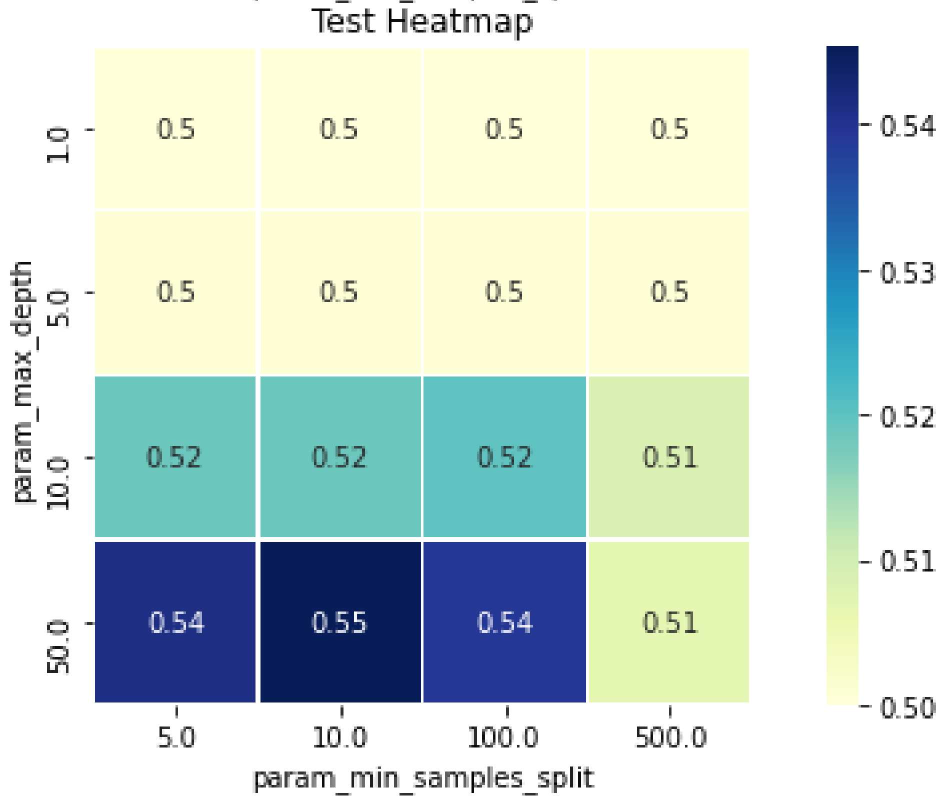
[28]:
```
x_axis = []
y_axis = []
for key in train_auc_score.keys():
    temp = list(map(int, key.split('-')))
    x_axis.append(temp[0])
    y_axis.append(temp[1])
# 3D Plot
subplot_args = {'projection':'3d'}
fig, ax = plt.subplots(1, 1, figsize=(10,10), subplot_kw=subplot_args)
ax.scatter3D(x_axis,y_axis,  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()
```



[29]:
```
# 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(2, 1, figsize=(10, 10))
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
1.0
5.0
10.0
50.0
param_min_samples_split
5.0
10.0
100.0
500.0
0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5
0.59 0.59 0.57 0.52
0.99 0.96 0.73 0.55
0.9
0.8
0.7
0.6
0.5


Test Heatmap
param_max_depth
1.0
5.0
10.0
50.0
param_min_samples_split
5.0
10.0
100.0
500.0
0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5
0.52 0.52 0.52 0.51
0.54 0.55 0.54 0.51
0.54
0.53
0.52
0.51
0.50

Observations

- Set I

  max_depth = 10 and min_sample_split = 500

- Set II

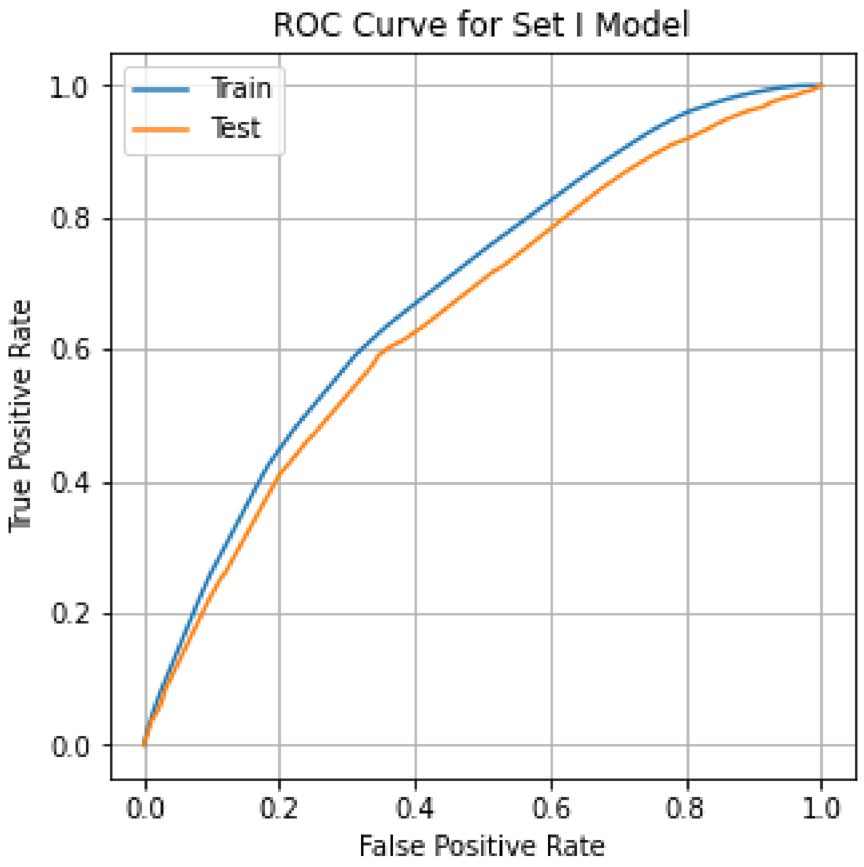
  Choosing max_depth = 10 and min_sample_split = 100

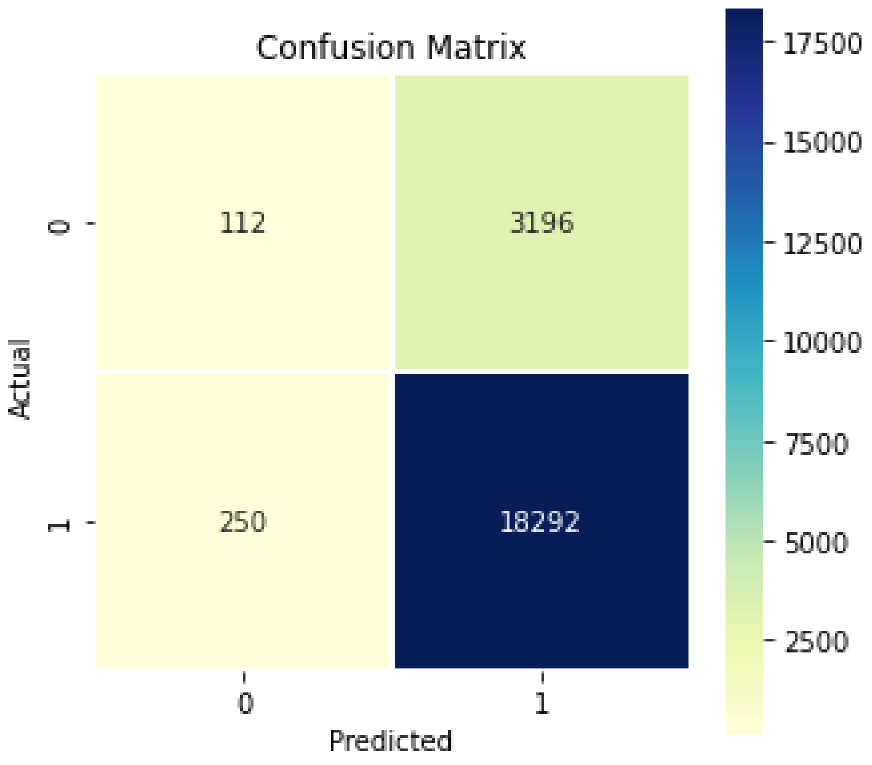
1.1.7 Training Best Model

Set I

[33]:
```
best_max_depth = 10
best_min_sample_split = 500

model_i = DecisionTreeClassifier(max_depth=best_max_depth,␣
 ↪min_samples_split=best_min_sample_split)
model_i.fit(x_train_set_i, y_train)

# Predicting Train
y_pred = model_i.predict(x_train_set_i)
train_auc_score = roc_auc_score(y_train, y_pred)
print('Train AUC Score',train_auc_score)

# Predicting Test
y_pred = model_i.predict(x_test_set_i)
test_auc_score = roc_auc_score(y_test, y_pred)
print('Test AUC Score', test_auc_score)
table.append_row(['TFIDF','Decision Tree','Max Depth = {} Min Sample Split =␣
 ↪{}'.format(best_max_depth, best_min_sample_split), test_auc_score])
# Plot ROC Curve
_, ax = plt.subplots(1,1,figsize=(5,5))
plot_roc_curve(model_i, x_train_set_i, y_train, ax=ax, label='Train')
plot_roc_curve(model_i, 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()
```

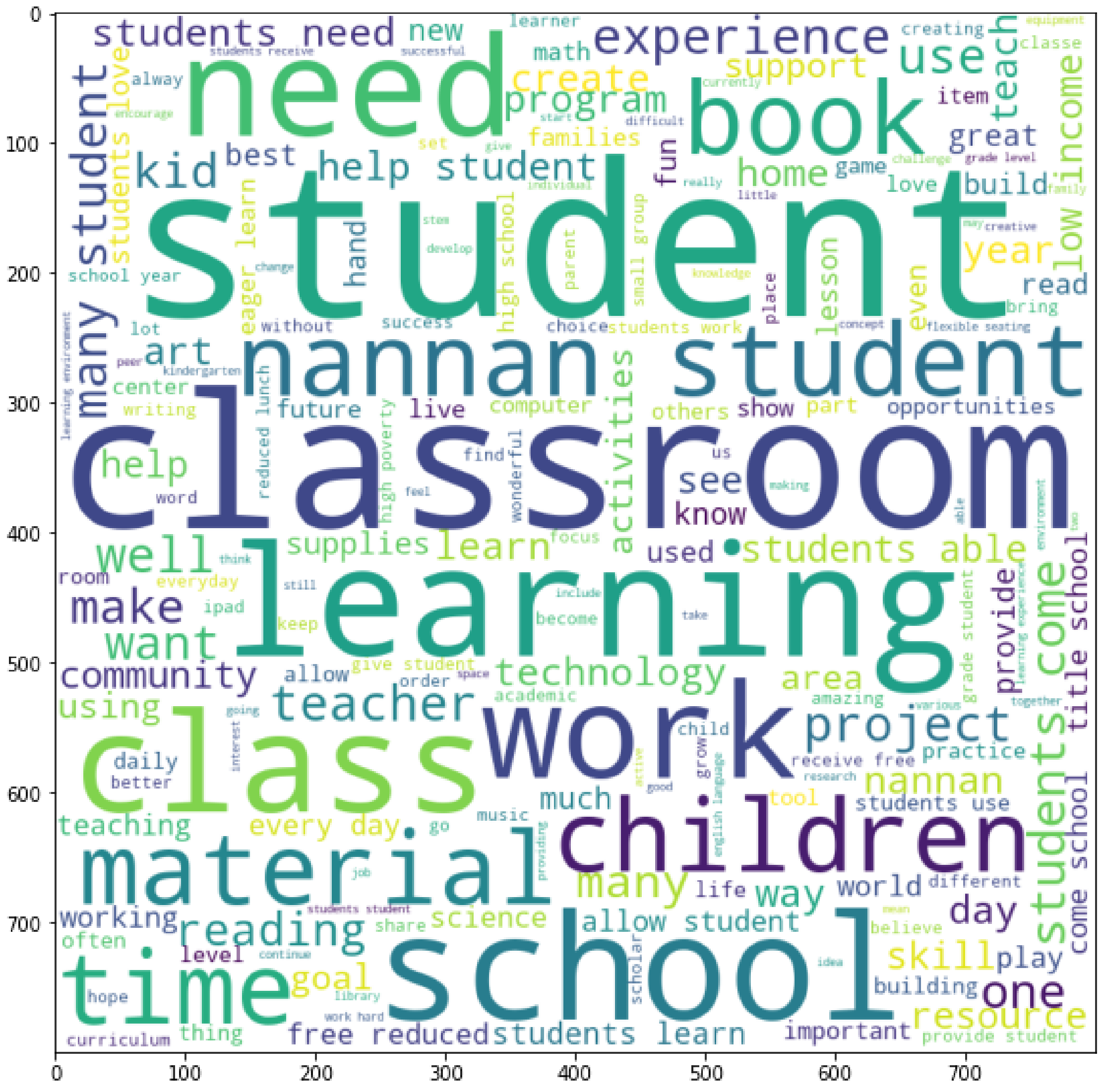
```
Train AUC Score 0.5358819984223926
Test AUC Score 0.5101872059602065
```





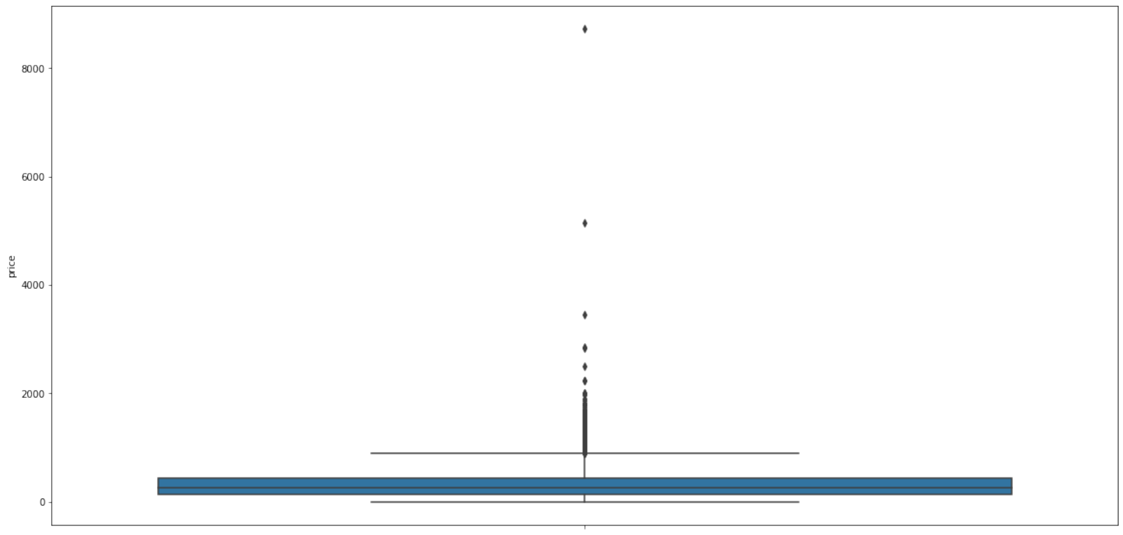
False Positive Data Point Analysis Essay WordCloud

[34]:
```
indices = np.where((y_pred - y_test ) > 0)
temp_df = x_test.iloc[indices]
text = ' '.join(temp_df['essay'].tolist())
_, ax = plt.subplots(1,1,figsize=(20,10))
wc = WordCloud(background_color="white", width=800, height=800,␣
 ↪random_state=42).generate(text)
ax.imshow(wc)
plt.show()
```



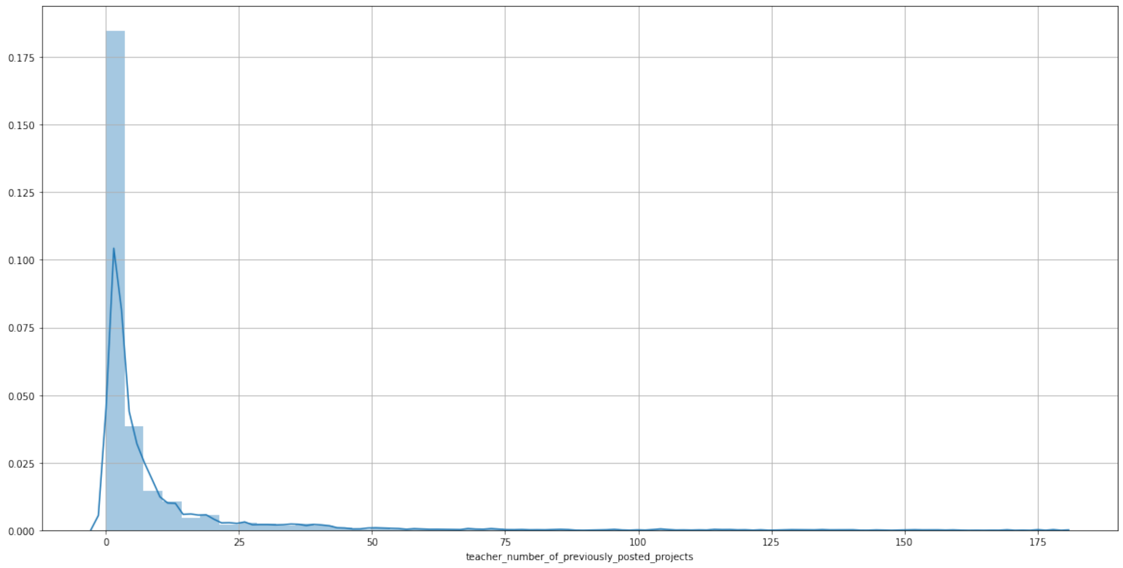
Price BoxPlot

```
[35]: indices = np.where((y_pred - y_test ) > 0)
      _, ax = plt.subplots(1, 1, figsize=(20,10))
      sns.boxplot(y='price',data=temp_df, ax=ax)
      plt.show()
```



teacher_number_of_previously_posted_projects PDF

[36]:
```
_, ax = plt.subplots(1, 1, figsize=(20,10))
sns.distplot(temp_df['teacher_number_of_previously_posted_projects'], ax=ax)
plt.grid()
plt.show()
```

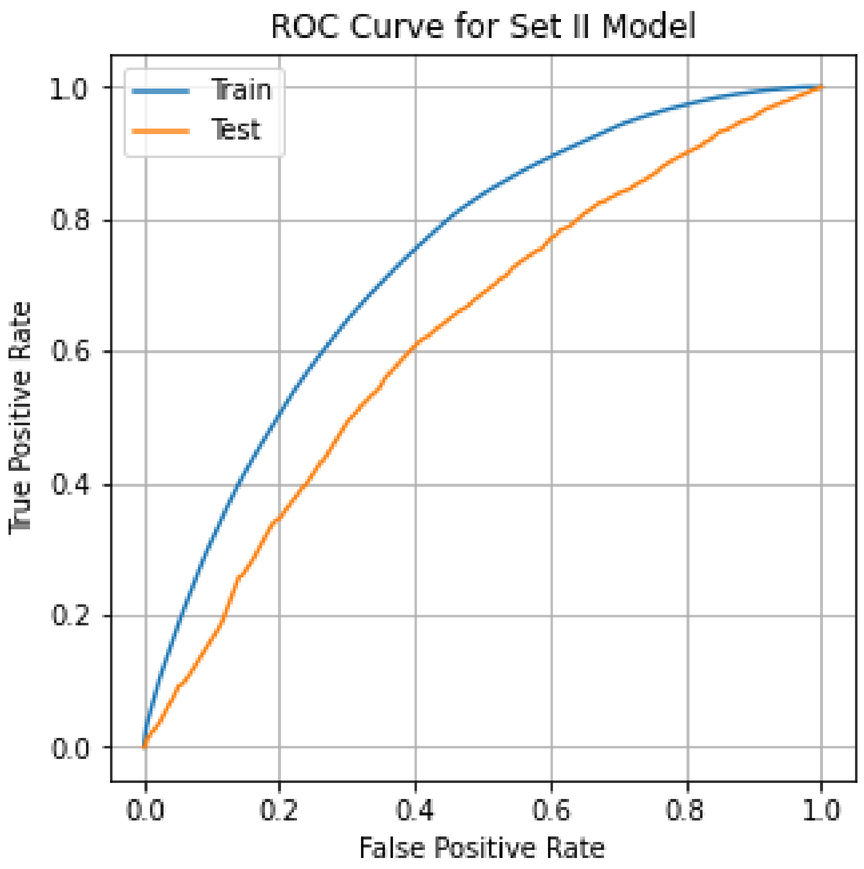


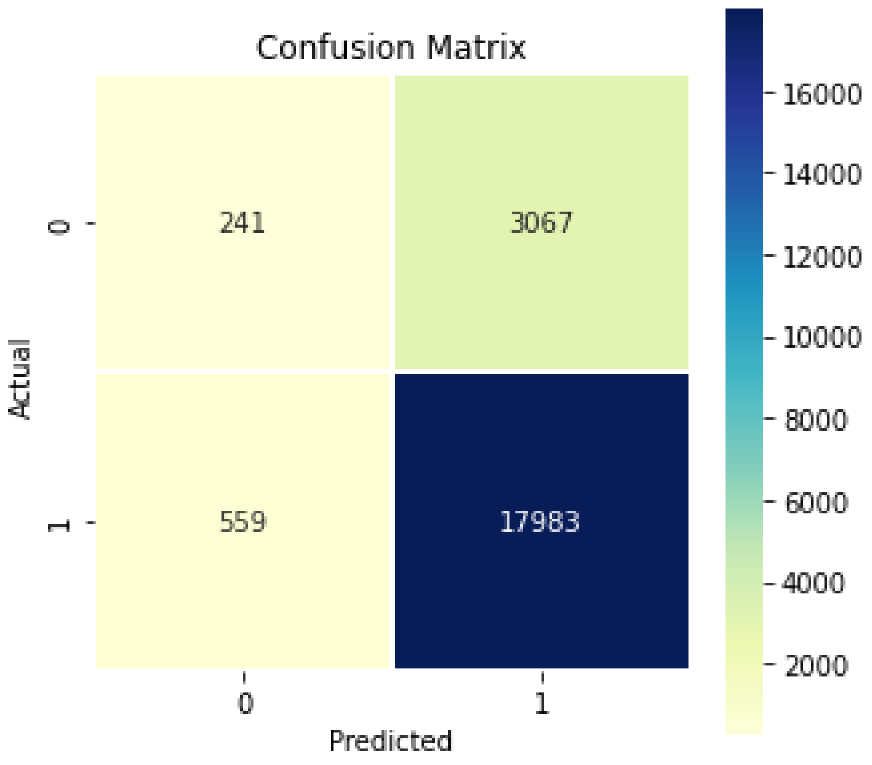
Set II

[37]:
```
best_max_depth = 10
best_min_sample_split = 100
```

```
model_ii = DecisionTreeClassifier(max_depth=best_max_depth,␣
↪min_samples_split=best_min_sample_split)
model_ii.fit(x_train_set_ii, y_train)
y_pred = model_ii.predict(x_test_set_ii)

# Predicting Train
y_pred = model_ii.predict(x_train_set_ii)
train_auc_score = roc_auc_score(y_train, y_pred)
print('Train AUC Score',train_auc_score)

# Predicting Test
y_pred = model_ii.predict(x_test_set_ii)
test_auc_score = roc_auc_score(y_test, y_pred)
print('Test AUC Score', test_auc_score)
table.append_row(['TFIDF W2V','Decision Tree','Max Depth = {} Min Sample Split␣
↪= {}'.format(best_max_depth, best_min_sample_split), test_auc_score])
# Plot ROC Curve
_, ax = plt.subplots(1,1,figsize=(5,5))
plot_roc_curve(model_ii, x_train_set_ii, y_train, ax=ax, label='Train')
plot_roc_curve(model_ii, 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()
```

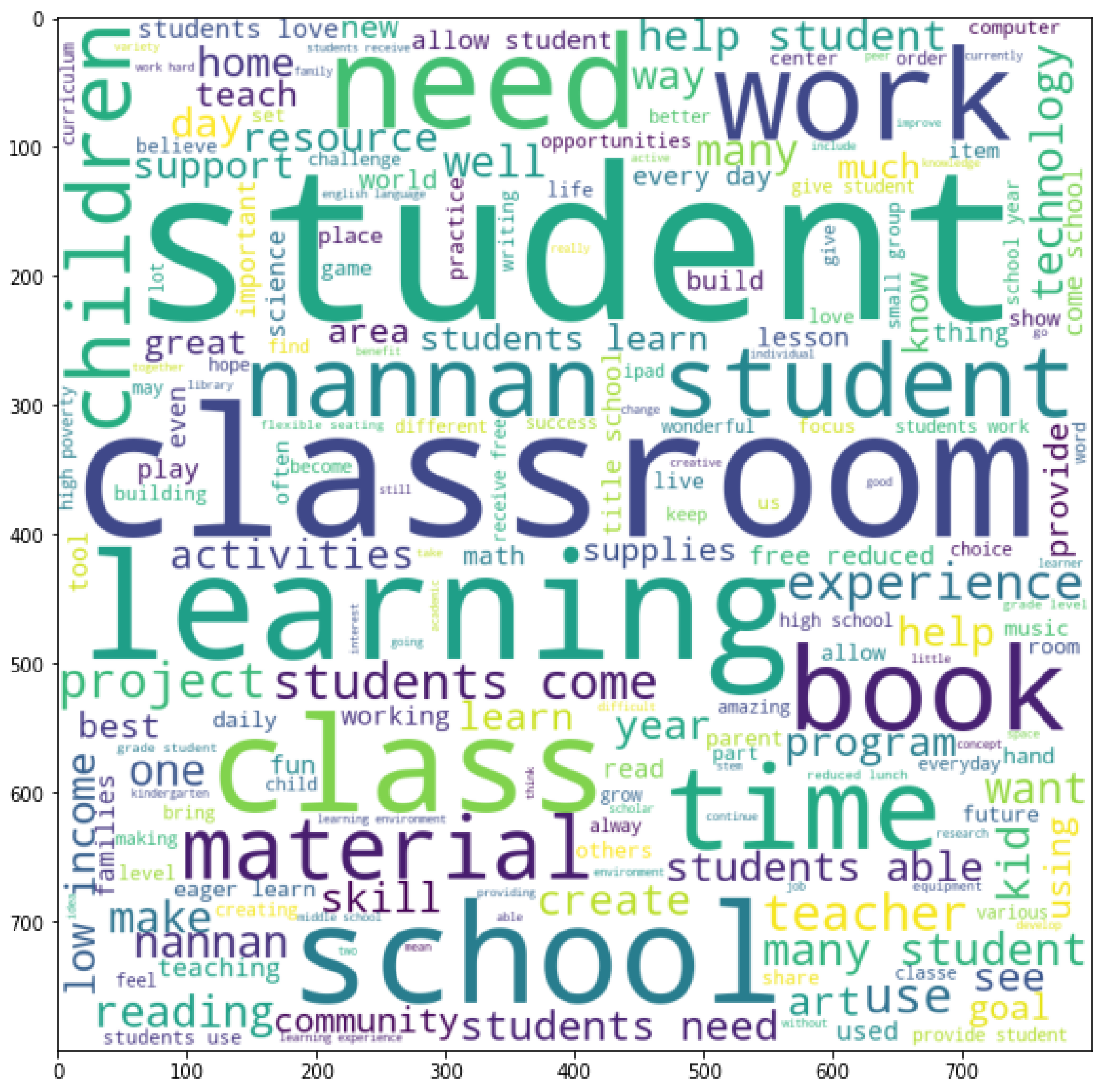
```
Train AUC Score 0.5659858819050512
Test AUC Score 0.5213529577023541
```


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



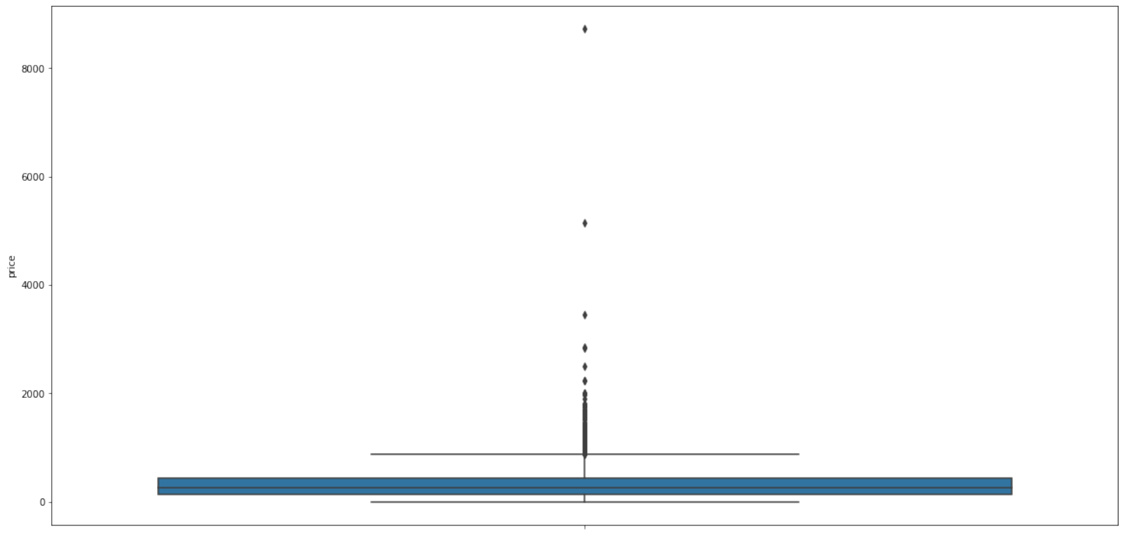
False Positive Data Point Analysis Essay Word Cloud

```
[38]: indices = np.where((y_pred - y_test ) > 0)
temp_df = x_test.iloc[indices]
text = ' '.join(temp_df['essay'].tolist())
_, ax = plt.subplots(1,1,figsize=(20,10))
wc = WordCloud(background_color="white", width=800, height=800,␣
 ↪random_state=42).generate(text)
ax.imshow(wc)
plt.show()
```



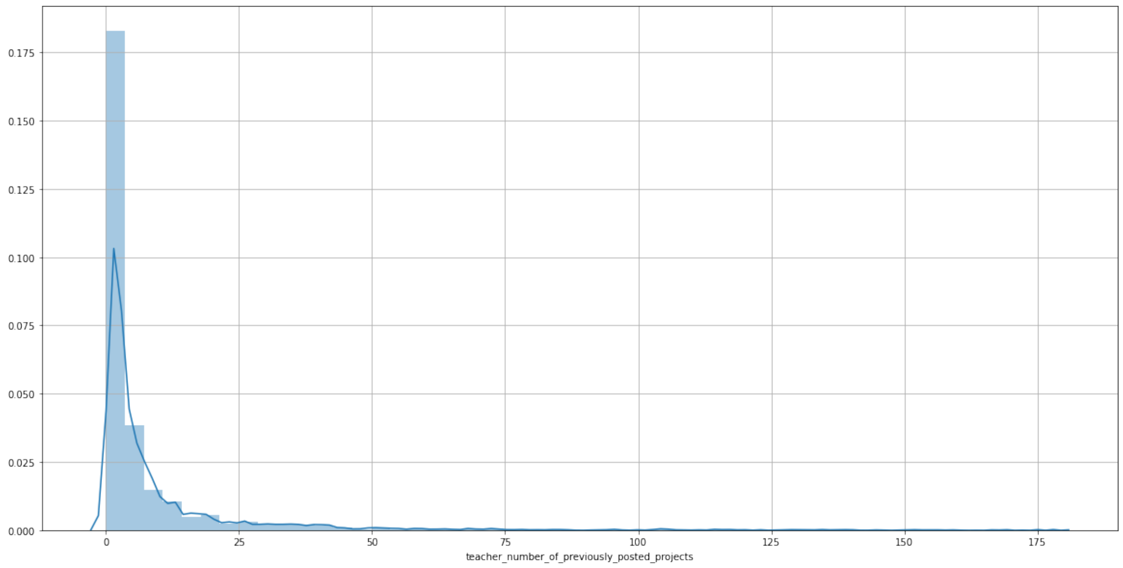
Prices BoxPlot

[39]:
```python
indices = np.where((y_pred - y_test ) > 0)
_, ax = plt.subplots(1, 1, figsize=(20,10))
sns.boxplot(y='price',data=temp_df, ax=ax)
plt.show()
```



teacher_number_of_previously_posted_projects PDF

[40]:
```
_, ax = plt.subplots(1, 1, figsize=(20,10))
sns.distplot(temp_df['teacher_number_of_previously_posted_projects'], ax=ax)
plt.grid()
plt.show()
```



1.2 Task 2

1.2.1 Data Preperation

[41]:
```
best_max_depth = None
best_min_sample_split = 500
model_i = DecisionTreeClassifier(max_depth=best_max_depth,␣
 ↪min_samples_split=best_min_sample_split)
model_i.fit(x_train_set_i, y_train)
indices = np.where(model_i.feature_importances_ > 0)[0]
print(indices.shape)
x_train_set_new = x_train_set_i.tocsr()[:,indices].todense()
x_test_set_new = x_test_set_i.tocsr()[:,indices].todense()
x_cv_set_new = x_cv_set_i.tocsr()[:,indices].todense()
```

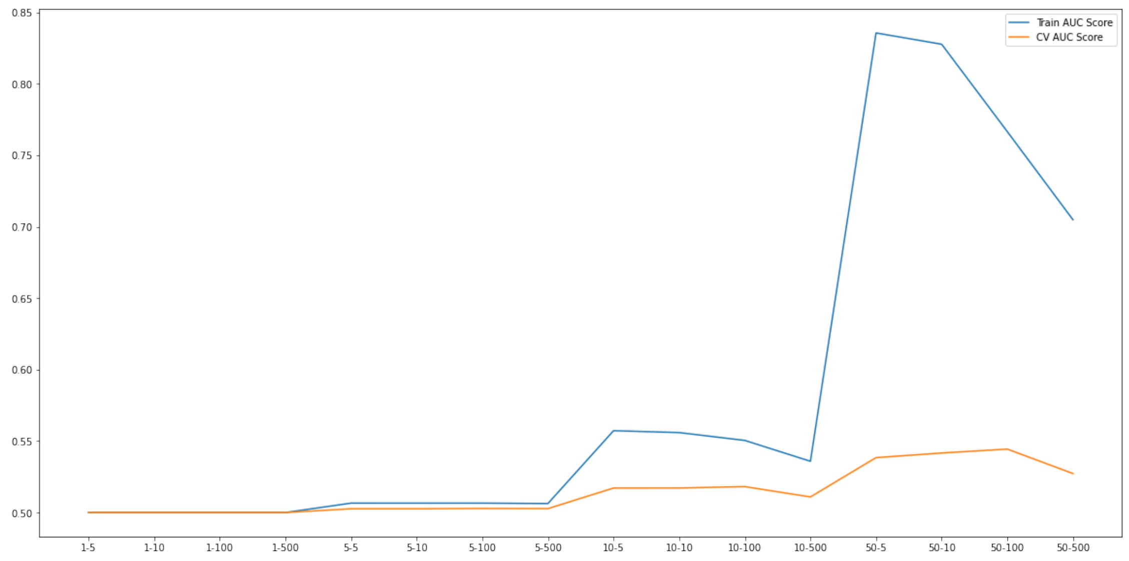
```
(2820,)
```

1.2.2 Hyperparameter Tuning

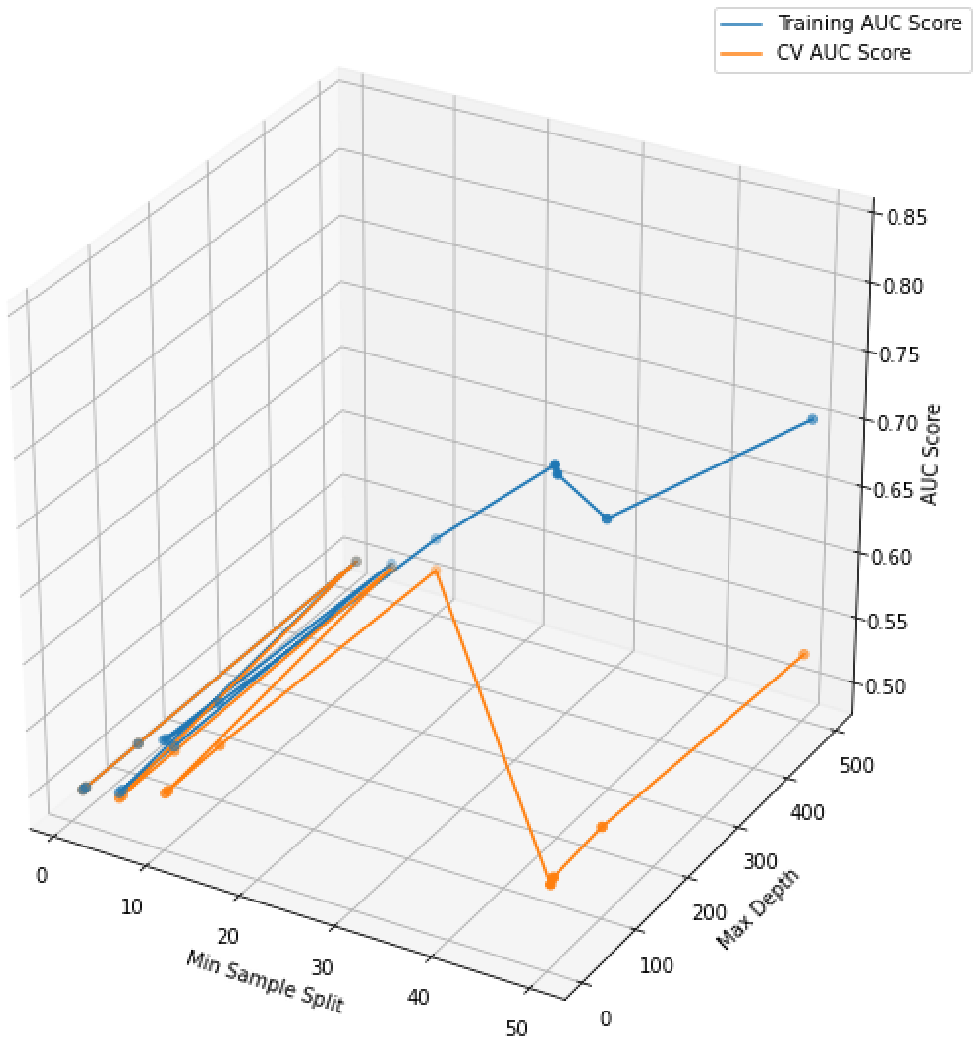
[42]:
```
# Hyperparameter Tuning
train_auc_score = {}
cv_auc_score = {}
max_depth=[1, 5, 10, 50]
min_sample_split= [5, 10, 100, 500]

for depth in max_depth:
    for sample in min_sample_split:
        order = '{}-{}'.format(depth, sample)
        clf = DecisionTreeClassifier(max_depth=depth, min_samples_split=sample)
        clf.fit(x_train_set_new, y_train)

        # Predict Train
        y_pred = clf.predict(x_train_set_new)
        auc_score = roc_auc_score(y_train, y_pred)
        print('For',order,'Training AUC Score',auc_score)
        train_auc_score[order] = auc_score
        # Predict CV
        y_pred = clf.predict(x_cv_set_new)
        auc_score = roc_auc_score(y_cv, y_pred)
        cv_auc_score[order] = auc_score
        print('For',order,'CV AUC Score',auc_score)
```

```
For 1-5 Training AUC Score 0.5
For 1-5 CV AUC Score 0.5
For 1-10 Training AUC Score 0.5
For 1-10 CV AUC Score 0.5
For 1-100 Training AUC Score 0.5
For 1-100 CV AUC Score 0.5
For 1-500 Training AUC Score 0.5
For 1-500 CV AUC Score 0.5
For 5-5 Training AUC Score 0.5065871235791305
For 5-5 CV AUC Score 0.5026387598833313
For 5-10 Training AUC Score 0.506570268983725
For 5-10 CV AUC Score 0.5026387598833313
For 5-100 Training AUC Score 0.506570268983725
For 5-100 CV AUC Score 0.5028613615925099
For 5-500 Training AUC Score 0.506295329888682
For 5-500 CV AUC Score 0.5027398857513282
For 10-5 Training AUC Score 0.5572540409107406
For 10-5 CV AUC Score 0.5171484435798013
For 10-10 Training AUC Score 0.555921656248984
For 10-10 CV AUC Score 0.5171687935529858
For 10-100 Training AUC Score 0.5504546420844041
For 10-100 CV AUC Score 0.5181406002824392
For 10-500 Training AUC Score 0.5358819984223926
For 10-500 CV AUC Score 0.5109067324053151
For 50-5 Training AUC Score 0.8355579417217762
For 50-5 CV AUC Score 0.5384293070584532
For 50-10 Training AUC Score 0.827667550522796
For 50-10 CV AUC Score 0.5416608395023594
For 50-100 Training AUC Score 0.7665316512800544
For 50-100 CV AUC Score 0.5443994772578352
For 50-500 Training AUC Score 0.7049035180918417
For 50-500 CV AUC Score 0.5272974769522545
```

[43]:
```
_,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()
```

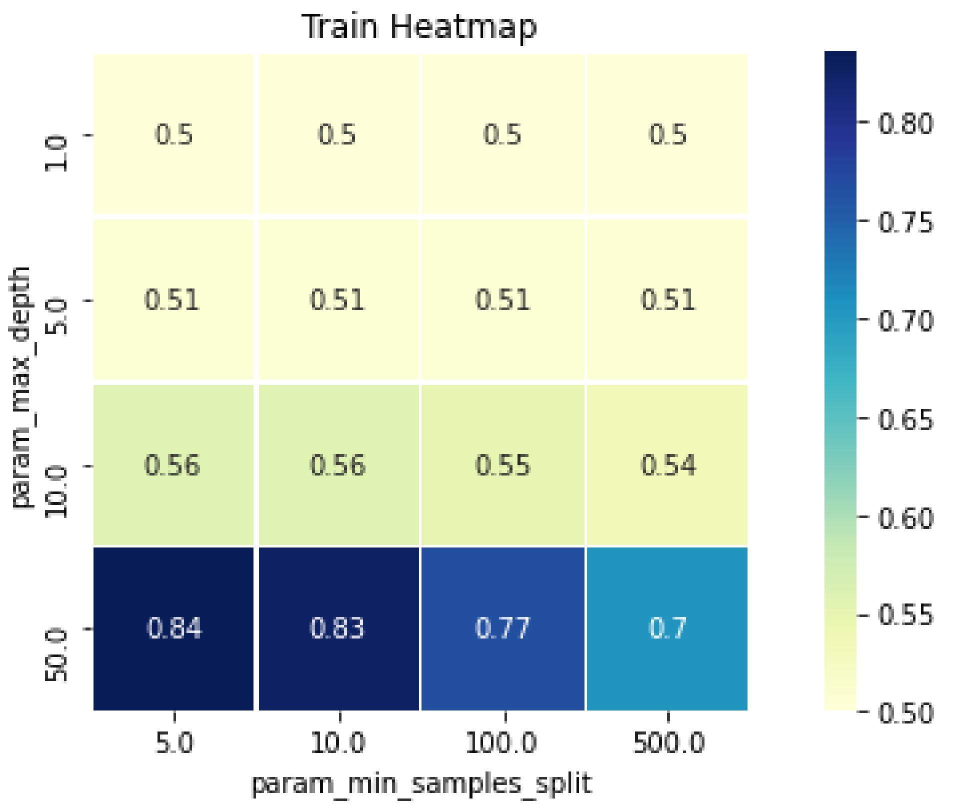


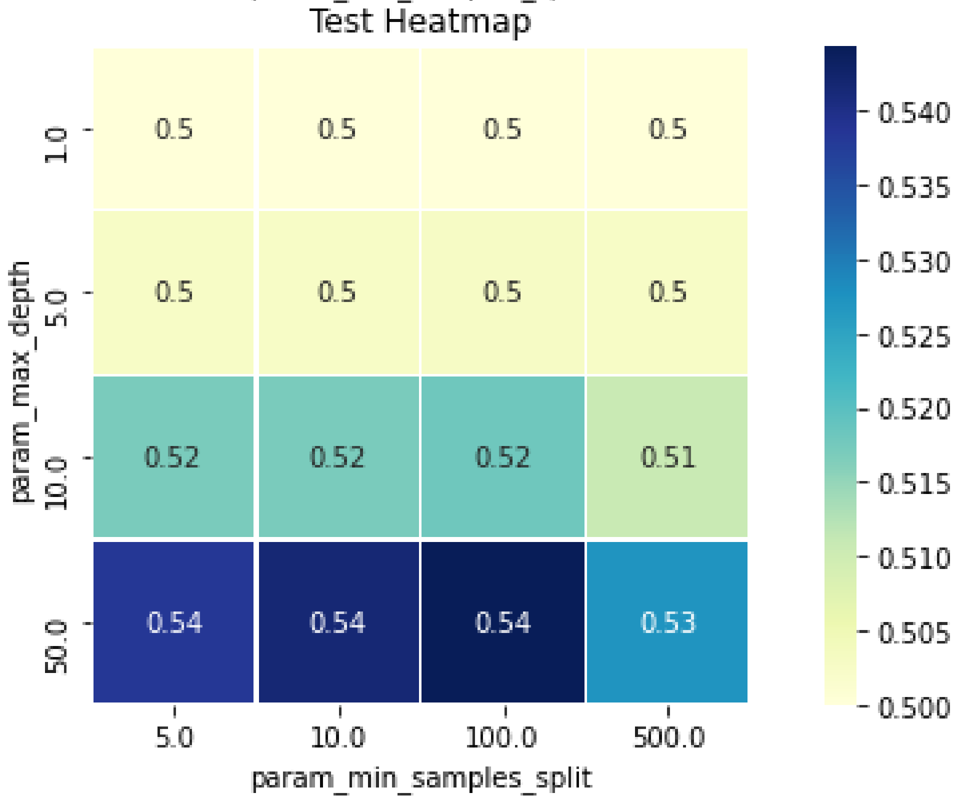
[44]:
```
x_axis = []
y_axis = []
for key in train_auc_score.keys():
    temp = list(map(int, key.split('-')))
    x_axis.append(temp[0])
    y_axis.append(temp[1])
# 3D Plot
subplot_args = {'projection':'3d'}
fig, ax = plt.subplots(1, 1, figsize=(10,10), subplot_kw=subplot_args)
ax.scatter3D(x_axis,y_axis,  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()
```



```
[45]: 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(2, 1, figsize=(10, 10))
      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()
```

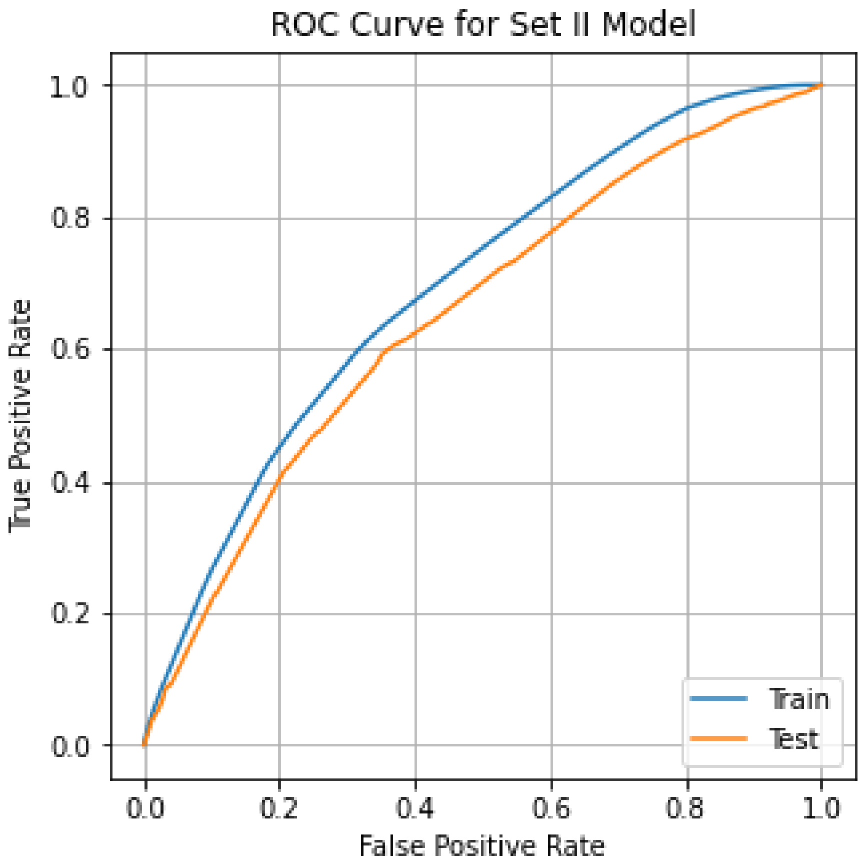

Train Heatmap
param_max_depth
1.0
5.0
10.0
50.0
param_min_samples_split
5.0
10.0
100.0
500.0
0.5
0.5
0.5
0.5
0.51
0.51
0.51
0.51
0.56
0.56
0.55
0.54
0.84
0.83
0.77
0.7
0.80
0.75
0.70
0.65
0.60
0.55
0.50

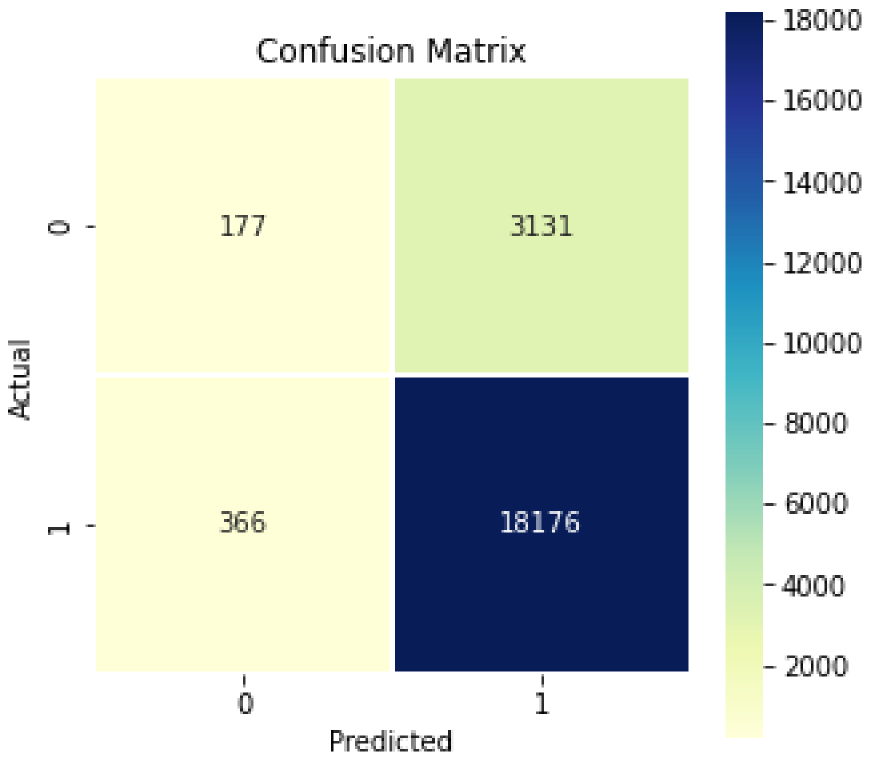

Test Heatmap
param_max_depth
1.0
5.0
10.0
50.0
param_min_samples_split
5.0
10.0
100.0
500.0
0.5
0.5
0.5
0.5
0.5
0.5
0.5
0.5
0.52
0.52
0.52
0.51
0.54
0.54
0.54
0.53
0.540
0.535
0.530
0.525
0.520
0.515
0.510
0.505
0.500

1.2.3 Best Model Training

[46]:
```
best_max_depth = 10
best_min_sample_split = 100

model_new = DecisionTreeClassifier(max_depth=best_max_depth,␣
 ↪min_samples_split=best_min_sample_split)
model_new.fit(x_train_set_new, y_train)

# Predicting Train
y_pred = model_new.predict(x_train_set_new)
train_auc_score = roc_auc_score(y_train, y_pred)
print('Train AUC Score',train_auc_score)

# Predicting Test
y_pred = model_new.predict(x_test_set_new)
test_auc_score = roc_auc_score(y_test, y_pred)
print('Test AUC Score', test_auc_score)
table.append_row(['TFIDF Imp Features','Decision Tree','Max Depth = {} Min␣
 ↪Sample Split = {}'.format(best_max_depth, best_min_sample_split),␣
 ↪test_auc_score])
# Plot ROC Curve
_, ax = plt.subplots(1,1,figsize=(5,5))
plot_roc_curve(model_new, x_train_set_new, y_train, ax=ax, label='Train')
plot_roc_curve(model_new, x_test_set_new, 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 0.5503349047270473
Test AUC Score 0.5168838397796721
```


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



1.3 Model Summary

[54]:
```
table.column_headers =["Featurization Type", "Model", "Hyperparameters", "AUC␣
 ↪Score"]
print(table)
```

```
+-----------------+---------------+----------------------------------+------------+
| Featurization   |     Model     |         Hyperparameters          | AUC Score  |
|      Type       |               |                                  |            |
+-----------------+---------------+----------------------------------+------------+
|      TFIDF      | Decision Tree | Max Depth = 10 Min Sample Split  |   0.51     |
|                 |               |             = 500                |            |
+-----------------+---------------+----------------------------------+------------+
|    TFIDF W2V    | Decision Tree | Max Depth = 10 Min Sample Split  |   0.521    |
|                 |               |             = 100                |            |
+-----------------+---------------+----------------------------------+------------+
| TFIDF Imp Feat  | Decision Tree | Max Depth = 10 Min Sample Split  |   0.517    |
|      ures       |               |             = 100                |            |
+-----------------+---------------+----------------------------------+------------+
```