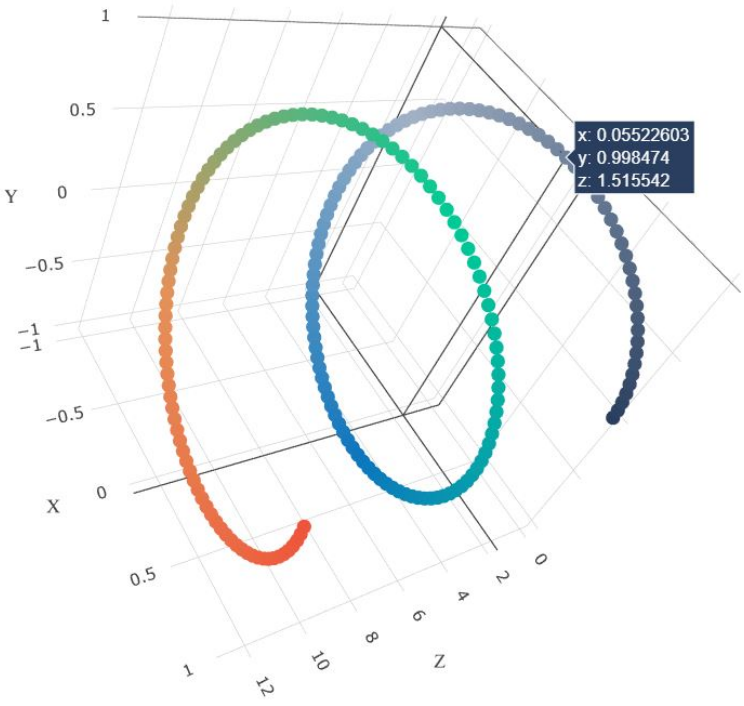
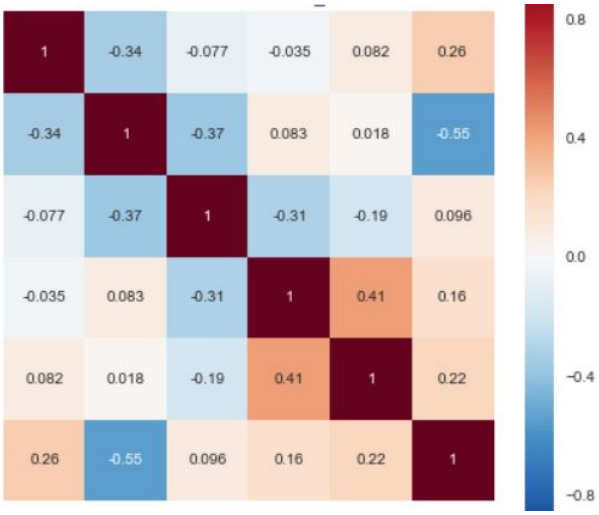
1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)
    - Set 2: categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)
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/) 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*

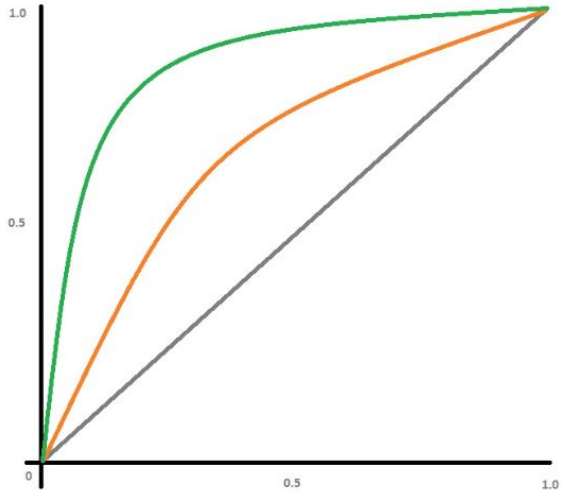
or

- 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) with rows as **min_sample_split**, 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/) 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/) 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`

In [10]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import nltk

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.metrics import roc_auc_score,roc_curve,auc


import re
import pickle
from tqdm import tqdm
import os


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

executed in 1.81s, finished 15:18:46 2020-10-26

```
Requirement already satisfied: chart_studio in c:\programdata\anaconda3\lib
\site-packages (1.1.0)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-pack
ages (from chart_studio) (1.11.0)
Requirement already satisfied: requests in c:\programdata\anaconda3\lib\site
-packages (from chart_studio) (2.18.4)
Requirement already satisfied: retrying>=1.3.3 in c:\programdata\anaconda3\l
ib\site-packages (from chart_studio) (1.3.3)
Requirement already satisfied: plotly in c:\programdata\anaconda3\lib\site-p
ackages (from chart_studio) (4.9.0)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in c:\programdata\anaco
nda3\lib\site-packages (from requests->chart_studio) (3.0.4)
Requirement already satisfied: idna<2.7,>=2.5 in c:\programdata\anaconda3\li
b\site-packages (from requests->chart_studio) (2.6)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in c:\programdata\anaco
nda3\lib\site-packages (from requests->chart_studio) (1.22)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anaconda
3\lib\site-packages (from requests->chart_studio) (2020.6.20)

mysql-connector-python 8.0.18 requires protobuf>=3.0.0, which is not install
ed.
distributed 1.21.8 requires msgpack, which is not installed.
jupyterlab-server 1.0.0 has requirement jsonschema>=3.0.1, but you'll have j
sonschema 2.6.0 which is incompatible.
You are using pip version 10.0.1, however version 20.2.4 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip'
command.
```

In [11]:

```
#please use below code to load glove vectors
import pickle

with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())   # it gives the name of the words

# glove_words contains the set of unique words in glove vector
```

executed in 379ms, finished 15:18:47 2020-10-26

In [12]:

```
# we can see what does contains

# model
```

executed in 15ms, finished 15:18:47 2020-10-26

In [13]:

```
# glove_words

# unique words present in glove vector
```

executed in 14ms, finished 15:18:47 2020-10-26

In [14]:

```
data = pd.read_csv('preprocessed_data.csv')
data.head()
```

executed in 1.49s, finished 15:18:48 2020-10-26

Out[14]:

	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 [15]:

```
data.shape

# it measn there are 109248 rows i.e. project submitted and 9 features
```

executed in 14ms, finished 15:18:48 2020-10-26

Out[15]:

```
(109248, 9)
```

In [16]:

```
# feature names

data.columns.values
```

executed in 14ms, finished 15:18:48 2020-10-26

Out[16]:

```
array(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects',
       'project_is_approved', 'clean_categories', 'clean_subcategories',
       'essay', 'price'], dtype=object)
```

In [17]:

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

executed in 11ms, finished 15:18:48 2020-10-26

```
[nltk_data] Downloading package vader_lexicon to C:\Users\yogesh
[nltk_data]     pal\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[17]:

```
True
```

In [18]:

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer

sid = SentimentIntensityAnalyzer()

# taking all the essay and find the sentiment score

score = []                  # store the sentiment score for each essay in score
for sen in data['essay']:
  ss=(sid.polarity_scores(sen))
  score.append(ss)

# it contains dictionary key
key = list(ss.keys())

# stores all the score in the list type so it can become easier to make dataframe
value= []
for val in score:
  value.append(val.values())

# make the dataframe with score and key as column name
newdf = pd.DataFrame(value,columns=key)
newdf.head()

# we can as we 109248 project review in data so newdf will also have 109248 rows i.e. for e
```

executed in 2m 57s, finished 15:21:45 2020-10-26

Out[18]:

	neg	neu	pos	compound
0	0.013	0.783	0.205	0.9867
1	0.072	0.680	0.248	0.9897
2	0.017	0.721	0.262	0.9860
3	0.030	0.783	0.187	0.9524
4	0.029	0.683	0.288	0.9873

In [19]:

```
# adding/joining newdf dataframe with data along column wise

data = pd.concat([data,newdf],axis =1)
data.head()
```

executed in 30ms, finished 15:21:45 2020-10-26

Out[19]:

	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 [20]:

```
data.shape

# we see there are 13 columns so 4 columns are added newly from newdf
```

executed in 15ms, finished 15:21:45 2020-10-26

Out[20]:

```
(109248, 13)
```

In [21]:

```
data.columns.values
```

executed in 14ms, finished 15:21:45 2020-10-26

Out[21]:

```
array(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects',
       'project_is_approved', 'clean_categories', 'clean_subcategories',
       'essay', 'price', 'neg', 'neu', 'pos', 'compound'], dtype=object)
```

In [22]:

```
X = data.drop('project_is_approved',axis=1)
Y = data['project_is_approved']
```

executed in 46ms, finished 15:21:45 2020-10-26

splitting dataset into train and test

In [23]:

```
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,stratify=Y)
```

executed in 62ms, finished 15:21:45 2020-10-26

vectorize Text data i.e. essay

In [24]:

```
# perform encoding on text feature BOW
# fearure_names contains all the feature names for different all the features present in da

feature_names_tfidf = []
feature_names_tfidfW2v = []


######################################### TFIDF ESSAY ##############################


###### applying TFIDF vectorizer

vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1,3))

######### build the vocabulary of words in essay present in train data
vectorizer.fit(X_train['essay'].values)

dictionary= dict(zip(vectorizer.get_feature_names(),list(vectorizer.idf_)))

# dictionary
```

executed in 1m 3.99s, finished 15:22:49 2020-10-26

In [25]:

```
tfidf_words = set(vectorizer.get_feature_names())


feature_names_tfidf.extend(vectorizer.get_feature_names())
feature_names_tfidfW2v.extend(vectorizer.get_feature_names())



# apply transform on both test and train data
x_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
x_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)


print('after tfidf \n',x_train_essay_tfidf.shape, y_train.shape)
print(x_test_essay_tfidf.shape, y_test.shape)
print('total number of features in essay_tfidf_vector',len(feature_names_tfidf))
print('\n')
```

executed in 44.2s, finished 15:23:34 2020-10-26

```
after tfidf
 (73196, 10000) (73196,)
(36052, 10000) (36052,)
total number of features in essay_tfidf_vector 10000
```

In [26]:

```
########################################## TFIDF W2V ESSAY ###################################



X_train_tfidf_w2v_vectors = [] # the avg-w2v for each sentence/review is stored in this lis
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_train_tfidf_w2v_vectors.append(vector)



X_test_tfidf_w2v_vectors = [] # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettin
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    X_test_tfidf_w2v_vectors.append(vector)


print('X_train_tfidf_w2v_vectors',len(X_train_tfidf_w2v_vectors))
print('X_test_tfidf_w2v_vectors',len(X_test_tfidf_w2v_vectors))
```

executed in 4m 30s, finished 15:28:03 2020-10-26

```
100%|████████████████████████████████████████████████████████████████████
████| 73196/73196 [03:02<00:00, 402.13it/s]
100%|████████████████████████████████████████████████████████████████████
████| 36052/36052 [01:27<00:00, 411.99it/s]

X_train_tfidf_w2v_vectors 73196
X_test_tfidf_w2v_vectors 36052
```

In [27]:

```
# X_train_tfidf_w2v_vectors
```

executed in 14ms, finished 15:28:03 2020-10-26

encoding categorical feature and numerical feature

In [28]:

```
################################  binary BOW on school state ################################


print('\nBinary BOW encoding on school state')


vectorizer = CountVectorizer(binary=True)

vectorizer.fit(X_train['school_state'].values)

feature_names_tfidf.extend(vectorizer.get_feature_names())
feature_names_tfidfW2v.extend(vectorizer.get_feature_names())


# apply transform
x_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
x_test_state_ohe = vectorizer.transform(X_test['school_state'].values)


print("\nAfter vectorizations")
print(x_train_state_ohe.shape, y_train.shape)
print(x_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())


print('\ntotal number of features',len(feature_names_tfidf))
print('total number of features',len(feature_names_tfidfW2v))
print("="*180)


############################ Binary BOW teacher prefix ###################################

print('\nencoding of teacher prefix\n')

vectorizer = CountVectorizer(binary=True)

vectorizer.fit(X_train['teacher_prefix'].values)

feature_names_tfidf.extend(vectorizer.get_feature_names())
feature_names_tfidfW2v.extend(vectorizer.get_feature_names())


# apply transform
x_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
x_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)


print("\nAfter vectorizations")
print(x_train_teacher_ohe.shape, y_train.shape)
print(x_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())


print('\ntotal number of features',len(feature_names_tfidf))
print('total number of features',len(feature_names_tfidfW2v))
print("="*180)
```

```
############################### Binary BOW on project_grade_category#####################
print('\n\nencoding of project grade category \n')

vectorizer = CountVectorizer(binary=True)
# apply fit
vectorizer.fit(X_train['project_grade_category'].values)


feature_names_tfidf.extend(vectorizer.get_feature_names())
feature_names_tfidfW2v.extend(vectorizer.get_feature_names())


# apply transform
x_train_pro_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
x_test_pro_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)


print("After vectorizations")
print(x_train_pro_grade_ohe.shape, y_train.shape)
print(x_test_pro_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())


print('\ntotal number of features',len(feature_names_tfidf))
print('total number of features',len(feature_names_tfidfW2v))
print("="*180)


################### binary BOW on clean  categories ########################################
print('\n\n encoding of clean category\n')

vectorizer = CountVectorizer(binary=True)
vectorizer.fit(X_train['clean_categories'].values)


feature_names_tfidf.extend(vectorizer.get_feature_names())
feature_names_tfidfW2v.extend(vectorizer.get_feature_names())


# apply transform
x_train_clean_category_ohe = vectorizer.transform(X_train['clean_categories'].values)
x_test_clean_category_ohe = vectorizer.transform(X_test['clean_categories'].values)


print("After vectorizations")
print(x_train_clean_category_ohe.shape, y_train.shape)
print(x_test_clean_category_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())


print('\ntotal number of features',len(feature_names_tfidf))
print('total number of features',len(feature_names_tfidfW2v))
print("="*180)



################ clean subcategories ##########################################
print('\n\n encoding of clean subcategories\n')


vectorizer =CountVectorizer(binary=True)
```

```
vectorizer.fit(X_train['clean_subcategories'].values)


feature_names_tfidf.extend(vectorizer.get_feature_names())
feature_names_tfidfW2v.extend(vectorizer.get_feature_names())


# apply transform
x_train_clean_subcategory_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
x_test_clean_subcategory_ohe = vectorizer.transform(X_test['clean_subcategories'].values)


print("After vectorizations")
print(x_train_clean_subcategory_ohe.shape, y_train.shape)
print(x_test_clean_subcategory_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())


print('\ntotal number of features',len(feature_names_tfidf))
print('total number of features',len(feature_names_tfidfW2v))
print("="*180)



################### price ############################################################
print('\n\nenoding of price\n')

feature_names_tfidf.append('price')
feature_names_tfidfW2v.append('price')


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 your data has a single feature
# array.reshape(1, -1) if it contains a single sample.

normalizer.fit(X_train['price'].values.reshape(1,-1))
x_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1))
x_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))


x_train_price_norm = x_train_price_norm.reshape(-1,1)
x_test_price_norm = x_test_price_norm.reshape(-1,1)


print("After normalizations")
print(x_train_price_norm.shape, y_train.shape)
print(x_test_price_norm.shape, y_test.shape)
#print('\n total number of features',len(feature_names))
print("="*100)



########## teacher_number_of_previously posted projects###################
print('\n\nenoding of teacher_number_of_previously_posted_project\n')
```

```
feature_names_tfidf.append('teacher_previous_posted_project')
feature_names_tfidfW2v.append('teacher_previous_posted_project')

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_teacher_project_norm = normalizer.transform(X_train['teacher_number_of_previously_p
x_test_teacher_project_norm = normalizer.transform(X_test['teacher_number_of_previously_pos

x_train_teacher_project_norm = x_train_teacher_project_norm.reshape(-1,1)
x_test_teacher_project_norm = x_test_teacher_project_norm.reshape(-1,1)

print("After normalizations")
print(x_train_teacher_project_norm.shape, y_train.shape)
print(x_test_teacher_project_norm.shape, y_test.shape)

########################################################## neg normalize ######################

print('\n\n normalize neg \n')


feature_names_tfidf.append('neg')
feature_names_tfidfW2v.append('neg')

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['neg'].values.reshape(1,-1))
x_train_neg_norm = normalizer.transform(X_train['neg'].values.reshape(1,-1))
x_test_neg_norm = normalizer.transform(X_test['neg'].values.reshape(1,-1))

x_train_neg_norm = x_train_neg_norm.reshape(-1,1)
x_test_neg_norm = x_test_neg_norm.reshape(-1,1)

print("After normalizations")
print(x_train_neg_norm.shape, y_train.shape)
print(x_test_neg_norm.shape, y_test.shape)


########################################################## neu normalize ######################

print('\n\n normalize neu \n')


feature_names_tfidf.append('neu')
feature_names_tfidfW2v.append('neu')

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['neu'].values.reshape(1,-1))
x_train_neu_norm = normalizer.transform(X_train['neu'].values.reshape(1,-1))
x_test_neu_norm = normalizer.transform(X_test['neu'].values.reshape(1,-1))

x_train_neu_norm = x_train_neu_norm.reshape(-1,1)
x_test_neu_norm = x_test_neu_norm.reshape(-1,1)

print("After normalizations")
print(x_train_neu_norm.shape, y_train.shape)
print(x_test_neu_norm.shape, y_test.shape)


########################################################## pos normalize ######################

print('\n\n normalize pos \n')


feature_names_tfidf.append('pos')
feature_names_tfidfW2v.append('pos')

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['pos'].values.reshape(1,-1))
x_train_pos_norm = normalizer.transform(X_train['pos'].values.reshape(1,-1))
x_test_pos_norm = normalizer.transform(X_test['pos'].values.reshape(1,-1))

x_train_pos_norm = x_train_pos_norm.reshape(-1,1)
x_test_pos_norm = x_test_pos_norm.reshape(-1,1)

print("After normalizations")
print(x_train_pos_norm.shape, y_train.shape)
print(x_test_pos_norm.shape, y_test.shape)


########################################################  normalize compound ###############

print('\n\n normalize compound \n')


feature_names_tfidf.append('compound')
feature_names_tfidfW2v.append('compound')

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['compound'].values.reshape(1,-1))
x_train_compound_norm = normalizer.transform(X_train['compound'].values.reshape(1,-1))
x_test_compound_norm = normalizer.transform(X_test['compound'].values.reshape(1,-1))

x_train_compound_norm = x_train_compound_norm.reshape(-1,1)
x_test_compound_norm = x_test_compound_norm.reshape(-1,1)

print("After normalizations")
print(x_train_compound_norm.shape, y_train.shape)
print(x_test_compound_norm.shape, y_test.shape)
```

executed in 3.25s, finished 15:28:06 2020-10-26

```
Binary BOW encoding on school state

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

total number of features 10051
total number of features 10051
==========================================================================
==========================================================================
==================================

encoding of teacher prefix
```

In [29]:

```
# merge all the sparse matrix

# concatenate the all the train and test for all the features aatribute

# merge two sparse matrix


from scipy.sparse import hstack

#  SET 1 -- categorical feature + numerical feature + tfidf(essay)

X_tr_tfidf = hstack((x_train_essay_tfidf,x_train_state_ohe,x_train_teacher_ohe,x_train_pro_
X_te_tfidf = hstack((x_test_essay_tfidf,x_test_state_ohe,x_test_teacher_ohe,x_test_pro_grad

print('final shape of SET 1 -- categorical feature + numerical feature + tfidf(essay) \n')
print(X_tr_tfidf.shape,y_train.shape)
print(X_te_tfidf.shape,y_test.shape)

# SET 2 --  categorical feature + numerical feature + tfidf W2v(essay)

X_tr_tfidfW2v = hstack((X_train_tfidf_w2v_vectors,x_train_state_ohe,x_train_teacher_ohe,x_t
X_te_tfidfW2v = hstack((X_test_tfidf_w2v_vectors,x_test_state_ohe,x_test_teacher_ohe,x_test

print('\nfinal shape of SET 2 --  categorical feature + numerical feature + tfidf W2v(essay
print(X_tr_tfidfW2v.shape,y_train.shape)
print(X_te_tfidfW2v.shape,y_test.shape)
```

executed in 2.96s, finished 15:28:09 2020-10-26

```
final shape of SET 1 -- categorical feature + numerical feature + tfidf(essa
y)

(73196, 10105) (73196,)
(36052, 10105) (36052,)

final shape of SET 2 --  categorical feature + numerical feature + tfidf W2v
(essay)

(73196, 405) (73196,)
(36052, 405) (36052,)
```

SET 1

hyperparater tuning for SET 1

In [30]:

```
# perform hyperparamter tuning on Decision tree by the help of grid search / random search
# https://www.dezyre.com/recipes/optimize-hyper-parameters-of-decisiontree-model-using-gria


from pandas import DataFrame
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

# best `depth` in range [1, 5, 10, 50], and the best `min_samples_split` in range [5, 10, 1

depth = [1,5,10,50]
min_samples_split = [5,10,100,500]

parameter = dict(max_depth = depth ,min_samples_split = min_samples_split)

DT = DecisionTreeClassifier(class_weight = 'balanced')

# apply grid sarch on Decision tree

clf = GridSearchCV(DT,parameter,cv = 5,scoring = 'roc_auc',return_train_score='True')
clf.fit(X_tr_tfidf,y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
```

executed in 39m 35s, finished 16:07:44 2020-10-26

In [31]:

```
results.head()
```

executed in 45ms, finished 16:07:44 2020-10-26

Out[31]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min
0	1.260289	0.024922	0.061656	0.003520	1	
1	1.287789	0.056548	0.059839	0.002097	1	
2	1.265244	0.029770	0.061625	0.002139	1	
3	1.284402	0.065349	0.059040	0.000750	1	
4	5.824360	0.137677	0.061028	0.003467	5	

5 rows × 22 columns

In [32]:

```
train_auc= results['mean_train_score']
cv_auc = results['mean_test_score']

max_depth_values=results['param_max_depth']
min_sample_split_values = results['param_min_samples_split']

# list of values

train_auc = list(train_auc)
cv_auc = list(cv_auc)
```

executed in 14ms, finished 16:07:44 2020-10-26

In [33]:

```
clf.best_params_
```

executed in 14ms, finished 16:07:44 2020-10-26

Out[33]:

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

In [34]:

```
print(train_auc)
print(list(max_depth_values))
print(list(min_sample_split_values))


print('\n',cv_auc)
```

executed in 14ms, finished 16:07:44 2020-10-26

```
[0.5500103334866484, 0.5500103334866484, 0.5500103334866484, 0.5500103334866
484, 0.6486184636130436, 0.6486063991903429, 0.6482478754822726, 0.647436785
4057606, 0.742801048429542, 0.7416329317256154, 0.7287701960795385, 0.712141
0682102615, 0.9813428000546708, 0.9742151425768906, 0.9240199305123955, 0.84
59188979103228]
[1, 1, 1, 1, 5, 5, 5, 5, 10, 10, 10, 10, 50, 50, 50, 50]
[5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500, 5, 10, 100, 500]

 [0.5494695093391988, 0.5494695093391988, 0.5494695093391988, 0.549469509339
1988, 0.629407158244523, 0.6293910037281808, 0.629316153305057, 0.6294174567
836158, 0.6401760976898317, 0.6394233559500185, 0.639937932254179, 0.6464019
206603576, 0.5603030783371067, 0.5643324658520512, 0.5839991811504068, 0.607
2626793169466]
```

plot the performance of model both on train data and cross validation data for each hyper parameter, using the heatmap with row as min_sample_split and column as max_depth and values inside the cell representing auc value

for train data and train score

In [35]:

```
# creating new dataframe for plotting heatmap
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

heap_data = {'max_depth':max_depth_values,'min_sample_split':min_sample_split_values,'trair

heap_dataframe = pd.DataFrame(heap_data)

heap_dataframe
```

executed in 14ms, finished 16:07:44 2020-10-26

Out[35]:

	max_depth	min_sample_split	train_score
0	1	5	0.550010
1	1	10	0.550010
2	1	100	0.550010
3	1	500	0.550010
4	5	5	0.648618
5	5	10	0.648606
6	5	100	0.648248
7	5	500	0.647437
8	10	5	0.742801
9	10	10	0.741633
10	10	100	0.728770
11	10	500	0.712141
12	50	5	0.981343
13	50	10	0.974215
14	50	100	0.924020
15	50	500	0.845919

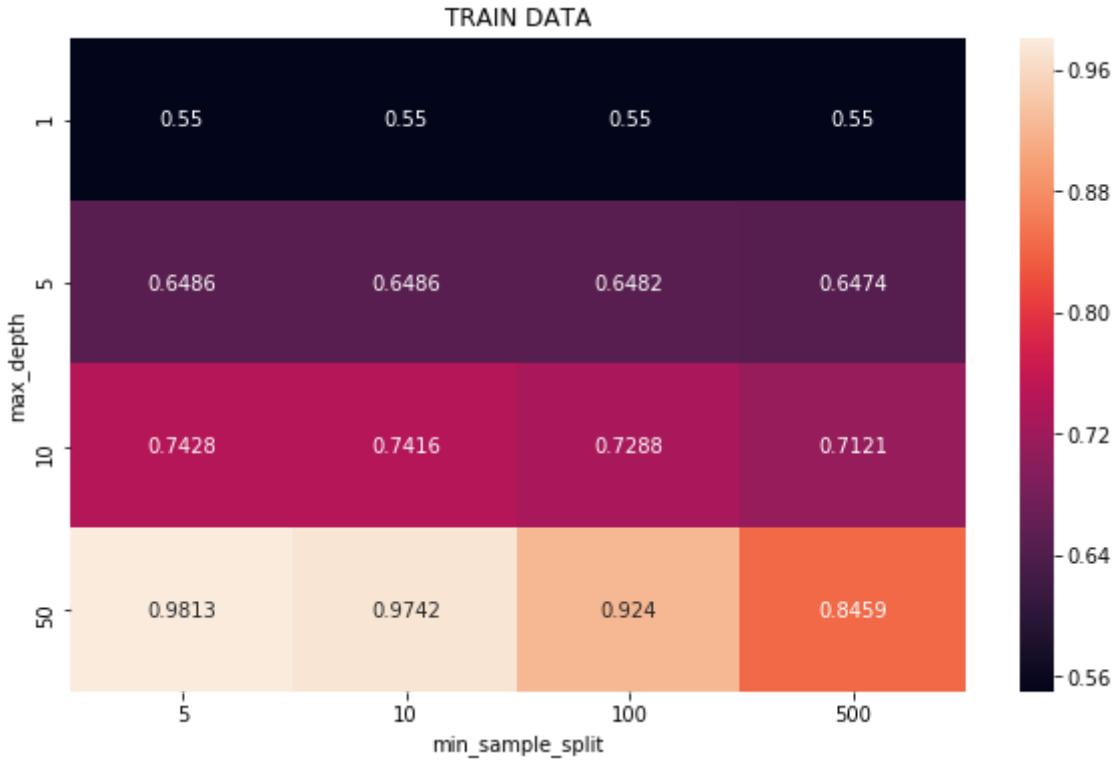
In [36]:

```
# i have done one mistake i had taken max depth on rows instead of column . I don't want to
# turning takes a lot of time .

# in SET 2 i have done it correct

da_heapmap = heap_dataframe.pivot("max_depth", "min_sample_split", "train_score")
fig,ax= plt.subplots(figsize=(10,6))
ax = sns.heatmap(da_heapmap,annot=True,fmt='.4g')
ax.set_title('TRAIN DATA')
```

executed in 342ms, finished 16:07:44 2020-10-26

Out[36]:

```
Text(0.5,1,'TRAIN DATA')
```



cv score

In [37]:

```
heap_data = {'max_depth':max_depth_values,'min_sample_split':min_sample_split_values,'cv_sc

heap_dataframe = pd.DataFrame(heap_data)

heap_dataframe
```

executed in 14ms, finished 16:07:44 2020-10-26

Out[37]:

	max_depth	min_sample_split	cv_score
0	1	5	0.549470
1	1	10	0.549470
2	1	100	0.549470
3	1	500	0.549470
4	5	5	0.629407
5	5	10	0.629391
6	5	100	0.629316
7	5	500	0.629417
8	10	5	0.640176
9	10	10	0.639423

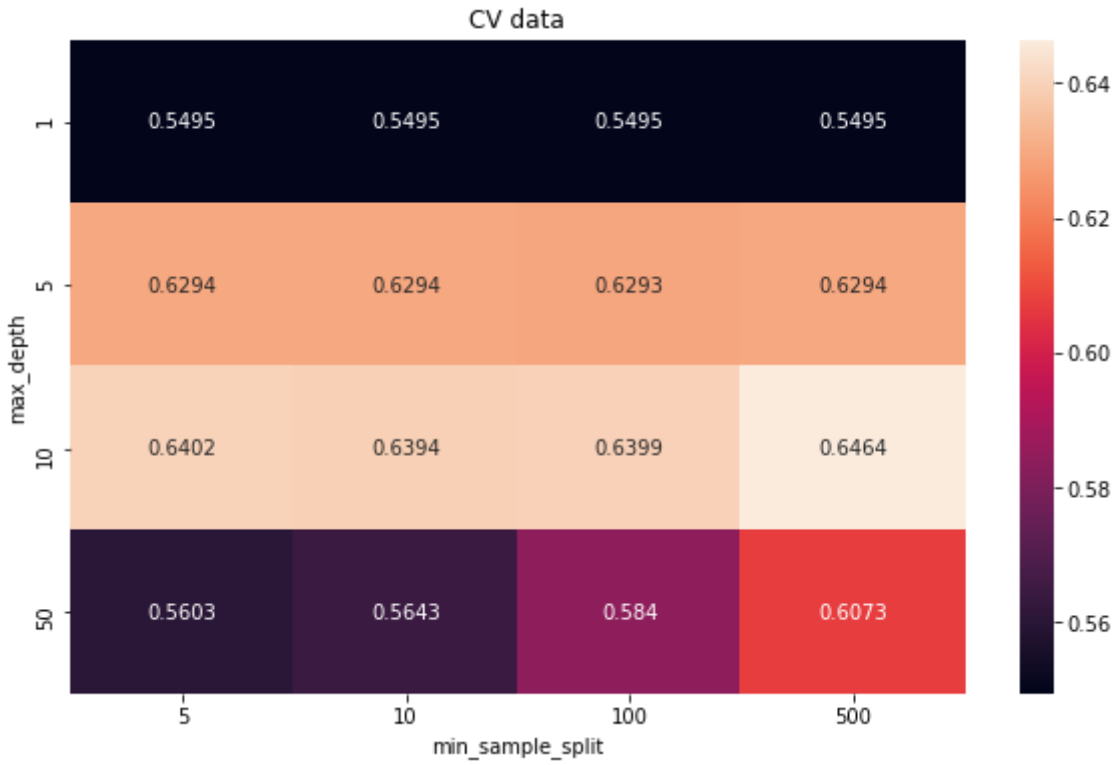
In [38]:

```
# https://seaborn.pydata.org/generated/seaborn.heatmap.html

da_heapmap = heap_dataframe.pivot("max_depth", "min_sample_split", "cv_score")
fig,ax= plt.subplots(figsize=(10,6))
ax = sns.heatmap(da_heapmap,annot=True,fmt='.4g')
ax.set_title('CV data')
```

executed in 190ms, finished 16:07:45 2020-10-26

Out[38]:

```
Text(0.5,1,'CV data')
```



ROC AUC plot for SET 1

In [39]:

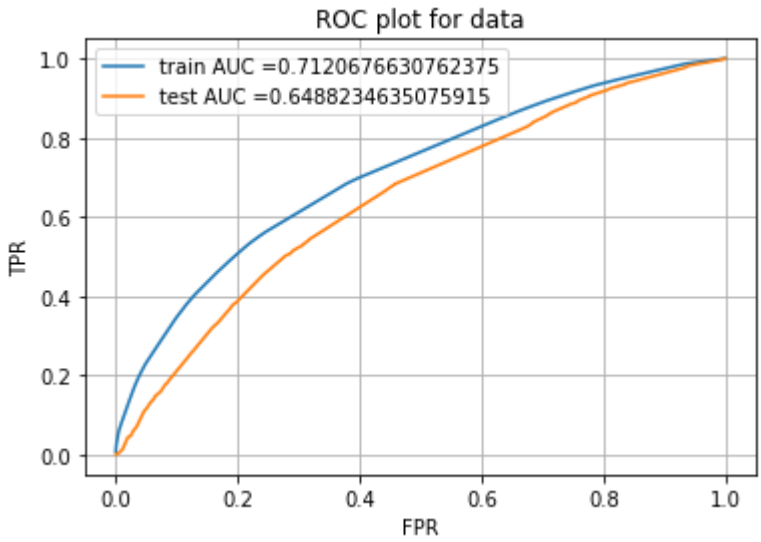
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 4900
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

executed in 15ms, finished 16:07:45 2020-10-26

In [40]:

```
y_train_pred = []  # both will contain the probability estimates
y_test_pred = []

dectree = DecisionTreeClassifier(max_depth = 10, min_samples_split =500,class_weight='balar

dectree.fit(X_tr_tfidf,y_train)


y_train_pred =  batch_predict(dectree,X_tr_tfidf)
y_test_pred = batch_predict(dectree,X_te_tfidf)

# by the help of of probability estimate and true_predicted we calculated the roc-auc

train_fpr,train_tpr,train_threshold = roc_curve(y_train,y_train_pred)
test_fpr,test_tpr,test_threshold = 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('ROC plot for data')
plt.grid()


### by mistake I written title as ROC plot for train data . but in actually its for both tr
```

executed in 18.2s, finished 16:08:03 2020-10-26



confusion matrix for SET 1

In [41]:

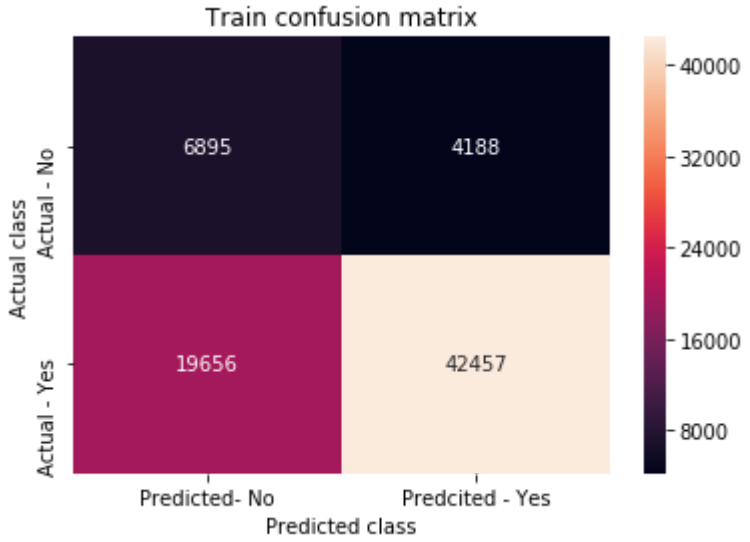
```
def find_best_threshold(threshold,fpr,tpr):
  t= threshold[np.argmax(tpr*(1-fpr))]
  print('maximum value of tpr*(1-fpr)',max(tpr*(1-fpr)),'for threshold',np.round(t,3))
  return t

def predict_with_best_t(proba,best_threshold):
    prediction=[]
    global y_pred
    for val in proba:
        if val >= best_threshold:
            prediction.append(1)
        else:
            prediction.append(0)
    y_pred = prediction
    return prediction

# https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea

print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(train_threshold, train_fpr, train_tpr)
#print("\nTrain confusion matrix")
confusion_matrix_train=confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))

ax =plt.subplot()
sns.heatmap(confusion_matrix_train,xticklabels=['Predicted- No','Predcited - Yes'],yticklab
ax.set_title('Train confusion matrix')
ax.set_xlabel('Predicted class')
ax.set_ylabel('Actual class')
```

executed in 217ms, finished 16:08:03 2020-10-26

```
====================================================================================
========================
maximum value of tpr*(1-fpr) 0.4252494252314327 for threshold 0.478
```

Out[41]:

```
Text(33,0.5,'Actual class')
```



In [42]:

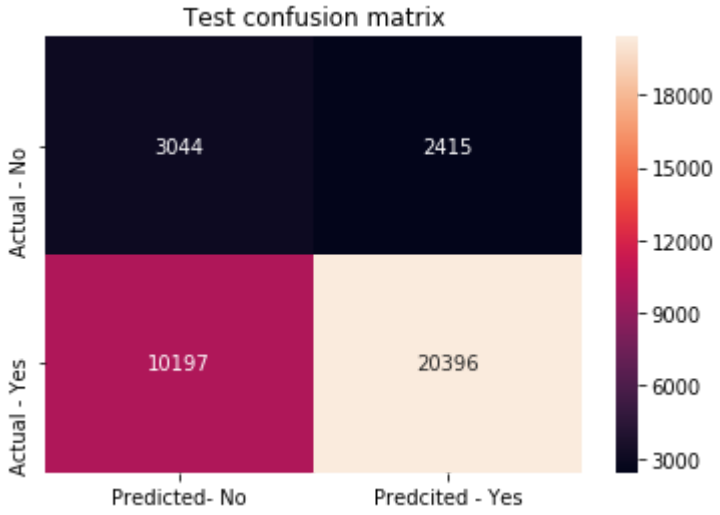
```
# for test


# https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea

print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(test_threshold, test_fpr, test_tpr)
print("\nTest confusion matrix")
confusion_matrix_train=confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))


ax =plt.subplot()
sns.heatmap(confusion_matrix_train,xticklabels=['Predicted- No','Predcited - Yes'],yticklab
ax.set_title('Test confusion matrix')
```

executed in 158ms, finished 16:08:03 2020-10-26

```
====================================================================================
========================
maximum value of tpr*(1-fpr) 0.37175300725231664 for threshold 0.458

Test confusion matrix
```

Out[42]:

```
Text(0.5,1,'Test confusion matrix')
```



In [43]:

```
print(len(y_test))
print(len(y_pred))
```

executed in 14ms, finished 16:08:03 2020-10-26

```
36052
36052
```

In [44]:

```
len(y_train)
```

executed in 14ms, finished 16:08:03 2020-10-26

Out[44]:

```
73196
```

In [45]:

```
# y_train
```

executed in 13ms, finished 16:08:03 2020-10-26

find the index of false positive datapoints of SET 1

In [46]:

```
# find the index in test data

index_fp = []

for i in range(len(y_pred)):
    if y_pred[i]==1 and y_test.values[i]==0:
        index_fp.append(i)

# after finding the index number we create the list of essay for each index

fp_essay =[]
for j in index_fp:
    fp_essay.append(X_test['essay'].values[j])
```

executed in 62ms, finished 16:08:03 2020-10-26

In [47]:

```
print(index_fp[:40])
```

executed in 14ms, finished 16:08:03 2020-10-26

```
[5, 12, 27, 30, 52, 75, 112, 119, 124, 159, 160, 180, 191, 192, 195, 208, 21
6, 270, 275, 284, 286, 325, 341, 372, 395, 396, 459, 465, 493, 525, 555, 55
6, 579, 586, 602, 645, 662, 676, 689, 693]
```

In [48]:

```
print(len(index_fp))
```

executed in 14ms, finished 16:08:03 2020-10-26

```
2415
```

In [50]:

```
print(len(fp_essay))
```

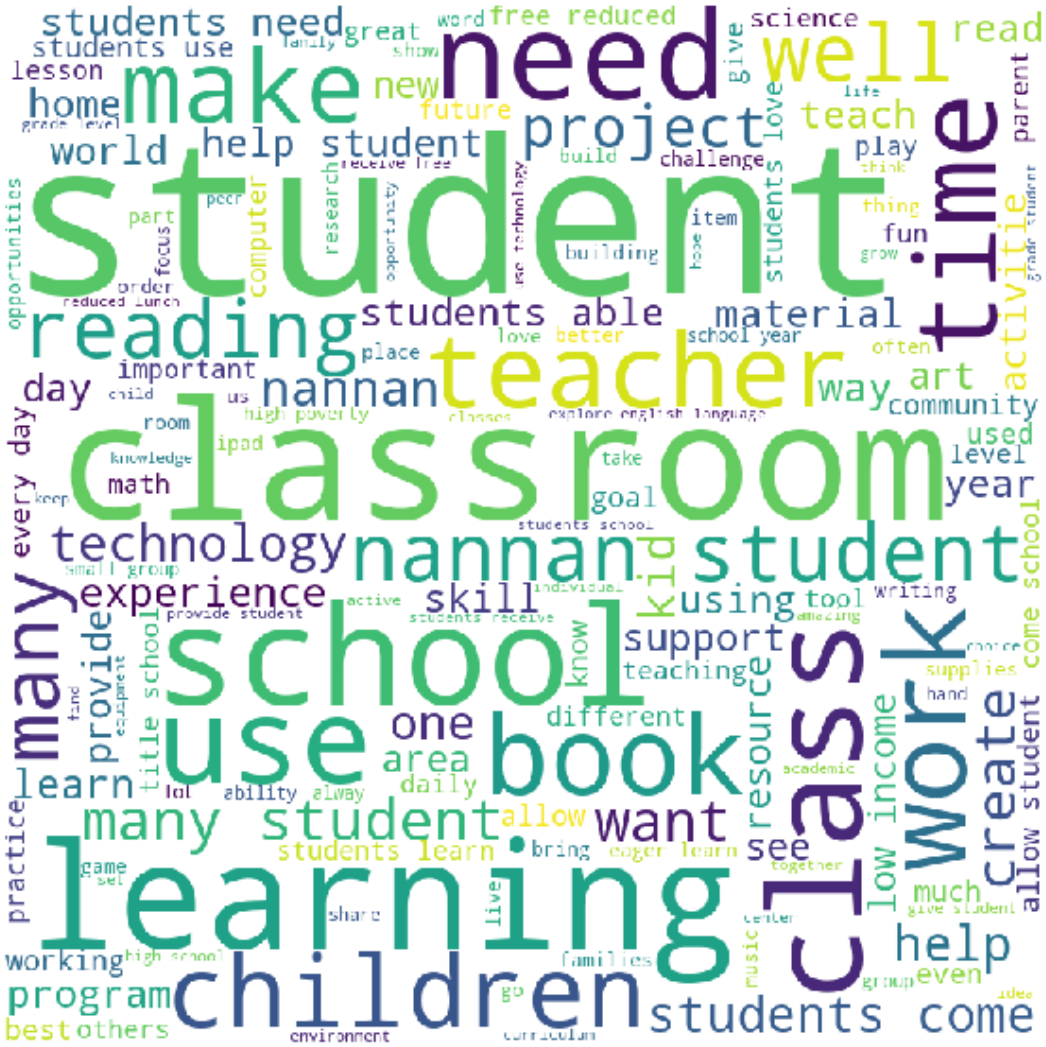
executed in 15ms, finished 16:08:03 2020-10-26

```
2415
```

In [51]:

```
# fp_essay contains the essay which model predicted as positive but it actual it was negati

# Python program to generate WordCloud

# importing all necessery modules
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

# Reads 'Youtube04-Eminem.csv' file

comment_words = ''
stopwords = set(STOPWORDS)

# iterate through the csv file
for val in fp_essay:

    # 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()

    comment_words += " ".join(tokens)+" "

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

executed in 4.34s, finished 16:08:08 2020-10-26



plot the box plot of price with these false positive datapoints

for this i will Creating the new dataframe that will contains the all values for all feature for false positive index datapoints in test data

In [52]:

```
# convert the series to dataframe and transpose it and then normally append it to new dataf
#  # https://stackoverflow.com/a/60684315

# example

X_test.iloc[1]
```

executed in 15ms, finished 16:08:08 2020-10-26

Out[52]:

```
school_state
tx
teacher_prefix
mrs
project_grade_category
grades_prek_2
teacher_number_of_previously_posted_projects
4
clean_categories                                                literacy_
language math_science
clean_subcategories                                             literatur
e_writing mathematics
essay                                       i amazing students they hard
working love lear...
price
59.49
neg
0.067
neu
0.585
pos
0.348
compound
0.9926
Name: 12126, dtype: object
```

In [53]:

```
#  # https://stackoverflow.com/a/60684315

# convert the series to dataframe and transpose it and then normally append it to new dataf

X_test.iloc[1].to_frame().T
```

executed in 15ms, finished 16:08:08 2020-10-26

Out[53]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_poste
12126	tx	mrs	grades_prek_2	

In [54]:

```
column_name = data.columns.values

# https://www.kite.com/python/answers/how-to-create-an-empty-dataframe-with-column-names-in
fp_data = pd.DataFrame(columns=column_name)

for i in index_fp:            # convert the series to dataframe and transpose it and then
    fp_data = fp_data.append(X_test.iloc[i].to_frame().T)          # https://stackoverflow.


fp_data.head()
```

executed in 4.81s, finished 16:08:13 2020-10-26

Out[54]:

	clean_categories	clean_subcategories	compound	essay	neg	neu	pos	
104759	math_science	appliedsciences mathematics	0.9913	my students amazing group children eager learn...	0.081	0.562	0.357	
64152	appliedlearning music_arts	charactereducation performingarts	0.9928	my students come different walks life bring di...	0.056	0.627	0.316	
103032	math_science literacy_language	environmentalscience literacy	0.9961	my name mrs w i teach 27 awesome kindergartner...	0.009	0.652	0.34	
54014	appliedlearning literacy_language	charactereducation literacy	0.9882	my first grade classroom filled diverse group ...	0.038	0.674	0.288	
32961	health_sports literacy_language	health_wellness literature_writing	0.9966	i class full energetic first grade sponges rea...	0.041	0.638	0.321	

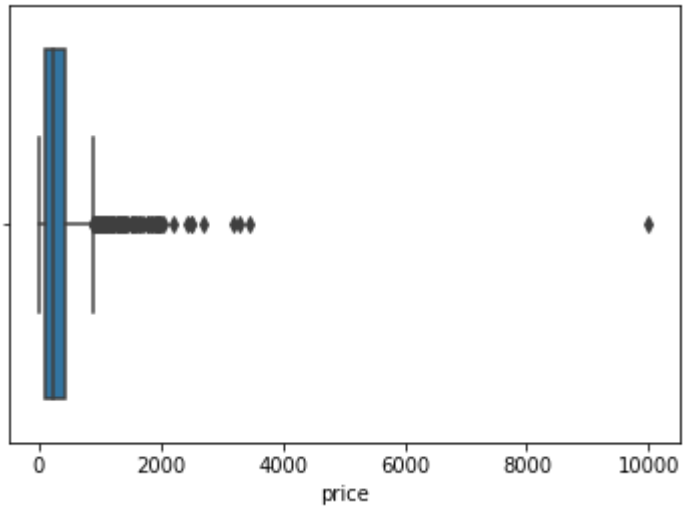
In [55]:

```
# plot the boxplot of price in fp_sn

#

fp_data['price'] = fp_data['price'].astype(float)
sns.boxplot(x='price',data=fp_data)
```

executed in 109ms, finished 16:08:13 2020-10-26

Out[55]:

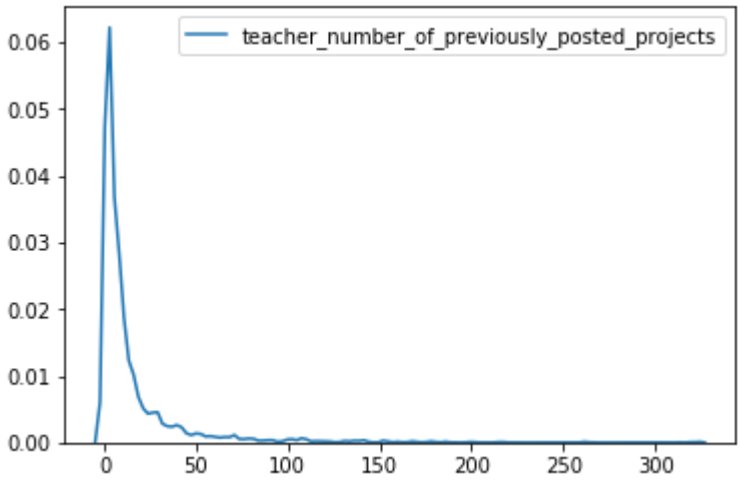
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bb3a9b55c0>
```



In [56]:

```
# plot the pdf for number of teacher previously posted for false negative datapoints

sns.kdeplot(fp_data['teacher_number_of_previously_posted_projects'])
plt.show()
```

executed in 140ms, finished 16:08:13 2020-10-26



set 2

hyperparameter tuninig for set 2

In [57]:

```
from pandas import DataFrame
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

# parameter of decision tree
depth = [1, 5, 10, 50]
min_samples_split = [5, 10, 100, 500]

# make a Decision tree classifier

DT = DecisionTreeClassifier(class_weight = 'balanced')

parameter = dict(max_depth = depth,min_samples_split= min_samples_split)

# make a grid search CV model

clf = GridSearchCV(DT,parameter,scoring='roc_auc',cv = 5,return_train_score= True)

# apply gridsearchCV on train dataset

clf.fit(X_tr_tfidfW2v,y_train)


results = pd.DataFrame.from_dict(clf.cv_results_)

results.head()
```

executed in 1h 14m 2s, finished 17:22:15 2020-10-26

Out[57]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_min
0	2.735879	0.040898	0.115904	0.005844	1	
1	2.707572	0.053575	0.113493	0.001930	1	
2	2.710752	0.044705	0.113280	0.000810	1	
3	2.691768	0.051024	0.116088	0.005074	1	
4	12.997447	0.089243	0.117488	0.004386	5	

5 rows × 22 columns

In [58]:

```
train_auc = results['mean_train_score']
cv_auc = results['mean_test_score']

max_depth_values = results['param_max_depth']
min_sample_split_values  = results['param_min_samples_split']

train_auc= list(train_auc)
cv_auc =list(cv_auc)
max_depth_values = list(max_depth_values)
min_sample_split_values = list(min_sample_split_values)
```

executed in 10ms, finished 17:22:15 2020-10-26

In [59]:

```
print(len(min_sample_split_values))
```

executed in 29ms, finished 17:22:15 2020-10-26

```
16
```

In [60]:

```
# best parameter values

clf.best_params_
```

executed in 13ms, finished 17:22:15 2020-10-26

Out[60]:

```
{'max_depth': 5, 'min_samples_split': 500}
```

plot the performance of model both on train data and cross validation data for each hyper parameter, using the heatmap with row as min_sample_split and column as max_depth and values inside the cell representing auc value

train data score

In [61]:

```
heap_data = {'max_depth':max_depth_values,'min_sample_split':min_sample_split_values,'trair

heap_dataframe = pd.DataFrame(heap_data)

heap_dataframe
```

executed in 29ms, finished 17:22:15 2020-10-26

Out[61]:

	max_depth	min_sample_split	train_score
0	1	5	0.550010
1	1	10	0.550010
2	1	100	0.550010
3	1	500	0.550010
4	5	5	0.656455
5	5	10	0.656455
6	5	100	0.656149
7	5	500	0.654887
8	10	5	0.802887
9	10	10	0.801853
10	10	100	0.778124
11	10	500	0.733364
12	50	5	0.999872
13	50	10	0.999027
14	50	100	0.905072
15	50	500	0.759899

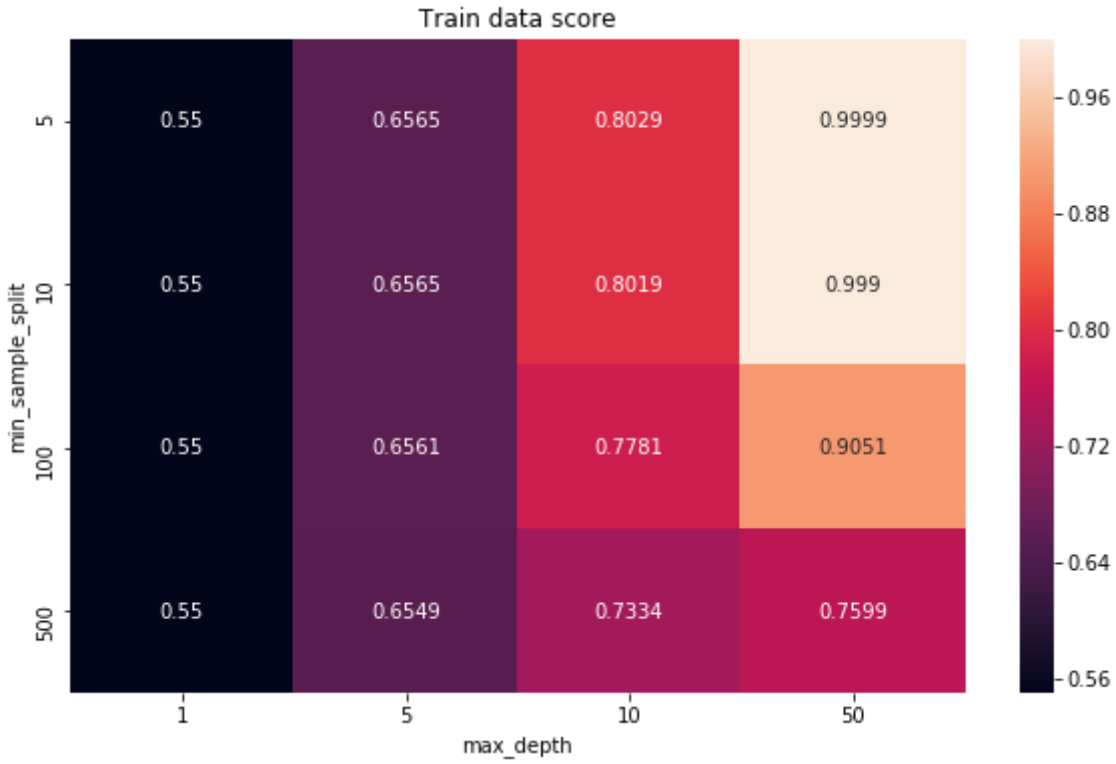
In [62]:

```
da_heatmap = heap_dataframe.pivot('min_sample_split','max_depth','train_score')
fig,ax= plt.subplots(figsize=(10,6))
ax = sns.heatmap(da_heatmap,annot=True,fmt='.4g')
ax.set_title('Train data score')
```

executed in 328ms, finished 17:22:16 2020-10-26

Out[62]:

```
Text(0.5,1,'Train data score')
```



test data score

In [63]:

```
heap_data = {'max_depth':max_depth_values,'min_sample_split':min_sample_split_values,'cv_sc

heap_dataframe = pd.DataFrame(heap_data)

heap_dataframe
```

executed in 13ms, finished 17:22:16 2020-10-26

Out[63]:

	max_depth	min_sample_split	cv_score
0	1	5	0.549470
1	1	10	0.549470
2	1	100	0.549470
3	1	500	0.549470
4	5	5	0.627237
5	5	10	0.627115
6	5	100	0.626966
7	5	500	0.627757
8	10	5	0.609247
9	10	10	0.608874
10	10	100	0.611851
11	10	500	0.624483
12	50	5	0.531557
13	50	10	0.533488
14	50	100	0.570934
15	50	500	0.616745

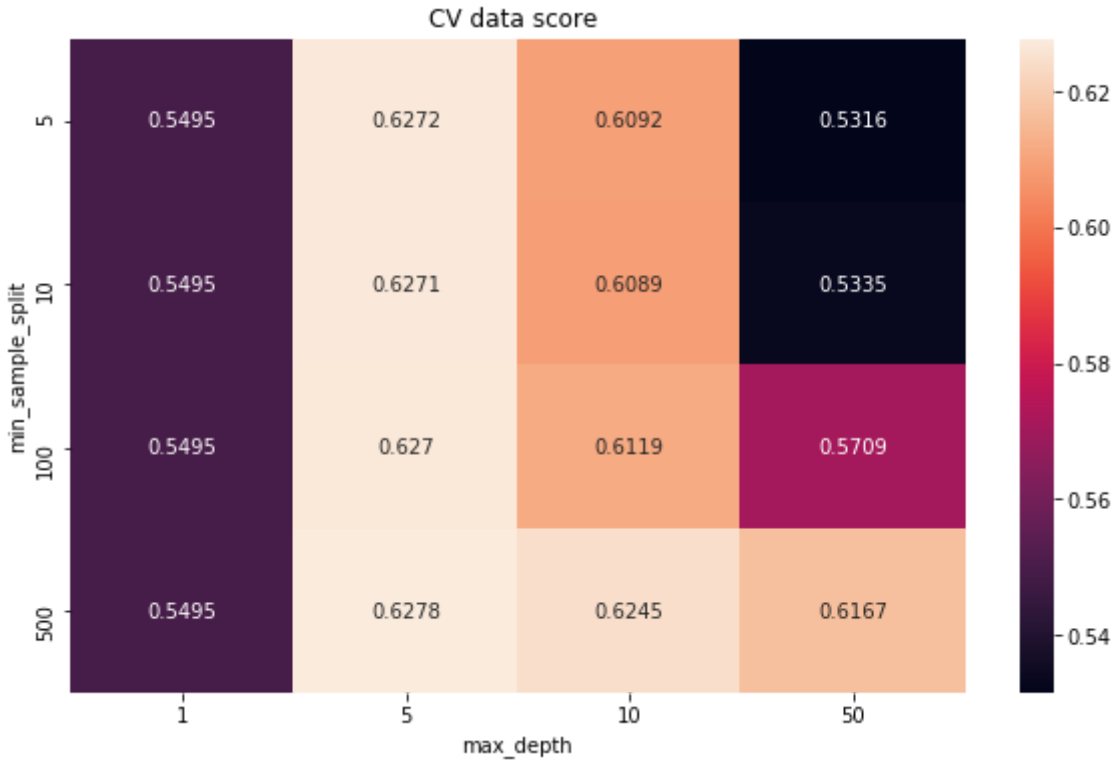
In [64]:

```
da_heatmap = heap_dataframe.pivot('min_sample_split','max_depth','cv_score')
fig,ax= plt.subplots(figsize=(10,6))
ax = sns.heatmap(da_heatmap,annot= True,fmt='.4g')
ax.set_title('CV data score ')
```

executed in 284ms, finished 17:22:16 2020-10-26

Out[64]:

```
Text(0.5,1,'CV data score ')
```



now using the best values parameter that we got above from clf.best_params_ train the model with these values

ROC plot for SET 2

In [65]:

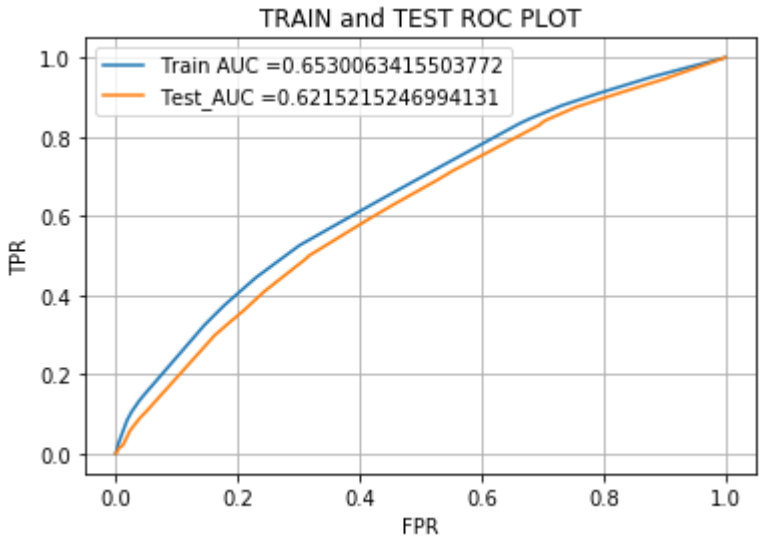
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 4900
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

executed in 12ms, finished 17:22:16 2020-10-26

In [66]:

```
y_train_pred=[]  # they store the probability estimate of predicted class
y_test_pred=[]

# apply decision tree with best parameter values

dtree = DecisionTreeClassifier(max_depth = 5 ,min_samples_split =5 ,class_weight='balanced'
dtree.fit(X_tr_tfidfW2v,y_train)

# find predicted probability

y_train_pred = batch_predict(dtree,X_tr_tfidfW2v)
y_test_pred = batch_predict(dtree,X_te_tfidfW2v)


# find the fpr , tpr for each model considering each probability estimates as threshold

train_fpr,train_tpr,train_threshold = roc_curve(y_train,y_train_pred)
test_fpr,test_tpr,test_threshold =roc_curve(y_test,y_test_pred)

# plot the roc curve for both train and test

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('TRAIN and TEST ROC PLOT')
plt.grid()
```

executed in 24.6s, finished 17:22:41 2020-10-26



confusion matrix

In [67]:

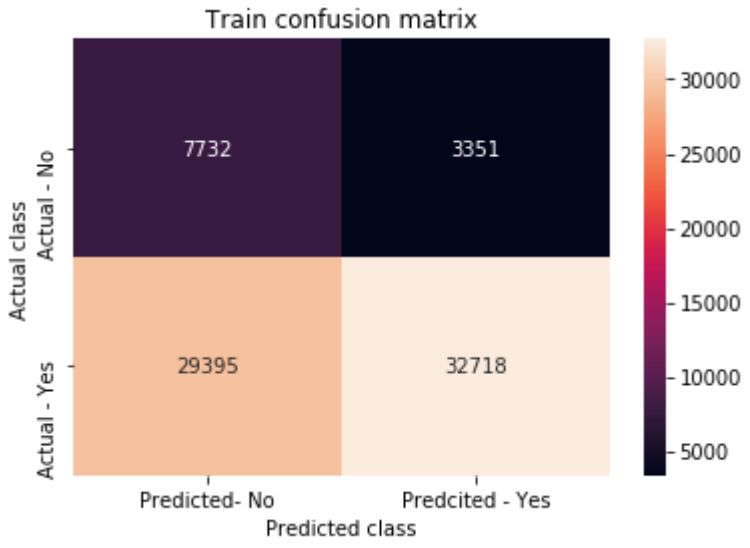
```
def find_best_threshold(threshold,fpr,tpr):
  t= threshold[np.argmax(tpr*(1-fpr))]
  print('maximum value of tpr*(1-fpr)',max(tpr*(1-fpr)),'for threshold',np.round(t,3))
  return t

def predict_with_best_t(proba,best_threshold):
    prediction=[]
    global y_pred_2
    for val in proba:
        if val >= best_threshold:
            prediction.append(1)
        else:
            prediction.append(0)
    y_pred_2 = prediction
    return prediction

# https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea

print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(train_threshold, train_fpr, train_tpr)
#print("\nTrain confusion matrix")
confusion_matrix_train=confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))

ax =plt.subplot()
sns.heatmap(confusion_matrix_train,xticklabels=['Predicted- No','Predcited - Yes'],yticklab
ax.set_title('Train confusion matrix')
ax.set_xlabel('Predicted class')
ax.set_ylabel('Actual class')
```

executed in 251ms, finished 17:22:41 2020-10-26

```
====================================================================================
========================
maximum value of tpr*(1-fpr) 0.3674842703253954 for threshold 0.504
```

Out[67]:

```
Text(33,0.5,'Actual class')
```



In [68]:

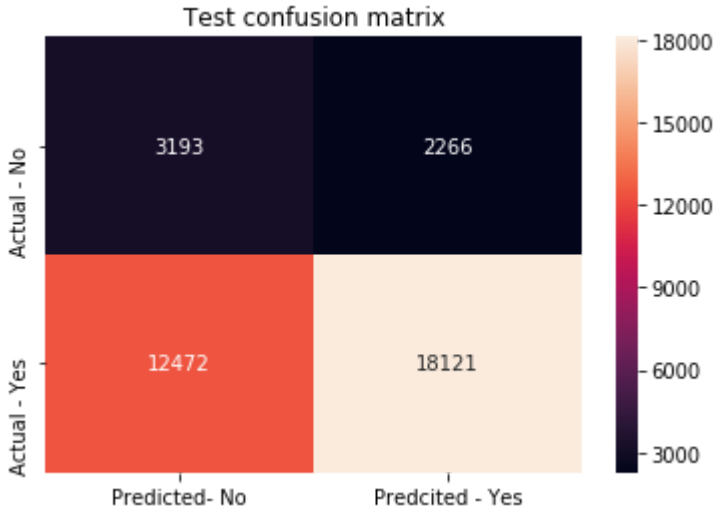
```
# for test


# https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea

print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(test_threshold, test_fpr, test_tpr)
print("\nTest confusion matrix")
confusion_matrix_train=confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))


ax =plt.subplot()
sns.heatmap(confusion_matrix_train,xticklabels=['Predicted- No','Predcited - Yes'],yticklab
ax.set_title('Test confusion matrix')
```

executed in 203ms, finished 17:22:41 2020-10-26

```
====================================================================================
========================
maximum value of tpr*(1-fpr) 0.3464542696596644 for threshold 0.465

Test confusion matrix
```

Out[68]:

```
Text(0.5,1,'Test confusion matrix')
```



In [69]:

```
# find the false positive datapoint from test

print(len(y_test))
print(len(y_pred_2))
```

executed in 14ms, finished 17:22:41 2020-10-26

```
36052
36052
```

In [70]:

```
# store the index position of false positive datapoints in test data

fp_index = []
for i in range(len(y_test)):
    if ((y_pred[i]==1) and (y_test.values[i]==0)):
        fp_index.append(i)

fp_index[:40]
```

executed in 45ms, finished 17:22:41 2020-10-26

Out[70]:

```
[5,
 12,
 27,
 30,
 52,
 75,
 112,
 119,
 124,
 159,
 160,
 180,
 191,
 192,
 195,
 208,
 216,
 270,
 275,
 284,
 286,
 325,
 341,
 372,
 395,
 396,
 459,
 465,
 493,
 525,
 555,
 556,
 579,
 586,
 602,
 645,
 662,
 676,
 689,
 693]
```

In [71]:

```
print(len(fp_index))
```

executed in 14ms, finished 17:22:41 2020-10-26

```
2415
```

In [72]:

```
# find the essay for above index only

fp_essay=[]
for i in fp_index:
    fp_essay.append(X_test['essay'].values[i])

# fp_essay
```

executed in 62ms, finished 17:22:41 2020-10-26

In [73]:

```
print(len(fp_essay))
```

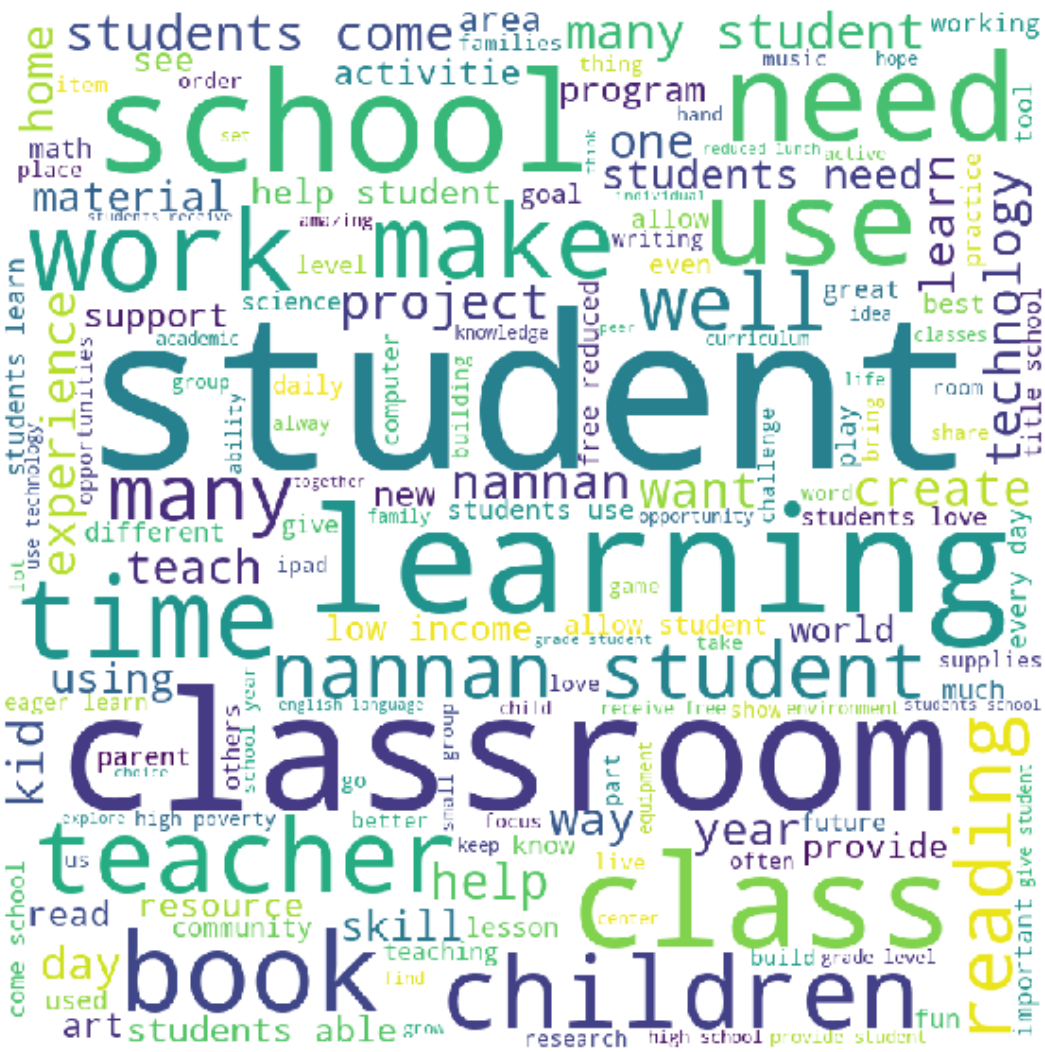
executed in 13ms, finished 17:22:41 2020-10-26

```
2415
```

In [74]:

```
# plot the wordcloud for above essay
# Python program to generate WordCloud

# importing all necessery modules
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

# Reads 'Youtube04-Eminem.csv' file

comment_words = ''
stopwords = set(STOPWORDS)

# iterate through the csv file
for val in fp_essay:

    # 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()

    comment_words += " ".join(tokens)+" "

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

executed in 5.58s, finished 17:22:47 2020-10-26



plot the box plot of price with these false negative datapoints

for this i will Creating the new dataframe that will contains the all values for all feature for negative index datapoints in test data

In [75]:

```
column_name = data.columns.values

# https://www.kite.com/python/answers/how-to-create-an-empty-dataframe-with-column-names-in
fp_data = pd.DataFrame(columns=column_name)

for i in fp_index:            # convert the series to dataframe and transpose it and then
    fp_data = fp_data.append(X_test.iloc[i].to_frame().T)          # https://stackoverflow.


fp_data.head()
```

executed in 6.70s, finished 17:22:54 2020-10-26

Out[75]:

	clean_categories	clean_subcategories	compound	essay	neg	neu	pos	
104759	math_science	appliedsciences mathematics	0.9913	my students amazing group children eager learn...	0.081	0.562	0.357	
64152	appliedlearning music_arts	charactereducation performingarts	0.9928	my students come different walks life bring di...	0.056	0.627	0.316	
103032	math_science literacy_language	environmentalscience literacy	0.9961	my name mrs w i teach 27 awesome kindergartner...	0.009	0.652	0.34	
54014	appliedlearning literacy_language	charactereducation literacy	0.9882	my first grade classroom filled diverse group ...	0.038	0.674	0.288	
32961	health_sports literacy_language	health_wellness literature_writing	0.9966	i class full energetic first grade sponges rea...	0.041	0.638	0.321	

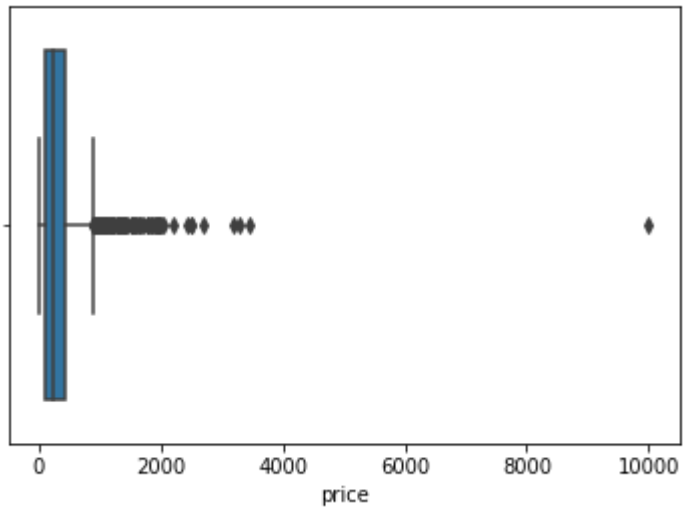
In [76]:

```
# plot the boxplot of price in fp_sn

fp_data['price'] = fp_data['price'].astype(float)
sns.boxplot(x='price',data=fp_data)
```

executed in 125ms, finished 17:22:54 2020-10-26

Out[76]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bb3f70e358>
```

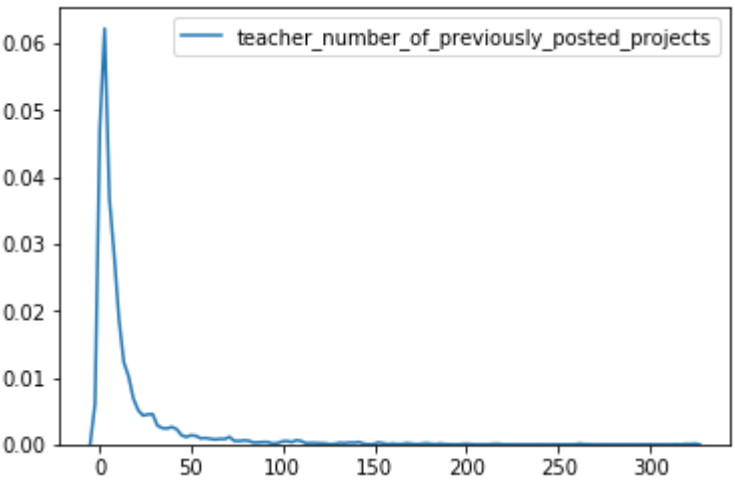


50% of prices ranges between 0 to 300

In [77]:

```
# plot the pdf for number of teacher previously posted for false negative datapoints

sns.kdeplot(fp_data['teacher_number_of_previously_posted_projects'])
plt.show()
```

executed in 141ms, finished 17:22:54 2020-10-26



most of the teacher posted 0 project previously

In [ ]: