Assignment 8: DT

1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**

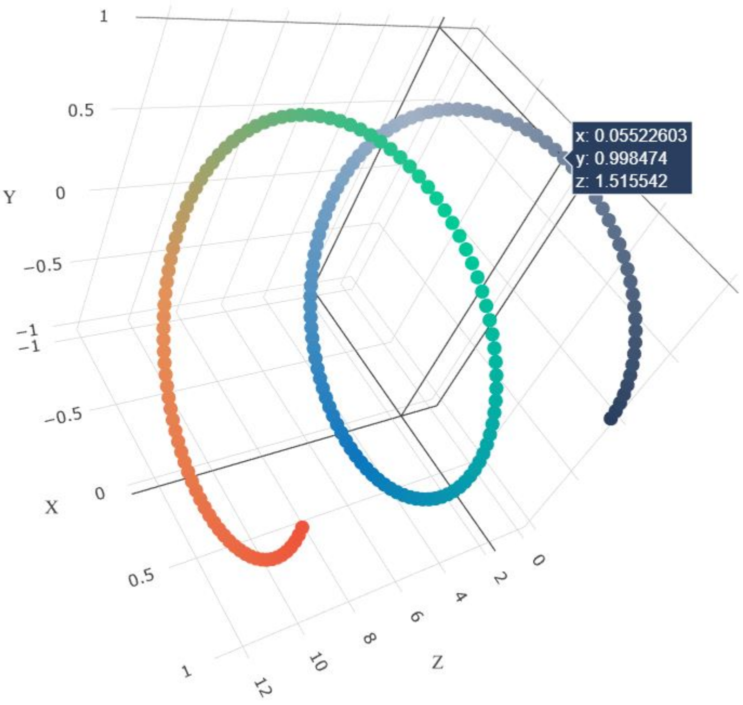
    - Set 1: categorical, numerical features + preprocessed_eassay (TFIDF)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF W2V)

2. **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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) 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)

3. **Representation of results**

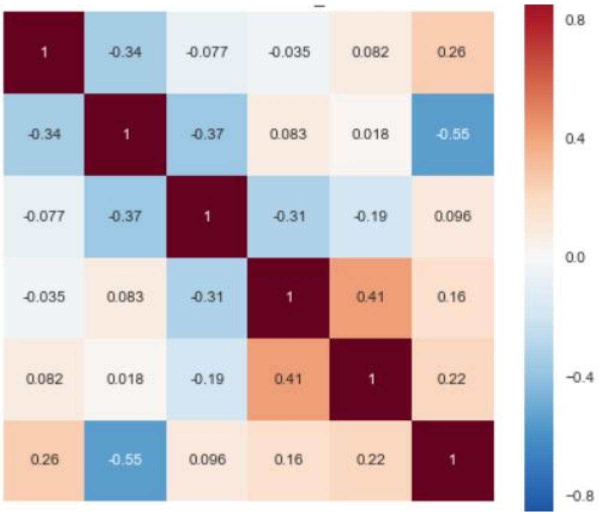
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



with X-axis as **min_sample_split**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

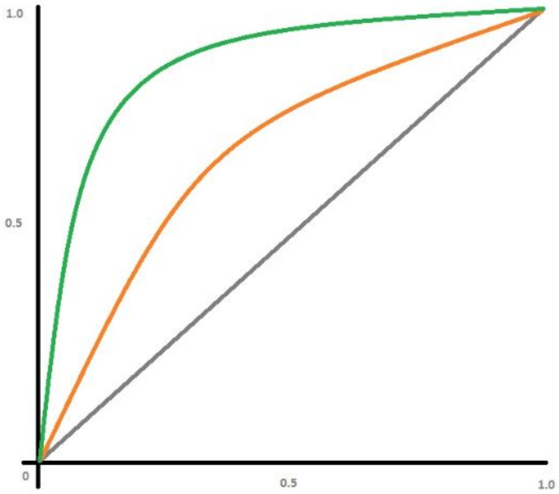
or

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure

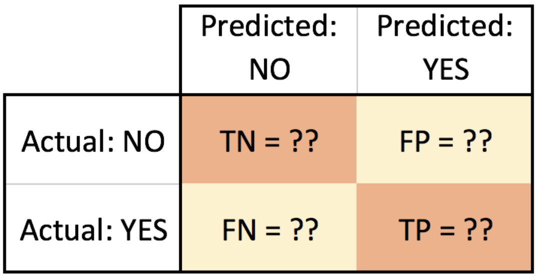


[seaborn heat maps (https://seaborn.pydata.org/generated/seaborn.heatmap.html)](https://seaborn.pydata.org/generated/seaborn.heatmap.html) with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**

- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.



- Along with plotting ROC curve, you need to print the [confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

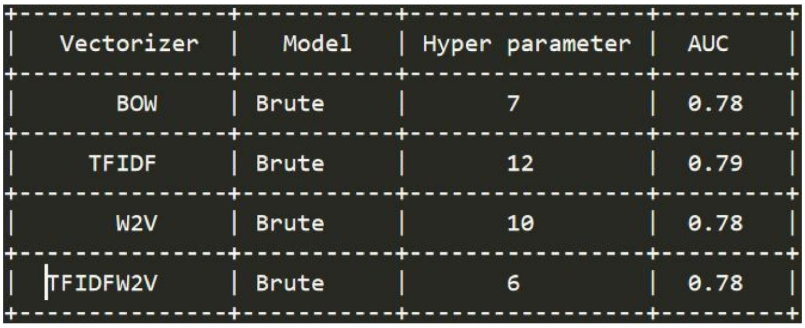


	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

- Once after you plot the confusion matrix with the test data, get all the `false positive data points`
  - Plot the WordCloud([https://www.geeksforgeeks.org/generating-word-cloud-python/ (https://www.geeksforgeeks.org/generating-word-cloud-python/)](https://www.geeksforgeeks.org/generating-word-cloud-python/)) with the words of essay text of these `false positive data points`
  - Plot the box plot with the `price` of these `false positive data points`
  - Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points`

4. **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) (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.

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



```
+--------------+----------+-----------------+---------+
|  Vectorizer  |  Model   | Hyper parameter |  AUC    |
+--------------+----------+-----------------+---------+
|     BOW      | Brute    |        7        |  0.78   |
+--------------+----------+-----------------+---------+
|    TFIDF     | Brute    |       12        |  0.79   |
+--------------+----------+-----------------+---------+
|     W2V      | Brute    |       10        |  0.78   |
+--------------+----------+-----------------+---------+
|   TFIDFW2V   | Brute    |        6        |  0.78   |
+--------------+----------+-----------------+---------+
```

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from chart_studio import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

```
C:\Users\MONIKA KUMARI\Anaconda3\lib\site-packages\gensim\utils.py:1197: Use
rWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

1. Decision Tree

1.1 Loading Data

In [2]:

```
import pandas
data = pandas.read_csv('preprocessed_data.csv')
data.head()
```

Out[2]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_pr
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	
2	ca	mrs	grades_prek_2	
3	ga	mrs	grades_prek_2	
4	wa	mrs	grades_3_5	

In [3]:

```
data.columns
```

Out[3]:

```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       'clean_categories', 'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

In [4]:

```
print(data['school_state'].values)
```

```
['ca' 'ut' 'ca' ... 'il' 'hi' 'ca']
```

1.2 Splitting data into Train and cross validation(or test): Stratified Sampling

In [5]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# 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 coc
# 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
```

In [6]:

```
y = data["project_is_approved"].values
#print(y)
X = data.drop(["project_is_approved"], axis= 1)
X.head(2)
```

Out[6]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_pr
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	

In [7]:

```
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Flase)#
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify= y) # th
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify


print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(73196, 8) (73196,)
(36052, 8) (36052,)
```

1.3 Make Data Model Ready: encoding essay

In [8]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# 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 coa
# make sure you featurize train and test data separatly

# 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
```

1.3.1 Vectorizing Text data with TFIDF

In [9]:

```
print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)
from sklearn.feature_extraction.text import TfidfVectorizer
essay_tfidf_vectorizer = TfidfVectorizer(min_df= 10, max_features= 5000)
essay_tfidf_vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = essay_tfidf_vectorizer.transform(X_train['essay'].values)
#X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = essay_tfidf_vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
#print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
(73196, 8) (73196,)
(36052, 8) (36052,)
============================================================================
========================
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
============================================================================
========================
```

1.3.2 TFIDF weighted W2V on essay

In [10]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickl
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [11]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'].values)
# 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())
```

In [12]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_essay_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].values, position= 0, leave= True): # for each review/
    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 words
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_essay_train.append(vector)

print(len(tfidf_w2v_essay_train))
print(len(tfidf_w2v_essay_train[0]))
```

```
100%|███████████████████████████████████| 73196/73196 [05:27<00:00, 223.50i
t/s]

73196
300
```

In [13]:

```
tfidf_w2v_essay_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values, position= 0, leave= True): # for each review/s
    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 words
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_essay_test.append(vector)

print(len(tfidf_w2v_essay_test))
print(len(tfidf_w2v_essay_test[0]))
```

```
100%|██████████████████████████████| 36052/36052 [02:38<00:00, 227.85i
t/s]

36052
300
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [14]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# 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 coc
# make sure you featurize train and test data separtly

# 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
```

1.4.1 Encoding categorical feature: School_state

In [15]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
#X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
#print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(73196, 51) (73196,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'i
a', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo',
'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'o
r', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv',
'wy']
============================================================================
========================
```

1.4.2. encoding categorical features: clean_categories

In [16]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_categories_ohe = vectorizer.transform(X_train['clean_categories'].values)
#X_cv_categories_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_categories_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_categories_ohe.shape, y_train.shape)
#print(X_cv_categories_ohe.shape, y_cv.shape)
print(X_test_categories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(73196, 9) (73196,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'liter
acy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
============================================================================
========================
```

1.4.3. encoding categorical features: clean_subcategories

In [17]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train dat

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
#X_cv_subcategories_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subcategories_ohe.shape, y_train.shape)
#print(X_cv_subcategories_ohe.shape, y_cv.shape)
print(X_test_subcategories_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(73196, 30) (73196,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_governmen
t', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economic
s', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy',
'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness',
'history_geography', 'literacy', 'literature_writing', 'mathematics', 'musi
c', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 's
ocialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
============================================================================
========================
```

1.4.4. encoding categorical features: teacher_prefix

In [18]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
#X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
#print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(73196, 5) (73196,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
============================================================================
========================
```

1.4.5. encoding categorical features: project_grade_category

In [19]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
#X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
#print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(73196, 4) (73196,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
============================================================================
========================
```

1.4.6. encoding numerical features: teacher_number_of_previously_posted_projects

In [20]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)

X_train_projects_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_p
#X_cv_projects_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projec
X_test_projects_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_pro

print("After vectorizations")
print(X_train_projects_norm.shape, y_train.shape)
#print(X_cv_projects_norm.shape, y_cv.shape)
print(X_test_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
============================================================================
========================
```

In [21]:

```
X_train_projects_norm = X_train_projects_norm.reshape(-1,1)
X_test_projects_norm = X_test_projects_norm.reshape(-1,1)
print(X_train_projects_norm.shape, y_train.shape)
print(X_test_projects_norm.shape, y_test.shape)
```

```
(73196, 1) (73196,)
(36052, 1) (36052,)
```

In [22]:

```
#print(X_train_projects_norm)
```

1.4.7. encoding numerical features: Price

In [23]:

```
normalizer.fit(X_train['price'].values.reshape(1,-1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
#print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
============================================================================
========================
```

In [24]:

```
X_train_price_norm = X_train_price_norm.reshape(-1,1)
X_test_price_norm = X_test_price_norm.reshape(-1,1)
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
```

```
(73196, 1) (73196,)
(36052, 1) (36052,)
```

In [25]:

```
#print(X_test_price_norm)
```

1.5 Appling Decision Tree on different kind of featurization as mentioned in the instructions

Apply Decision Tree on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

In [26]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# 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 coc
# 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
```

1.5.1.Concatenating all the Features of Set 1: categorical, numerical features + essay (TFIDF)

In [27]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr1 = hstack((X_train_essay_tfidf, X_train_categories_ohe, X_train_subcategories_ohe, X_t
#X_cr = hstack((X_cv_essay_bow, X_cv_title_bow, X_cv_resources_bow, X_cv_categories_ohe, X_
X_te1 = hstack((X_test_essay_tfidf, X_test_categories_ohe, X_test_subcategories_ohe, X_test

print("Final Data matrix")
print(X_tr1.shape, y_train.shape)
#print(X_cr.shape, y_cv.shape)
print(X_te1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(73196, 5101) (73196,)
(36052, 5101) (36052,)
============================================================================
========================
```

1.5.1.1 Hyperparameter tuning for Set 1

In [28]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.ht
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(criterion='gini', class_weight= 'balanced')
parameters = {"max_depth" :[1,5,10,50], "min_samples_split" : [5,10,100,500]}

clf1 = GridSearchCV(dt, parameters, cv=3, scoring='roc_auc')
clf1.fit(X_tr1, y_train)
```

Out[28]:

```
GridSearchCV(cv=3, error_score='raise',
       estimator=DecisionTreeClassifier(class_weight='balanced', criterion
='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10,
100, 500]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [29]:

```
results = pd.DataFrame.from_dict(clf1.cv_results_)
results = results.sort_values(['param_max_depth'])
#results = results.sort_values(['param_min_samples_split'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
max_depth =  results['param_max_depth']
min_samples_split = results['param_min_samples_split']
```

In [30]:

```
results.head()
```

Out[30]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min
0	1.695089	0.015674	0.151036	0.007355	1	
1	1.682167	0.007363	0.145842	0.007365	1	
2	1.671751	0.000008	0.161454	0.007375	1	
3	1.687373	0.012749	0.145822	0.007366	1	
4	6.790998	0.027070	0.151024	0.007362	5	

In [31]:

```
print('Best score: ',clf1.best_score_)
print('Parameters with best score: ',clf1.best_params_)
print('='*75)
print('Train AUC scores')
print(results['mean_train_score'])
print('CV AUC scores')
print(results['mean_test_score'])
```

```
Best score:  0.6449901419934234
Parameters with best score:  {'max_depth': 10, 'min_samples_split': 500}
===========================================================================
Train AUC scores
0     0.550555
1     0.550555
2     0.550555
3     0.550555
4     0.647011
5     0.646984
6     0.646440
7     0.645149
8     0.746228
9     0.744912
10    0.729106
11    0.708368
12    0.981852
13    0.974886
14    0.910503
15    0.840727
Name: mean_train_score, dtype: float64
CV AUC scores
0     0.547693
1     0.547693
2     0.547693
3     0.547693
4     0.628371
5     0.628192
6     0.627919
7     0.628318
8     0.636292
9     0.636754
10    0.637983
11    0.644990
12    0.568678
13    0.571435
14    0.595913
15    0.615165
Name: mean_test_score, dtype: float64
```

1.5.1.2 Representation of results

In [32]:

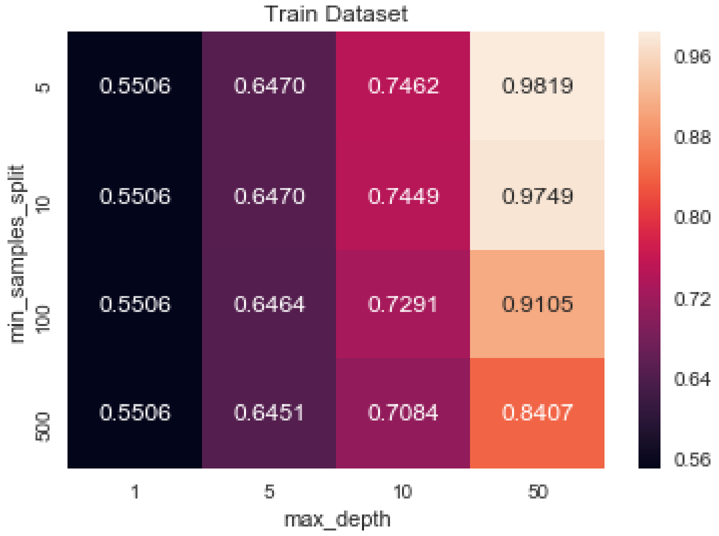
```
'''# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x= min_samples_split, y= max_depth, z=train_auc, name = 'train')
trace2 = go.Scatter3d(x= min_samples_split, y= max_depth, z=cv_auc, name = 'test')
data = [trace1, trace2]

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

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

Out[32]:

```
"# https://plot.ly/python/3d-axes/\ntrace1 (https://plot.ly/python/3d-axes/
\ntrace1) = go.Scatter3d(x= min_samples_split, y= max_depth, z=train_auc, na
me = 'train')\ntrace2 = go.Scatter3d(x= min_samples_split, y= max_depth, z=c
v_auc, name = 'test')\ndata = [trace1, trace2]\n\nlayout = go.Layout(scene =
dict(\n        xaxis = dict(title='min_samples_split'),\n        yaxis = dic
t(title='max_depth'),\n        zaxis = dict(title='AUC'),))\n\nfig = go.Figu
re(data=data, layout=layout)\noffline.iplot(fig, filename='3d-scatter-colors
cale')"
```

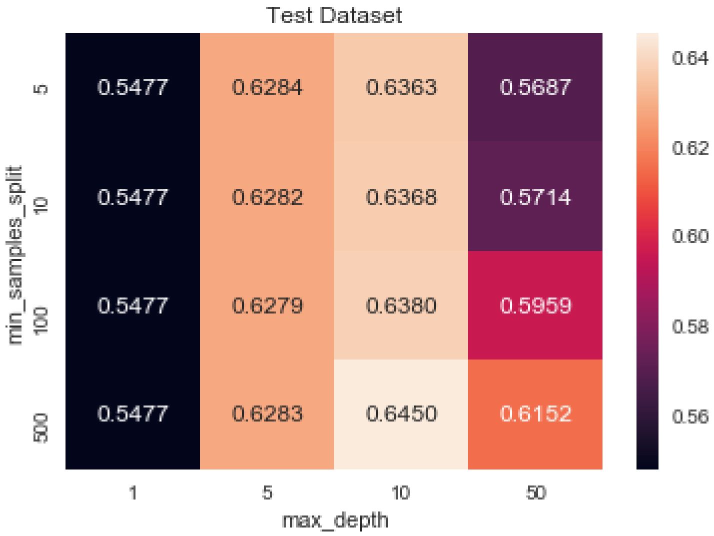
In [33]:

```
import numpy as np; np.random.seed(0)
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
uniform_data = pd.DataFrame({'min_samples_split': min_samples_split, 'max_depth': max_depth
uniform_data = uniform_data.pivot("min_samples_split","max_depth","Z")
ax= sns.heatmap(uniform_data, annot= True, fmt= ".4f")
plt.title("Train Dataset")
plt.show()
```



In [34]:

```
uniform_data = pd.DataFrame({'min_samples_split': min_samples_split, 'max_depth': max_depth
uniform_data = uniform_data.pivot("min_samples_split","max_depth","Z")
ax= sns.heatmap(uniform_data, annot= True, fmt= ".4f")
plt.title("Test Dataset")
plt.show()
```



1.5.1.3 Testing the performance of the model on test data, plotting ROC Curves

In [35]:

```
def prob_predict(clf, data):
    y_data_pred = []
    y_data_pred.extend(clf.predict_proba(data)[:,1])
    return y_data_pred
```

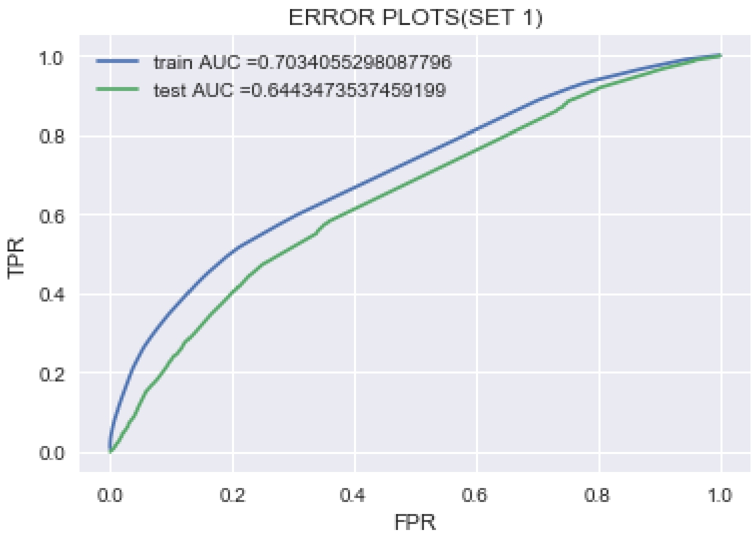
In [36]:

```
best_max_depth= clf1.best_params_['max_depth']
best_min_samples_split= clf1.best_params_['min_samples_split']
print("best_max_depth= ",best_max_depth)
print("best_min_samples_split= ",best_min_samples_split)
```

```
best_max_depth=  10
best_min_samples_split=  500
```

In [37]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.
from sklearn.metrics import roc_curve, auc


dt_set1 = DecisionTreeClassifier(max_depth= best_max_depth, min_samples_split= best_min_sam
dt_set1.fit(X_tr1, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = prob_predict(dt_set1, X_tr1)
y_test_pred = prob_predict(dt_set1, X_te1)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS(SET 1)")
plt.grid(True)
plt.show()
```



1.5.1.4 Confusion Matrix

In [38]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [39]:

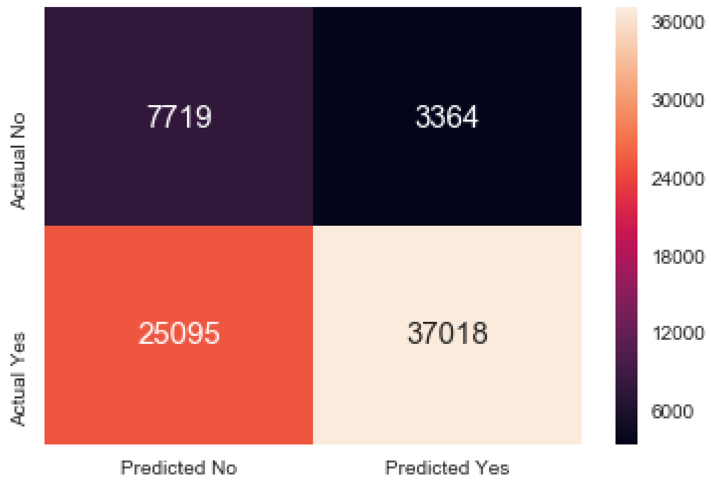
```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)

def get_confusion_matrix(y_train, y_train_pred):
    cm = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
    sns.heatmap(cm, annot = True, fmt= 'd',annot_kws={"size": 15}, xticklabels= ['Predictec
```

```
the maximum value of tpr*(1-fpr) 0.41508224120905435 for threshold 0.453
```

In [40]:

```
print("Train Confusion Matrix")
get_confusion_matrix(y_train, y_train_pred)
```

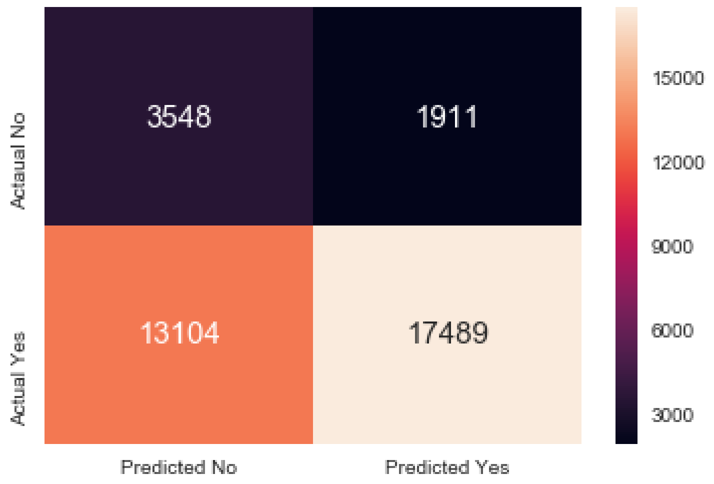
```
Train Confusion Matrix
```



In [41]:

```
print("Test Confusion Matrix")
get_confusion_matrix(y_test, y_test_pred)
```

```
Test Confusion Matrix
```



In [42]:

```
#get all the false positive data points with test dataset
actual_output = y_test
pred_output = predict_with_best_t(y_test_pred, best_t)
#print(len(actual_output))
#print(len(pred_output))
print(actual_output[280], pred_output[280])
```

```
1 1
```

In [43]:

```
false_positive_data = []
for i in range(len(y_test)):
    if (actual_output[i] == 0) &  (pred_output[i] == 1):
        #print(i)
        false_positive_data.append(i)
print(false_positive_data[0:20])
print(len(false_positive_data))
```

```
[4, 57, 72, 142, 144, 153, 162, 198, 202, 212, 224, 241, 260, 283, 285, 304,
339, 368, 388, 417]
1911
```

In [44]:

```
false_positive_essay1= []
for i in false_positive_data:
    false_positive_essay1.append(X_test['essay'].values[i])
print(len(false_positive_essay1))
#print(false_positive_essay1[0:20])
```

```
1911
```

In [45]:

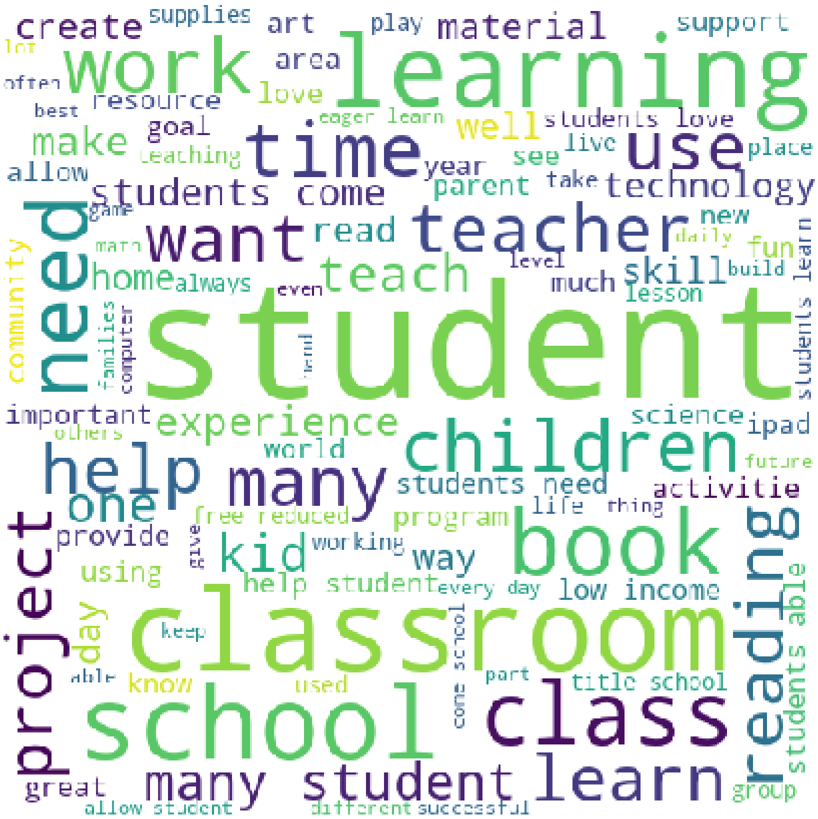
```
pip install wordcloud
```

```
The following command must be run outside of the IPython shell:

    $ pip install wordcloud

The Python package manager (pip) can only be used from outside of IPython.
Please reissue the `pip` command in a separate terminal or command prompt.

See the Python documentation for more information on how to install package
s:

    https://docs.python.org/3/installing/ (https://docs.python.org/3/install
ing/)
```

1.5.1.5 Plot the WordCloud with the words of essay text of these false positive data points

In [46]:

```
import nltk
import string
from nltk.corpus import stopwords

# Python program to generate WordCloud

# importing all necessery modules
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = ["nannan"] + list(STOPWORDS)
for val in false_positive_essay1:
    # typecaste each val to string
    val = str(val)
   # split the value
    tokens = val.split()
    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '
wordcloud = WordCloud(width = 500, height = 500,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)
# plot the WordCloud image
plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



1.5.1.6 Plot the box plot with the price of the false positive data points of set 1

In [47]:

```
false_positive_price1= []
for i in false_positive_data:
    false_positive_price1.append(X_test['price'].values[i])
print(len(false_positive_price1))
#print(false_positive_price1[3])
```

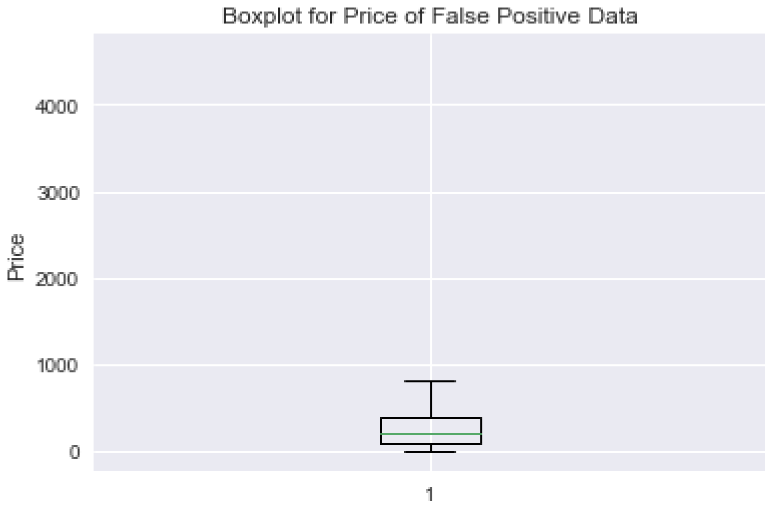
```
1911
```

In [48]:

```
#print(false_positive_price1)
```

In [49]:

```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
plt.boxplot([false_positive_price1])
plt.ylabel('Price')
plt.title("Boxplot for Price of False Positive Data")
plt.grid(True)
plt.show()
```



1.5.1.7 Plot pdf with teacher_number_of_previously_posted_projects of false positive data points of set_1

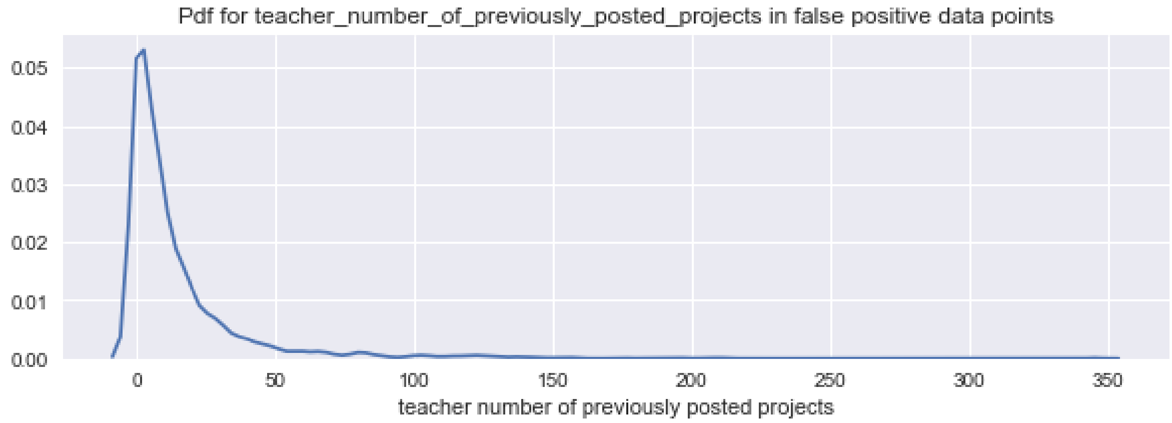
In [50]:

```
false_positive_prev_projects1= []
for i in false_positive_data:
    false_positive_prev_projects1.append(X_test['teacher_number_of_previously_posted_projec
print(len(false_positive_prev_projects1))
#print(false_positive_prev_projects1[3])
```

```
1911
```

In [51]:

```
plt.figure(figsize=(10,3))
sns.distplot(false_positive_prev_projects1, hist=False)
#sns.distplot(rejected_word_count, hist=False, label="Not Approved Projects")
#plt.legend()
plt.title("Pdf for teacher_number_of_previously_posted_projects in false positive data poin
plt.xlabel("teacher number of previously posted projects")
plt.show()
```



1.5.2.Concatenating all the Features of Set 2: categorical, numerical features + essay (TFIDF W2V)

In [52]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr2 = hstack((tfidf_w2v_essay_train, X_train_categories_ohe, X_train_subcategories_ohe, X
#X_cr = hstack((X_cv_essay_bow, X_cv_title_bow, X_cv_resources_bow, X_cv_categories_ohe, X_
X_te2 = hstack((tfidf_w2v_essay_test, X_test_categories_ohe, X_test_subcategories_ohe, X_te

print("Final Data matrix")
print(X_tr2.shape, y_train.shape)
#print(X_cr.shape, y_cv.shape)
print(X_te2.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(73196, 401) (73196,)
(36052, 401) (36052,)
============================================================================
========================
```

1.5.2.1 Hyperparameter tuning for Set 2

In [53]:

```
dt = DecisionTreeClassifier(criterion='gini', class_weight= 'balanced')
parameters = {"max_depth" :[1,5,10,50], "min_samples_split" : [5,10,100,500]}

clf2 = GridSearchCV(dt, parameters, cv=3, scoring='roc_auc')
clf2 = clf2.fit(X_tr2, y_train)
```

In [54]:

```
results = pd.DataFrame.from_dict(clf2.cv_results_)
results = results.sort_values(['param_max_depth'])
#results = results.sort_values(['param_min_samples_split'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
max_depth =  results['param_max_depth']
min_samples_split = results['param_min_samples_split']
results.head()
```

Out[54]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min
0	5.127194	0.052321	0.484341	1.275756e-02	1	
1	5.077764	0.025514	0.473928	7.364292e-03	1	
2	5.091953	0.047954	0.468717	9.199649e-07	1	
3	5.088180	0.051556	0.468718	1.275610e-02	1	
4	21.540147	0.229741	0.468717	1.072147e-06	5	

In [55]:

```
print('Best score: ',clf2.best_score_)
print('Parameters with best score: ',clf2.best_params_)
print('='*75)
print('Train AUC scores')
print(results['mean_train_score'])
print('CV AUC scores')
print(results['mean_test_score'])
```

```
Best score:  0.6263674657373611
Parameters with best score:  {'max_depth': 5, 'min_samples_split': 500}
===========================================================================
Train AUC scores
0     0.552369
1     0.552369
2     0.552369
3     0.552369
4     0.655990
5     0.655957
6     0.655913
7     0.655029
8     0.821639
9     0.820384
10    0.788803
11    0.733719
12    0.999920
13    0.999132
14    0.901688
15    0.751752
Name: mean_train_score, dtype: float64
CV AUC scores
0     0.544688
1     0.544688
2     0.544688
3     0.544688
4     0.625840
5     0.625776
6     0.625775
7     0.626367
8     0.603605
9     0.604125
10    0.610622
11    0.622850
12    0.531973
13    0.532737
14    0.571627
15    0.614325
Name: mean_test_score, dtype: float64
```

1.5.2.2 Representation of results

In [56]:

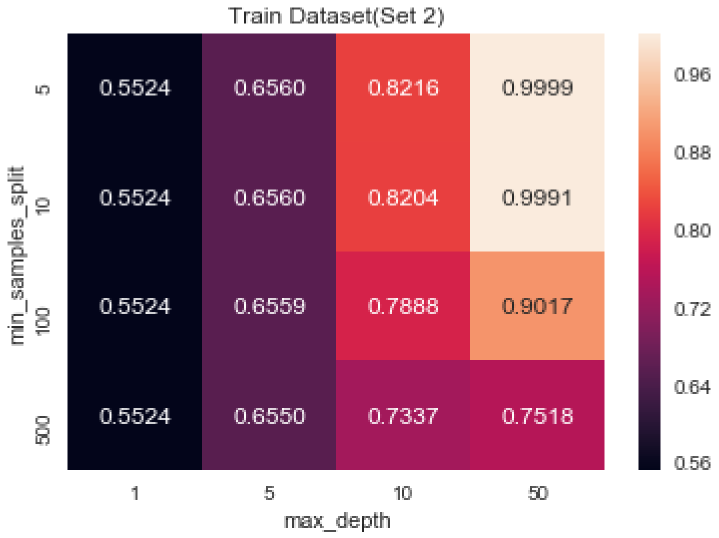
```
'''# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x= min_samples_split, y= max_depth, z=train_auc, name = 'train')
trace2 = go.Scatter3d(x= min_samples_split, y= max_depth, z=cv_auc, name = 'test')
data = [trace1, trace2]

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

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

Out[56]:

```
"# https://plot.ly/python/3d-axes/\ntrace1 (https://plot.ly/python/3d-axes/
\ntrace1) = go.Scatter3d(x= min_samples_split, y= max_depth, z=train_auc, na
me = 'train')\ntrace2 = go.Scatter3d(x= min_samples_split, y= max_depth, z=c
v_auc, name = 'test')\ndata = [trace1, trace2]\n\nlayout = go.Layout(scene =
dict(\n        xaxis = dict(title='min_samples_split'),\n        yaxis = dic
t(title='max_depth'),\n        zaxis = dict(title='AUC'),))\n\nfig = go.Figu
re(data=data, layout=layout)\noffline.iplot(fig, filename='3d-scatter-colors
cale')"
```

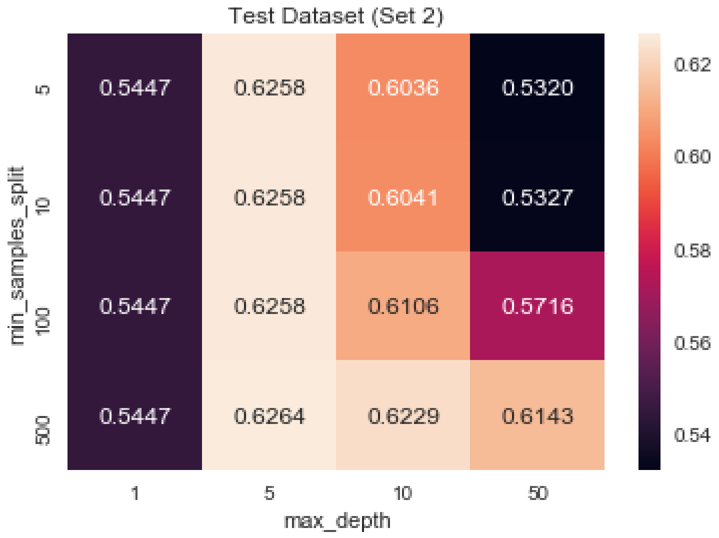
In [57]:

```
uniform_data = pd.DataFrame({'min_samples_split': min_samples_split, 'max_depth': max_depth
uniform_data = uniform_data.pivot("min_samples_split","max_depth","Z")
ax= sns.heatmap(uniform_data, annot= True, fmt= ".4f")
plt.title("Train Dataset(Set 2)")
plt.show()
```



In [58]:

```
uniform_data = pd.DataFrame({'min_samples_split': min_samples_split, 'max_depth': max_depth
uniform_data = uniform_data.pivot("min_samples_split","max_depth","Z")
ax= sns.heatmap(uniform_data, annot= True, fmt= ".4f")
plt.title("Test Dataset (Set 2)")
plt.show()
```



1.5.2.3 Testing the performance of the model on test data, plotting ROC Curves

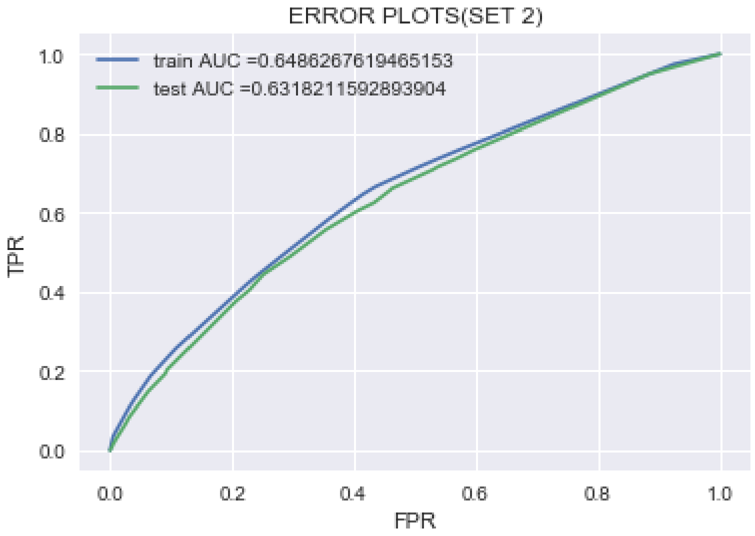
In [59]:

```
best_max_depth= clf2.best_params_['max_depth']
best_min_samples_split= clf2.best_params_['min_samples_split']
print("best_max_depth= ",best_max_depth)
print("best_min_samples_split= ",best_min_samples_split)
```

```
best_max_depth=  5
best_min_samples_split=  500
```

In [60]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.
from sklearn.metrics import roc_curve, auc


dt_set2 = DecisionTreeClassifier(max_depth= best_max_depth, min_samples_split= best_min_sam
dt_set2.fit(X_tr2, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = prob_predict(dt_set2, X_tr2)
y_test_pred = prob_predict(dt_set2, X_te2)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS(SET 2)")
plt.grid(True)
plt.show()
```



1.5.2.4 Confusion Matrix

In [61]:

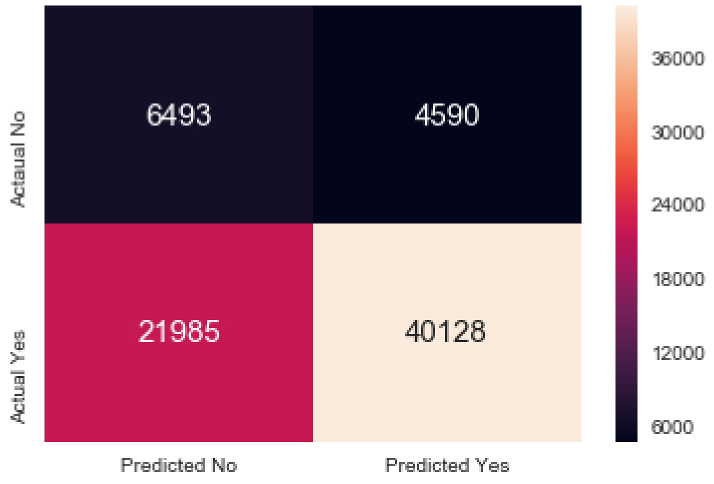
```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
```

```
the maximum value of tpr*(1-fpr) 0.3784888401080909 for threshold 0.524
```

In [62]:

```
print("Train Confusion Matrix")
get_confusion_matrix(y_train, y_train_pred)
```

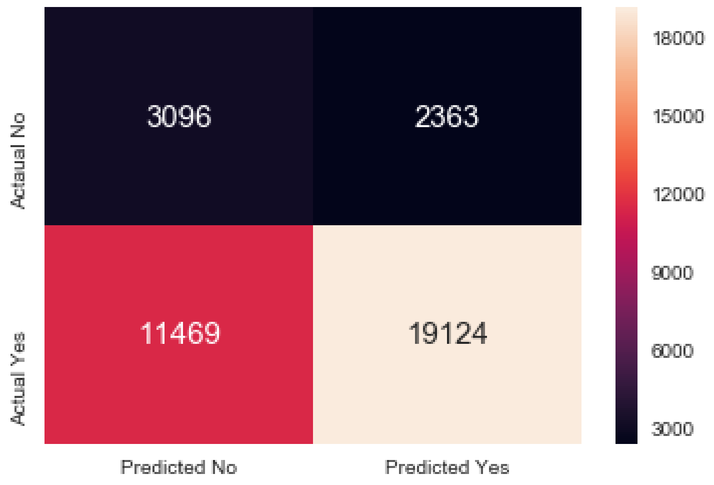
```
Train Confusion Matrix
```



In [63]:

```
print("Test Confusion Matrix")
get_confusion_matrix(y_test, y_test_pred)
```

```
Test Confusion Matrix
```



In [64]:

```
#get all the false positive data points with test dataset
actual_output = y_test
pred_output = predict_with_best_t(y_test_pred, best_t)
print(len(actual_output))
print(len(pred_output))
```

```
36052
36052
```

In [65]:

```
false_positive_data = []
for i in range(len(y_test)):
    if (actual_output[i] == 0) &  (pred_output[i] == 1):
        #print(i)
        false_positive_data.append(i)
print(false_positive_data[0:20])
print(len(false_positive_data))
```

```
[4, 13, 72, 96, 142, 144, 153, 186, 198, 200, 202, 212, 220, 224, 237, 282,
285, 304, 310, 339]
2363
```

In [66]:

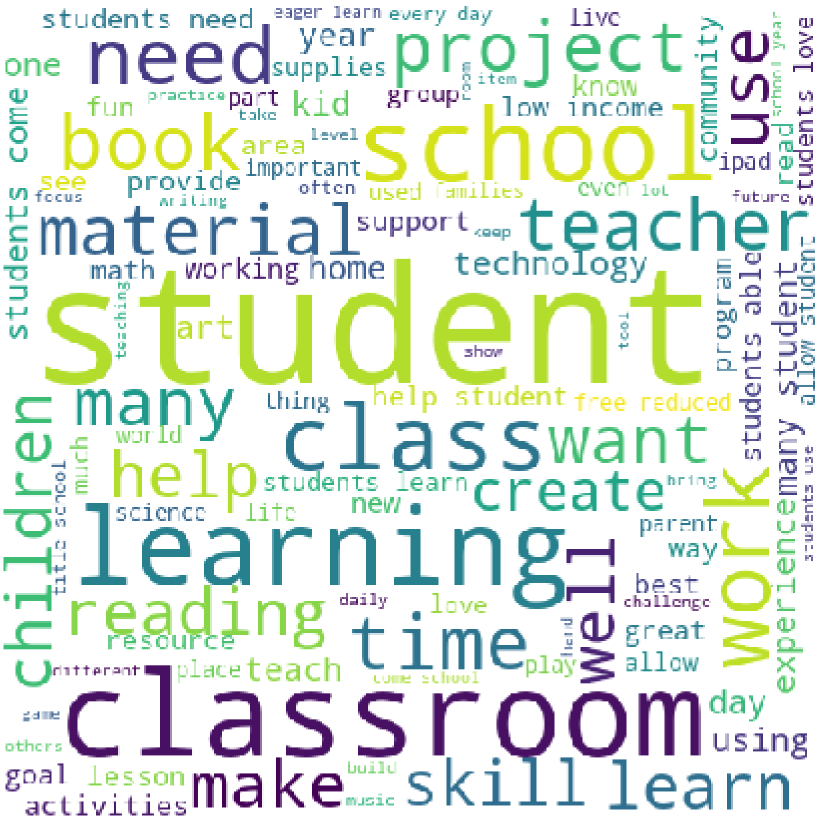
```
false_positive_essay2= []
for i in false_positive_data:
    false_positive_essay2.append(X_test['essay'].values[i])
print(len(false_positive_essay2))
#print(false_positive_essay1[0:20])
```

```
2363
```

1.5.2.5 Plot the WordCloud with the words of essay text of the false positive data points of set 2

In [67]:

```
### Plot the WordCloud with the words of essay text of these false positive data points
# Python program to generate WordCloud

# importing all necessery modules
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = ["nannan"] + list(STOPWORDS)
for val in false_positive_essay2:
    # typecaste each val to string
    val = str(val)
   # split the value
    tokens = val.split()
    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '
wordcloud = WordCloud(width = 500, height = 500,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)
# plot the WordCloud image
plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



1.5.2.6 Plot the box plot with the price of the false positive data points of set 2

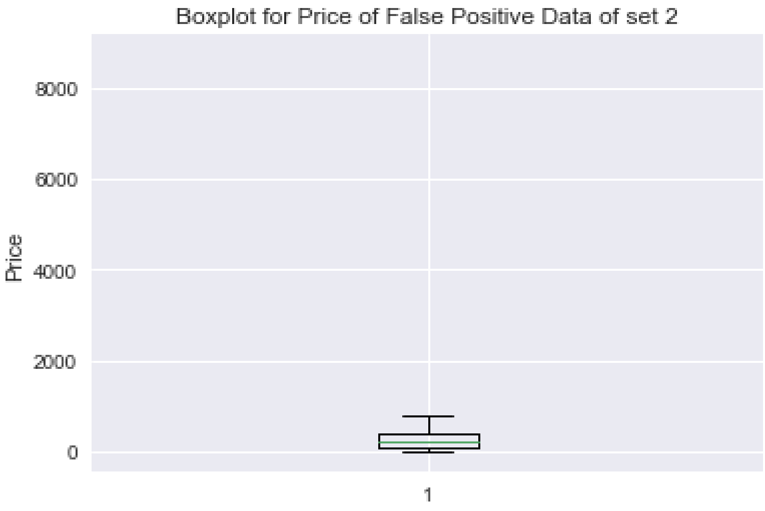
In [68]:

```
false_positive_price2= []
for i in false_positive_data:
    false_positive_price2.append(X_test['price'].values[i])
print(len(false_positive_price2))
#print(false_positive_price2[3])
```

```
2363
```

In [69]:

```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
plt.boxplot([false_positive_price2])
plt.ylabel('Price')
plt.title("Boxplot for Price of False Positive Data of set 2")
plt.grid(True)
plt.show()
```



1.5.2.7 Plot pdf with teacher_number_of_previously_posted_projects of false positive data points of set_2

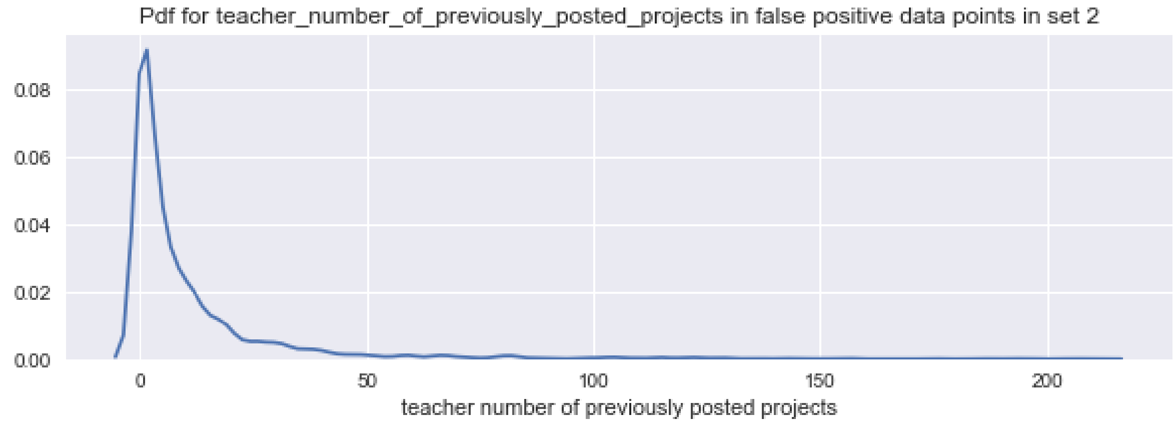
In [70]:

```
false_positive_prev_projects2= []
for i in false_positive_data:
    false_positive_prev_projects2.append(X_test['teacher_number_of_previously_posted_projec
print(len(false_positive_prev_projects2))
#print(false_positive_prev_projects1[3])
```

```
2363
```

In [71]:

```
plt.figure(figsize=(10,3))
sns.distplot(false_positive_prev_projects2, hist=False)
#sns.distplot(rejected_word_count, hist=False, label="Not Approved Projects")
#plt.legend()
plt.title("Pdf for teacher_number_of_previously_posted_projects in false positive data poin
plt.xlabel("teacher number of previously posted projects")
plt.show()
```



1.6 Getting top features using `feature_importances_`

In [72]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# 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 coc
# 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
```

In [73]:

```
dt = DecisionTreeClassifier(class_weight = 'balanced')
dt = dt.fit(X_tr1, y_train)
```

In [74]:

```
print(X_tr1.shape,y_train.shape)
```

```
(73196, 5101) (73196,)
```

In [75]:

```
fi = dt.feature_importances_
print(fi[2])
```

```
0.0001146982967835256
```

In [76]:

```
#print(X_tr1)
```

In [77]:

```
index = []
reqd_features_number = 0
for i in range (len(fi)):
    if fi[i] > 0:
        #print(i)
        index.append(i)
        reqd_features_number+=1
print(reqd_features_number)
```

```
2376
```

In [78]:

```
#print("Index with Non zero feature importance\n",index)
```

In [79]:

```
#reqd_feat = []
#for j in index:
    #print(j)
    #reqd_feat.append(X_tr1[j]) #this is wrong, use[:,j]
```

In [80]:

```
#https://stackoverflow.com/questions/48099075/how-to-get-columns-from-big-sparse-csc-matrix
imp_feat = []
for i in tqdm(index):
    imp_feat.append(X_tr1[:,i])
X_new_tr = hstack(imp_feat)
```

```
100%|██████████████████████████████████████| 2376/2376 [01:25<00:00, 35.59i
t/s]
```

In [81]:

```
X_new_tr.shape
```

Out[81]:

```
(73196, 2376)
```

In [82]:

```
imp_feat = []
for i in tqdm(index):
    imp_feat.append(X_te1[:,i])
X_new_te = hstack(imp_feat)
```

```
100%|██████████████████████████████████████| 2376/2376 [00:43<00:00, 54.55i
t/s]
```

In [83]:

```
X_new_te.shape, y_test.shape
```

Out[83]:

```
((36052, 2376), (36052,))
```

1.6.1 Hyperparameter tuning for new set with non zero features_importance_

In [84]:

```
dt = DecisionTreeClassifier(criterion='gini', class_weight= 'balanced')
parameters = {"max_depth" :[1,5,10,50], "min_samples_split" : [5,10,100,500]}

clf3 = GridSearchCV(dt, parameters, cv=3, scoring='roc_auc')
clf3.fit(X_new_tr, y_train)
```

Out[84]:

```
GridSearchCV(cv=3, error_score='raise',
       estimator=DecisionTreeClassifier(class_weight='balanced', criterion
='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10,
100, 500]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [85]:

```
results = pd.DataFrame.from_dict(clf3.cv_results_)
results = results.sort_values(['param_max_depth'])
#results = results.sort_values(['param_min_samples_split'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
max_depth =  results['param_max_depth']
min_samples_split = results['param_min_samples_split']
results.head()
```

Out[85]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min
0	1.503216	0.021576	0.136734	0.008459	1	
1	1.517649	0.004232	0.139927	0.000973	1	
2	1.510311	0.007365	0.135406	0.007367	1	
3	1.510310	0.007364	0.140616	0.000002	1	
4	5.911041	0.041006	0.140623	0.000008	5	

In [86]:

```
print('Best score: ',clf3.best_score_)
print('Parameters with best score: ',clf3.best_params_)
print('='*75)
print('Train AUC scores')
print(results['mean_train_score'])
print('CV AUC scores')
print(results['mean_test_score'])
```

```
Best score:  0.64450747634415
Parameters with best score:  {'max_depth': 10, 'min_samples_split': 500}
===========================================================================
Train AUC scores
0     0.550555
1     0.550555
2     0.550555
3     0.550555
4     0.647225
5     0.647209
6     0.646377
7     0.645184
8     0.746848
9     0.745357
10    0.728728
11    0.707287
12    0.984469
13    0.976910
14    0.910977
15    0.830848
Name: mean_train_score, dtype: float64
CV AUC scores
0     0.547693
1     0.547693
2     0.547693
3     0.547693
4     0.627792
5     0.627776
6     0.627716
7     0.628280
8     0.635718
9     0.636543
10    0.636546
11    0.644507
12    0.565965
13    0.566692
14    0.590702
15    0.615307
Name: mean_test_score, dtype: float64
```

1.6.2 Representation of results

In [87]:

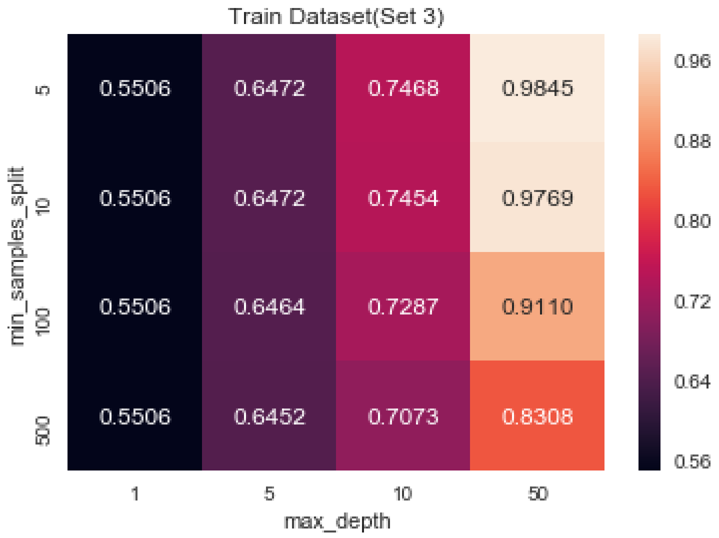
```
'''# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x= min_samples_split, y= max_depth, z=train_auc, name = 'train')
trace2 = go.Scatter3d(x= min_samples_split, y= max_depth, z=cv_auc, name = 'test')
data = [trace1, trace2]

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

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

Out[87]:

```
"# https://plot.ly/python/3d-axes/\ntrace1 (https://plot.ly/python/3d-axes/
\ntrace1) = go.Scatter3d(x= min_samples_split, y= max_depth, z=train_auc, na
me = 'train')\ntrace2 = go.Scatter3d(x= min_samples_split, y= max_depth, z=c
v_auc, name = 'test')\ndata = [trace1, trace2]\n\nlayout = go.Layout(scene =
dict(\n        xaxis = dict(title='min_samples_split'),\n        yaxis = dic
t(title='max_depth'),\n        zaxis = dict(title='AUC'),))\n\nfig = go.Figu
re(data=data, layout=layout)\noffline.iplot(fig, filename='3d-scatter-colors
cale')"
```

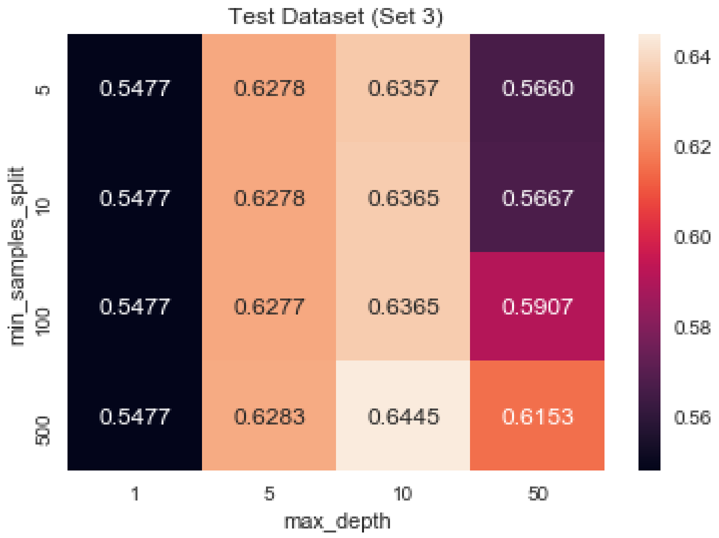
In [88]:

```
uniform_data = pd.DataFrame({'min_samples_split': min_samples_split, 'max_depth': max_depth
uniform_data = uniform_data.pivot("min_samples_split","max_depth","Z")
ax= sns.heatmap(uniform_data, annot= True, fmt= ".4f")
plt.title("Train Dataset(Set 3)")
plt.show()
```



In [89]:

```
uniform_data = pd.DataFrame({'min_samples_split': min_samples_split, 'max_depth': max_depth
uniform_data = uniform_data.pivot("min_samples_split","max_depth","Z")
ax= sns.heatmap(uniform_data, annot= True, fmt= ".4f")
plt.title("Test Dataset (Set 3)")
plt.show()
```



1.6.3 Testing the performance of the model on test data, plotting ROC Curves

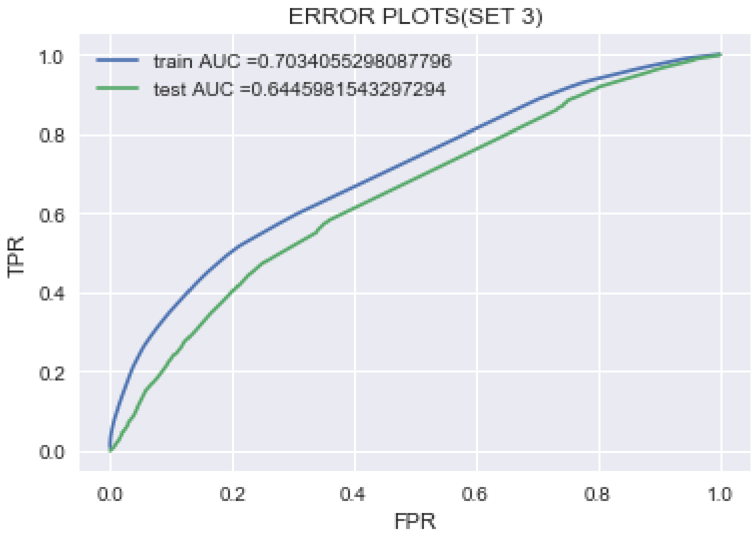
In [90]:

```
best_max_depth= clf3.best_params_['max_depth']
best_min_samples_split= clf3.best_params_['min_samples_split']
print("best_max_depth= ",best_max_depth)
print("best_min_samples_split= ",best_min_samples_split)
```

```
best_max_depth=  10
best_min_samples_split=  500
```

In [91]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.
from sklearn.metrics import roc_curve, auc


dt_set3 = DecisionTreeClassifier(max_depth= best_max_depth, min_samples_split= best_min_sam
dt_set3.fit(X_new_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = prob_predict(dt_set3, X_new_tr)
y_test_pred = prob_predict(dt_set3, X_new_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ERROR PLOTS(SET 3)")
plt.grid(True)
plt.show()
```



1.6.3 Confusion Matrix

In [92]:

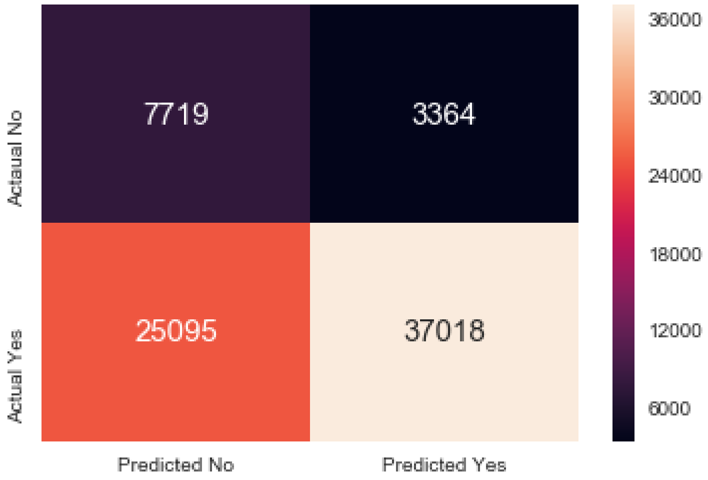
```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
```

```
the maximum value of tpr*(1-fpr) 0.41508224120905435 for threshold 0.453
```

In [93]:

```
print("Train Confusion Matrix")
get_confusion_matrix(y_train, y_train_pred)
```

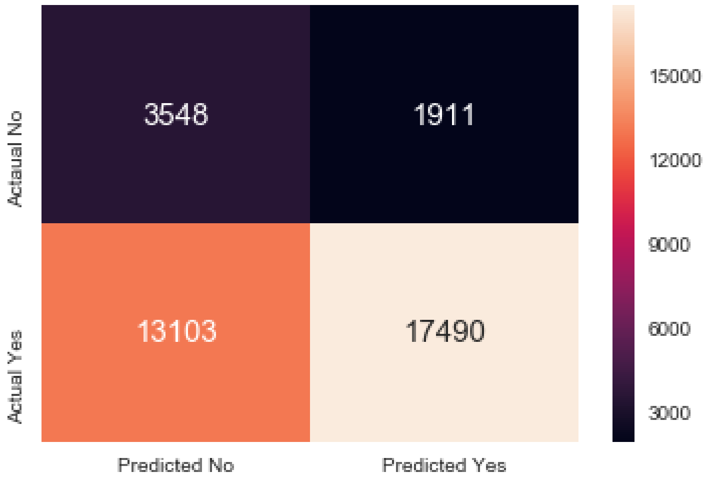
```
Train Confusion Matrix
```



In [94]:

```
print("Test Confusion Matrix")
get_confusion_matrix(y_test, y_test_pred)
```

```
Test Confusion Matrix
```



In [95]:

```
#get all the false positive data points with test dataset
actual_output = y_test
pred_output = predict_with_best_t(y_test_pred, best_t)
print(len(actual_output))
print(len(pred_output))
```

```
36052
36052
```

In [96]:

```
false_positive_data = []
for i in range(len(y_test)):
    if (actual_output[i] == 0) &  (pred_output[i] == 1):
        #print(i)
        false_positive_data.append(i)
print(false_positive_data[0:20])
print(len(false_positive_data))
```

```
[4, 57, 72, 142, 144, 153, 162, 198, 202, 212, 224, 241, 260, 283, 285, 304,
339, 368, 388, 417]
1911
```

In [97]:

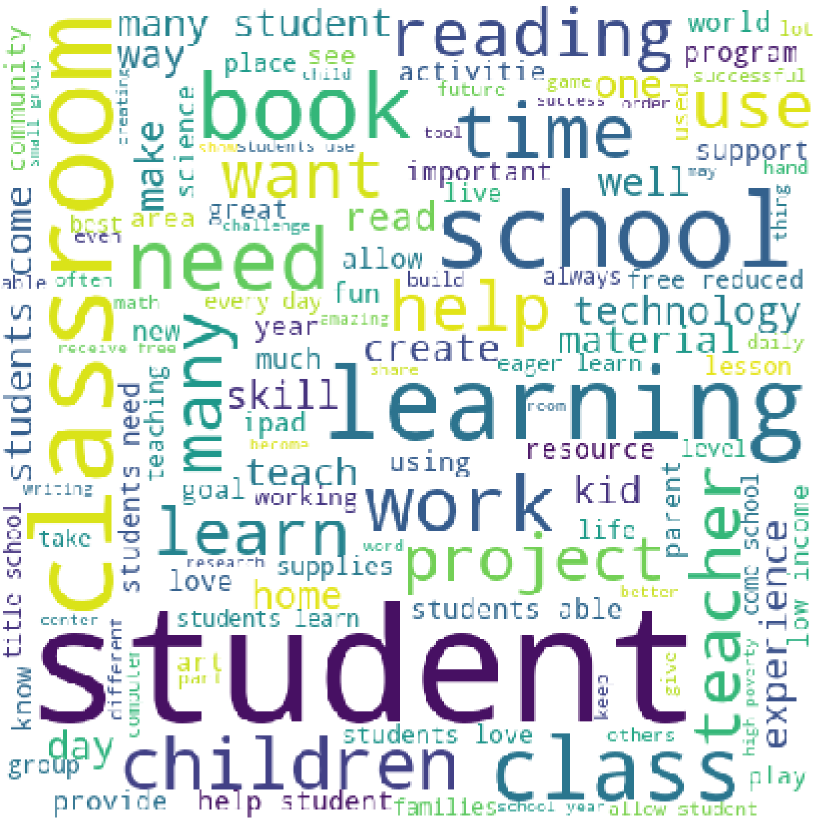
```
false_positive_essay3= []
for i in false_positive_data:
    false_positive_essay3.append(X_test['essay'].values[i])
print(len(false_positive_essay3))
#print(false_positive_essay3[0:20])
```

```
1911
```

1.6.4 Plot the WordCloud with the words of essay text of the false positive data points of set 3

In [98]:

```
### Plot the WordCloud with the words of essay text of these false positive data points
# Python program to generate WordCloud

# importing all necessery modules
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = ["nannan"] + list(STOPWORDS)
for val in false_positive_essay3:
    # typecaste each val to string
    val = str(val)
   # split the value
    tokens = val.split()
    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '
wordcloud = WordCloud(width = 500, height = 500,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)
# plot the WordCloud image
plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



1.6.5 Plot the box plot with the price of the false positive data points of set 3

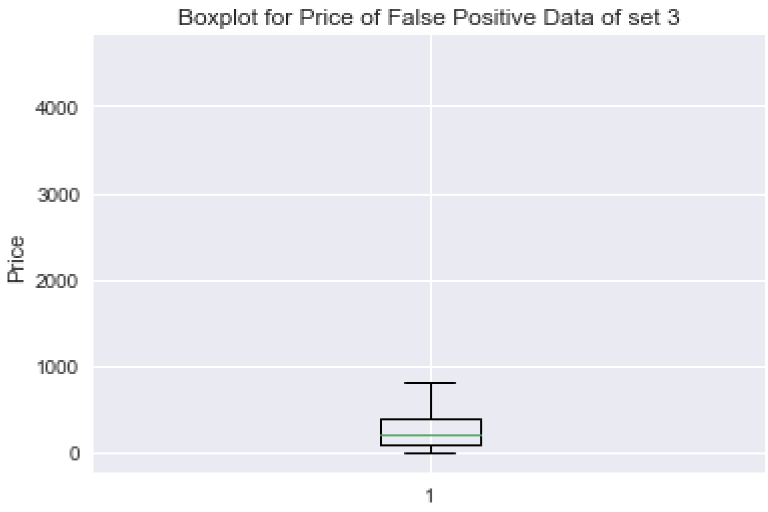
In [99]:

```
false_positive_price3= []
for i in false_positive_data:
    false_positive_price3.append(X_test['price'].values[i])
print(len(false_positive_price3))
#print(false_positive_price2[3])
```

```
1911
```

In [100]:

```
# https://glowingpython.blogspot.com/2012/09/boxplot-with-matplotlib.html
plt.boxplot([false_positive_price3])
plt.ylabel('Price')
plt.title("Boxplot for Price of False Positive Data of set 3")
plt.grid(True)
plt.show()
```



1.6.6 Plot pdf with teacher_number_of_previously_posted_projects of false positive data points of set_3

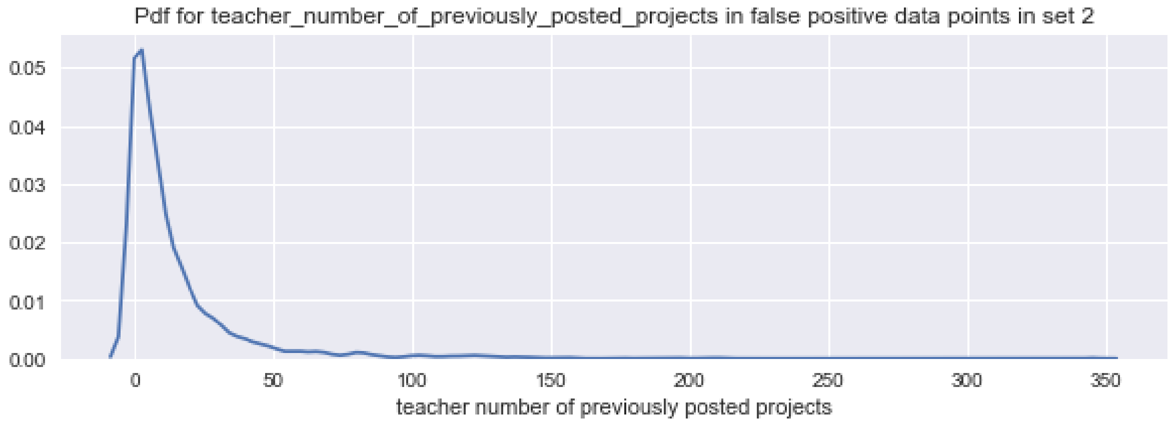
In [101]:

```
false_positive_prev_projects3= []
for i in false_positive_data:
    false_positive_prev_projects3.append(X_test['teacher_number_of_previously_posted_projec
print(len(false_positive_prev_projects3))
#print(false_positive_prev_projects1[3])
```

```
1911
```

In [102]:

```
plt.figure(figsize=(10,3))
sns.distplot(false_positive_prev_projects3, hist=False)
#sns.distplot(rejected_word_count, hist=False, label="Not Approved Projects")
#plt.legend()
plt.title("Pdf for teacher_number_of_previously_posted_projects in false positive data poin
plt.xlabel("teacher number of previously posted projects")
plt.show()
```



2. Summary

In [1]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer","Model","Max_Depth","Min_Samples_Split", "Train AUC", "Test A
x.add_row(["TFIDF","DECISION_TREE_CLASSIFIER", 10, 500, 0.70, 0.64 ])
x.add_row(["------------------","--------------------","---------------","-----------","---
x.add_row(["TFIDF_W2V","DECISION_TREE_CLASSIFIER", 5, 100, 0.64, 0.63])
x.add_row(["------------------","--------------------","---------------","-----------","---
x.add_row(["TFIDF(NEW SET)","DECISION_TREE_CLASSIFIER", 10, 500, 0.70, 0.64])
print(x)
```

```
+--------------------+--------------------------+-----------------+---------
----------+-------------+-------------+
|     Vectorizer     |          Model           |    Max_Depth    | Min_Samp
les_Split |  Train AUC  |   Test Auc  |
+--------------------+--------------------------+-----------------+---------
----------+-------------+-------------+
|       TFIDF        | DECISION_TREE_CLASSIFIER |        10       |        5
00        |     0.7     |     0.64    |
| ------------------ |   --------------------   | --------------- |    -----
------    | ----------- | ----------- |
|     TFIDF_W2V      | DECISION_TREE_CLASSIFIER |        5        |        1
00        |     0.64    |     0.63    |
| ------------------ |   --------------------   | --------------- |    -----
------    | ----------- | ----------- |
|   TFIDF(NEW SET)   | DECISION_TREE_CLASSIFIER |        10       |        5
00        |     0.7     |     0.64    |
+--------------------+--------------------------+-----------------+---------
----------+-------------+-------------+
```