In [3]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
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
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import pickle
from tqdm import tqdm
import os
from chart_studio import plotly # use chart_studio instead of plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

1. Decision Tree

1.1 loading data

In [5]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv',nrows=50000)
data.head(5)
```

Out[5]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clea
0	ca	mrs	grades_prek_2	53	1	math_science	l
1	ut	ms	grades_3_5	4	1	specialneeds	
2	ca	mrs	grades_prek_2	10	1	literacy_language	
3	ga	mrs	grades_prek_2	2	1	appliedlearning	
4	wa	mrs	grades_3_5	2	1	literacy_language	

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

In [ ]:
```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
```

In [8]:
```
# train test split
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)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

set 1: TFIDF

1.3 Make Data Model Ready: encoding numerical, categorical features

1.3.1 encoding categorical features: essays

In [119]:
```
# Tfidf vectorizer transform text into feature vector.

## initialize model with different parameters
vectorizer_essay_set2 = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=10000)

# fitting the data into model
X_train_essay_tfidf=vectorizer_essay_set2.fit_transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer_essay_set2.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer_essay_set2.transform(X_test['essay'].values)

print("After using tfidf vectorization: ")
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)
```

```
After using tfidf vectorization:
(22445, 10000) (22445,)
(11055, 10000) (11055,)
(16500, 10000) (16500,)
====================================================================================================
```

1.3.2 encoding categorical features: School State

In [28]:
```
# Tfidf vectorizer transform text into feature vector.

## initialize model
vectorizer_school_state = TfidfVectorizer()

# fitting the data into model
X_train_school_state=vectorizer_school_state.fit_transform(X_train['school_state'].values)
X_cv_school_state = vectorizer_school_state.transform(X_cv['school_state'].values)
X_test_school_state = vectorizer_school_state.transform(X_test['school_state'].values)

print("After using tfidf vectorization: ")
print(X_train_school_state.shape, y_train.shape)
print(X_cv_school_state.shape, y_cv.shape)
print(X_test_school_state.shape, y_test.shape)
print(vectorizer_school_state.get_feature_names())
print("="*100)
```

```
After using tfidf vectorization:
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'm
a', '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']
====================================================================================================
```

1.3.3 encoding categorical features: teacher_prefix

In [29]:
```
#create object of the model
vectorizer_teacher_prefix = TfidfVectorizer()

# fitting the data into model
X_train_teacher_prefix = vectorizer_teacher_prefix.fit_transform(X_train['teacher_prefix'].values)
X_cv_teacher_prefix    = vectorizer_teacher_prefix.transform(X_cv['teacher_prefix'].values)
X_test_teacher_prefix  = vectorizer_teacher_prefix.transform(X_test['teacher_prefix'].values)

print("After using tfidf vectorization: ")
print(X_train_teacher_prefix.shape, y_train.shape)
print(X_cv_teacher_prefix.shape, y_cv.shape)
print(X_test_teacher_prefix.shape, y_test.shape)
print(vectorizer_teacher_prefix.get_feature_names())
print("="*100)
```

```
After using tfidf vectorization:
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

1.3.4 encoding categorical features: project_grade_category

In [27]:
```
vectorizer_project_grade_category = TfidfVectorizer()

# fitting the data into model
X_train_project_grade_category = vectorizer_project_grade_category.fit_transform(X_train['project_grade_category'].val
ues)
X_cv_project_grade_category    = vectorizer_project_grade_category.transform(X_cv['project_grade_category'].values)
X_test_project_grade_category  = vectorizer_project_grade_category.transform(X_test['project_grade_category'].values)

print("After using tfidf vectorization: ")
print(X_train_project_grade_category.shape, y_train.shape)
print(X_cv_project_grade_category.shape, y_cv.shape)
print(X_test_project_grade_category.shape, y_test.shape)
print(vectorizer_project_grade_category.get_feature_names())
print("="*100)
```

```
After using tfidf vectorization:
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================================
```

1.3.5 encoding categorical features: clean_category

In [26]:
```
vectorizer_clean_category = TfidfVectorizer()

# fitting the data into model
X_train_clean_category = vectorizer_clean_category.fit_transform(X_train['clean_categories'].values)
X_cv_clean_category    = vectorizer_clean_category.transform(X_cv['clean_categories'].values)
X_test_clean_category  = vectorizer_clean_category.transform(X_test['clean_categories'].values)

print("After using tfidf vectorization: ")
print(X_train_clean_category.shape, y_train.shape)
print(X_cv_clean_category.shape, y_cv.shape)
print(X_test_clean_category.shape, y_test.shape)
print(vectorizer_clean_category.get_feature_names())
print("="*100)
```

```
After using tfidf vectorization:
(22445, 7) (22445,)
(11055, 7) (11055,)
(16500, 7) (16500,)
['appliedlearning', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_arts', 'specialnee
ds']
====================================================================================================
```

1.3.6 encoding categorical features: clean_subcategorie

In [25]:
```
vectorizer_clean_subcategories = TfidfVectorizer()

# fitting the data into model
X_train_clean_subcategories = vectorizer_clean_subcategories.fit_transform(X_train['clean_subcategories'].values)
X_cv_clean_subcategories    = vectorizer_clean_subcategories.transform(X_cv['clean_subcategories'].values)
X_test_clean_subcategories  = vectorizer_clean_subcategories.transform(X_test['clean_subcategories'].values)

print("After using tfidf vectorization: ")
print(X_train_clean_subcategories.shape, y_train.shape)
print(X_cv_clean_subcategories.shape, y_cv.shape)
print(X_test_clean_subcategories.shape, y_test.shape)
print(vectorizer_clean_subcategories.get_feature_names())
print("="*100)
```

```
After using tfidf vectorization:
(22445, 28) (22445,)
(11055, 28) (11055,)
(16500, 28) (16500,)
['appliedsciences', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice', 'earlydevelo
pment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_
fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'mathematic
s', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialsciences', 'specialneeds',
'teamsports', 'visualarts']
====================================================================================================
```

1.3.7 encoding numerical features: price

In [23]:
```
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['price'].values.reshape(1,-1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1)).reshape(-1,1)
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1)).reshape(-1,1)
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1)).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
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

1.3.8 encoding numerical features: teacher_number_of_previously_posted_project

In [24]:
```
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

X_train_TNPP_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
.reshape(-1,1)
X_cv_TNPP_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)).resha
pe(-1,1)
X_test_TNPP_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)).r
eshape(-1,1)

print("After vectorizations")
print(X_train_TNPP_norm.shape, y_train.shape)
print(X_cv_TNPP_norm.shape, y_cv.shape)
print(X_test_TNPP_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
```

Concatenation of features of set_1

In [31]:
```
from scipy.sparse import hstack #import library for concatenate features.

##here we cancatenating all features of training, test and cross validation
X_tr_set1 = hstack((X_train_essay_tfidf,X_train_school_state,X_train_teacher_prefix,X_train_project_grade_category,X_train_clean_category,X_train_clean_subcategories,X_train_price_norm,X_train_TNPP_norm)).tocsr()
X_cr_set1 = hstack((X_cv_essay_tfidf,X_cv_school_state,X_cv_teacher_prefix,X_cv_project_grade_category,X_cv_clean_category,X_cv_clean_subcategories,X_cv_price_norm,X_cv_TNPP_norm)).tocsr()
X_te_set1 = hstack((X_test_essay_tfidf,X_test_school_state,X_test_teacher_prefix,X_test_project_grade_category,X_test_clean_category,X_test_clean_subcategories,X_test_price_norm,X_test_TNPP_norm)).tocsr()

print("Final Data matrix for set 1: ")
print(X_tr_set1.shape, y_train.shape)
print(X_cr_set1.shape, y_cv.shape)
print(X_te_set1.shape, y_test.shape)
print("="*100)
print("="*100)
```

```
Final Data matrix for set 1:
(22445, 10096) (22445,)
(11055, 10096) (11055,)
(16500, 10096) (16500,)
====================================================================================================
====================================================================================================
```

set_2 : using TFIDF W2V

encoding categorical features: essays

In [32]:

```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

# ============================
Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

# ============================

words = []
for i in preproced_texts:
    words.extend(i.split(' '))

for i in preproced_titles:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and our coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
words_glove = set(model.keys())
for i in words:
```

```
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variable
s-in-python/

import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_courpus, f)


'''
```

Out[32]:
```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef loadGloveModel(gloveFile):\n
print ("Loading Glove Model")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    model = {}\n    for line in tqdm
(f):\n        splitLine = line.split()\n        word = splitLine[0]\n        embedding = np.array([float(val) for val
in splitLine[1:]])\n        model[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return mode
l\nmodel = loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading Glove Mod
el\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n# ============================\n\nwords = []\nfor
i in preproced_texts:\n    words.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\'
\'))\nprint("all the words in the coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus", l
en(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words that are present in bot
h glove vectors and our coupus",       len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nword
s_courpus = {}\nwords_glove = set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i]
= model[i]\nprint("word 2 vec length", len(words_courpus))\n\n\n\n# stronging variables into pickle files python: htt
p://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pickle\nwith open(\'glove_v
ectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [33]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variable
s-in-python/
# 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 [35]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'])
# 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 [36]:
```
 #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']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.
split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each
 word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_essay_train.append(vector)

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

```
100%|██████████████████████████████████████████████████████████████████████████| 22445/22445 [01:25<00:00, 262.26it/
s]

22445
300
```

In [37]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_cv['essay'])
# 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_)))
#x=list(tfidf_model.idf_)
tfidf_words = set(tfidf_model.get_feature_names())
```

In [38]:
```
 #average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_essay_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.
split())))

            # here we are using idf value of train data###
            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_cv.append(vector)

print(len(tfidf_w2v_essay_cv))
print(len(tfidf_w2v_essay_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 11055/11055 [00:13<00:00, 810.38it/
s]

11055
300
```

In [39]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test['essay'])
# 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_)))
#x=list(tfidf_model.idf_)
tfidf_words = set(tfidf_model.get_feature_names())
```

In [40]:
```
 #average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_essay_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.
split())))

            # here we are using idf value of train data###
            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%|████████████████████████████████████████████████████████████████████████████| 16500/16500 [00:19<00:00, 856.11it/
s]

16500
300
```

concatenation of features of set_2

In [41]:
```
from scipy.sparse import hstack #import library for concatenate features.

##here we cancatenating all features of training, test and cross validation
X_tr_set2 = hstack((tfidf_w2v_essay_train,X_train_school_state,X_train_teacher_prefix,X_train_project_grade_category,X
_train_clean_category,X_train_clean_subcategories,X_train_price_norm,X_train_TNPP_norm)).tocsr()
X_cr_set2 = hstack((tfidf_w2v_essay_cv,X_cv_school_state,X_cv_teacher_prefix,X_cv_project_grade_category,X_cv_clean_ca
tegory,X_cv_clean_subcategories,X_cv_price_norm,X_cv_TNPP_norm)).tocsr()
X_te_set2 = hstack((tfidf_w2v_essay_test,X_test_school_state,X_test_teacher_prefix,X_test_project_grade_category,X_tes
t_clean_category,X_test_clean_subcategories,X_test_price_norm,X_test_TNPP_norm)).tocsr()

print("Final Data matrix for set 1: ")
print(X_tr_set2.shape, y_train.shape)
print(X_cr_set2.shape, y_cv.shape)
print(X_te_set2.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix for set 1:
(22445, 396) (22445,)
(11055, 396) (11055,)
(16500, 396) (16500,)
====================================================================================================
```

Apply Decision tree classifier on set_1

In [42]:
```
## import Decision tree classiffier
from sklearn.tree import DecisionTreeClassifier

## importing confusion matrix
from sklearn.metrics import confusion_matrix
```

In [43]:
```
# creating model
DTC_model = DecisionTreeClassifier()

## fitting the data into model
DTC_model.fit(X_tr_set1,y_train)

## here i am doing prediction
set1_prediction= DTC_model.predict(X_te_set1)
```

In [44]:
```
### find confusion matrix ###

confusion_matrix(set1_prediction, y_test)
```

Out[44]:
```
array([[  632,  2155],
       [ 2010, 11703]], dtype=int64)
```

Apply Decision tree classifier on set_2

In [47]:
```
# creating model
DTC_model = DecisionTreeClassifier()

## fitting the data into model
DTC_model.fit(X_tr_set2,y_train)

## here i am doing prediction
set2_prediction= DTC_model.predict(X_te_set2)
```

In [48]:
```
### find confusion matrix ###

confusion_matrix(set2_prediction, y_test)
```

Out[48]:
```
array([[  528,  1621],
       [ 2114, 12237]], dtype=int64)
```

2. The hyper paramter tuning(find best alpha:smoothing parameter)

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

2.1.1 using TFIDF

In [59]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # 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 = 49000
    # 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
```

In [208]:

```
import math
import matplotlib.pyplot as plt
#from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or non-thresholded measur
e of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
depth = [1, 5, 10, 50]
min_samples_split = [5, 10, 100, 500]
for i in tqdm(depth):
    for j in tqdm(min_samples_split):
        DTC_model = DecisionTreeClassifier(max_depth=i, min_samples_split= j) #checking ,model with all values of alph
a
        DTC_model.fit(X_tr_set1, y_train)

        y_train_pred = batch_predict(DTC_model, X_tr_set1)
        y_cv_pred = batch_predict(DTC_model, X_cr_set1)

        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
  0%|                                                                                    | 0/4 [00:00<?, ?it/
s]

  0%|                                                                                    | 0/4 [00:00<?, ?it/
s]

 25%|█████████████████████                                                               | 1/4 [00:02<00:08,  2.68s/i
t]

 50%|██████████████████████████████████████████                                          | 2/4 [00:03<00:04,  2.17s/i
t]

 75%|███████████████████████████████████████████████████████████████                     | 3/4 [00:04<00:01,  1.82s/i
t]
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 4/4 [00:05<00:00,  1.43s/i
t]


 25%|████████████████▌                                                               | 1/4 [00:05<00:17,  5.73s/i
t]


  0%|                                                                                | 0/4 [00:00<?, ?it/
s]


 25%|████████████████▌                                                               | 1/4 [00:03<00:10,  3.35s/i
t]


 50%|█████████████████████████████████                                               | 2/4 [00:06<00:06,  3.33s/i
t]
```

```
 75%|██████████████████████████████████████████████████████████████                     | 3/4 [00:09<00:03,  3.31s/i
t]
```

```
100%|███████████████████████████████████████████████████████████████████████████████████| 4/4 [00:13<00:00,  3.27s/i
t]
```

```
 50%|█████████████████████████████████████████▌                                         | 2/4 [00:18<00:15,  7.94s/i
t]
```

```
  0%|                                                                                           | 0/4 [00:00<?, ?it/
s]
```

```
 25%|████████████████████▊                                                              | 1/4 [00:07<00:21,  7.18s/i
t]
```

```
 50%|████████████████████████████████████                                    | 2/4 [00:14<00:14,  7.10s/i
t]

 75%|████████████████████████████████████████████████████                    | 3/4 [00:20<00:06,  6.94s/i
t]

100%|████████████████████████████████████████████████████████████████████████| 4/4 [00:26<00:00,  6.70s/i
t]

 75%|████████████████████████████████████████████████████                    | 3/4 [00:45<00:13, 13.60s/i
t]

  0%|                                                                          | 0/4 [00:00<?, ?it/
s]

 25%|██████████████████                                                      | 1/4 [00:38<01:55, 38.40s/i
t]
```

```
 50%|████████████████████████████████████                                        | 2/4 [01:13<01:15, 37.53s/i
t]

 75%|████████████████████████████████████████████████████                        | 3/4 [01:48<00:36, 36.78s/i
t]

100%|████████████████████████████████████████████████████████████████████████████| 4/4 [02:21<00:00, 35.30s/i
t]

100%|████████████████████████████████████████████████████████████████████████████| 4/4 [03:06<00:00, 46.74s/i
t]
```

In [58]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [209]:

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

trace1= go.Scatter3d(x=depth, y=min_samples_split, z=train_auc, name='train' )
trace2= go.Scatter3d(x=depth, y=min_samples_split, z=cv_auc , name='cross validation' )
data = [trace1, trace2]

layout = go.Layout(scene=dict(
                            xaxis = dict(title='min_sample_split'),
                            yaxis = dict(title='max_depth'),
                            zaxis = dict(title='AUC'),
                            ))
fig = go.Figure(data = data, layout = layout)
offline.iplot(fig, filename = "3d-scatter-colorsale")
```

2.1.2: using TFIDF W2V

In [210]:

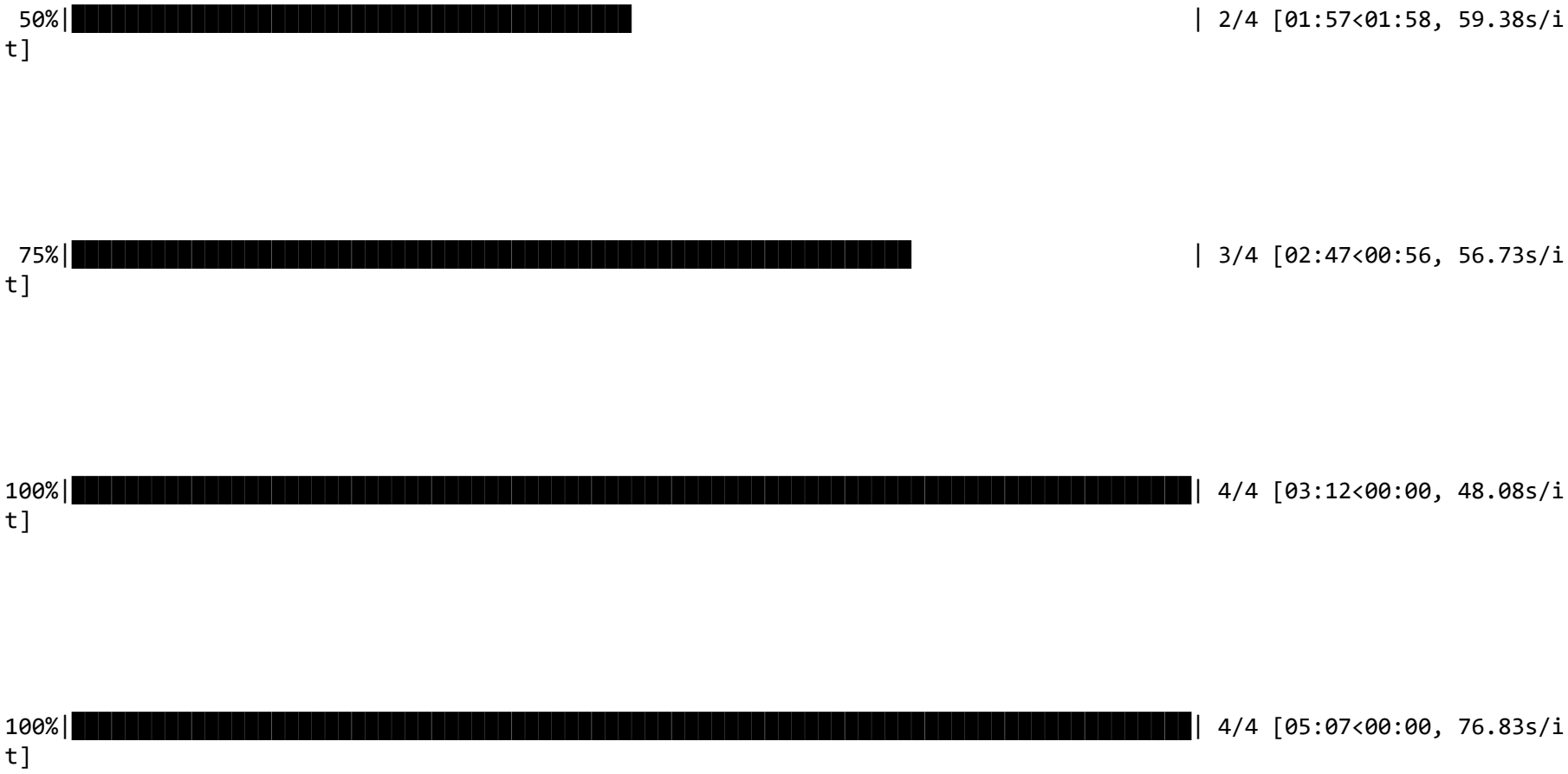
```
import math
import matplotlib.pyplot as plt
#from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or non-thresholded measur
e of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
depth = [1, 5, 10, 50]
min_samples_split = [5, 10, 100, 500]
for i in tqdm(depth):
    for j in tqdm(min_samples_split):
        DTC_model = DecisionTreeClassifier(max_depth=i, min_samples_split= j) #checking ,model with all values of alph
a
        DTC_model.fit(X_tr_set2, y_train)

        y_train_pred = batch_predict(DTC_model, X_tr_set2)
        y_cv_pred = batch_predict(DTC_model, X_cr_set2)

        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

```
  0%|                                                  | 0/4 [00:00<?, ?it/
s]

  0%|                                                  | 0/4 [00:00<?, ?it/
s]

 25%|                                          | 1/4 [00:04<00:13,  4.39s/i
t]

 50%|                                          | 2/4 [00:06<00:07,  3.75s/i
t]

 75%|                                          | 3/4 [00:08<00:03,  3.32s/i
t]
```

```
100%|██████████████████████████████████████████████████████████████████████████| 4/4 [00:11<00:00,  2.80s/i
t]


 25%|████████████████▌                                                          | 1/4 [00:11<00:33, 11.20s/i
t]


  0%|                                                                           | 0/4 [00:00<?, ?it/
s]


 25%|████████████████▌                                                          | 1/4 [00:07<00:23,  7.71s/i
t]


 50%|█████████████████████████████████                                          | 2/4 [00:15<00:15,  7.61s/i
t]
```

```
 75%|                                                              | 3/4 [00:22<00:07,  7.57s/i
t]

100%|                                                              | 4/4 [00:29<00:00,  7.47s/i
t]

 50%|                                                              | 2/4 [00:41<00:33, 16.81s/i
t]

  0%|                                                                      | 0/4 [00:00<?, ?it/
s]

 25%|                                                              | 1/4 [00:20<01:00, 20.08s/i
t]
```

```
 50%|████████████████████████████████████                                    | 2/4 [00:40<00:40, 20.04s/i
t]


 75%|████████████████████████████████████████████████████                    | 3/4 [00:59<00:19, 19.80s/i
t]


100%|████████████████████████████████████████████████████████████████████████| 4/4 [01:13<00:00, 18.47s/i
t]


 75%|████████████████████████████████████████████████████                    | 3/4 [01:54<00:33, 33.93s/i
t]


  0%|                                                                          | 0/4 [00:00<?, ?it/
s]


 25%|██████████████████                                                      | 1/4 [01:00<03:01, 60.54s/i
t]
```


50%| | 2/4 [01:57<01:58, 59.38s/it]
75%| | 3/4 [02:47<00:56, 56.73s/it]
100%| | 4/4 [03:12<00:00, 48.08s/it]
100%| | 4/4 [05:07<00:00, 76.83s/it]

In [211]:

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

trace1= go.Scatter3d(x=depth, y=min_samples_split, z=train_auc, name='train' )
trace2= go.Scatter3d(x=depth, y=min_samples_split, z=cv_auc , name='cross validation' )
data = [trace1, trace2]

layout = go.Layout(scene=dict(
                            xaxis = dict(title='min_sample_split'),
                            yaxis = dict(title='max_depth'),
                            zaxis = dict(title='AUC'),
                            ))
fig = go.Figure(data = data, layout = layout)
offline.iplot(fig, filename = "3d-scatter-colorsale")
```

observation: with the help of above curves the best parameter are depth = 50 and min_sample_split = 500

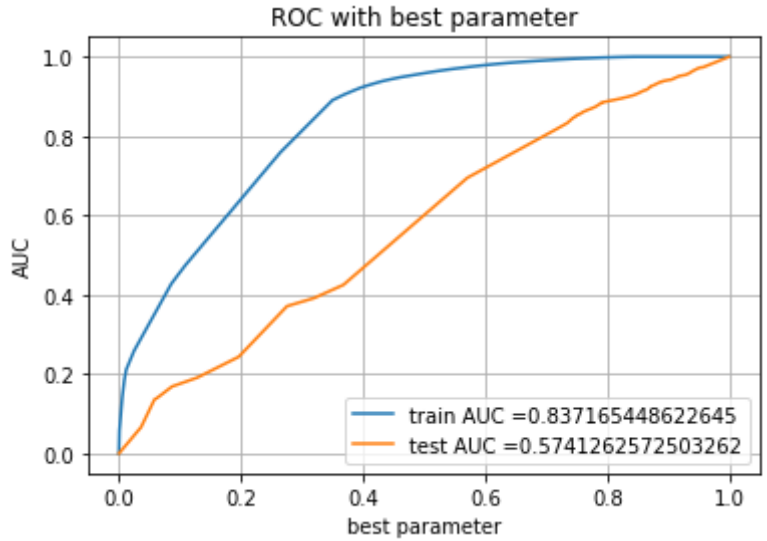
3. repersentation of results

3.1 plot roc curve with best smoothing parameters:

In [72]:
```
## function for calculating the ROC curve ###

def ROC_curve_with_best_parameter(y_train_pred,y_test_pred):

    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("best parameter")
    plt.ylabel("AUC")
    plt.title("ROC with best parameter")
    plt.grid()
    plt.show()
```

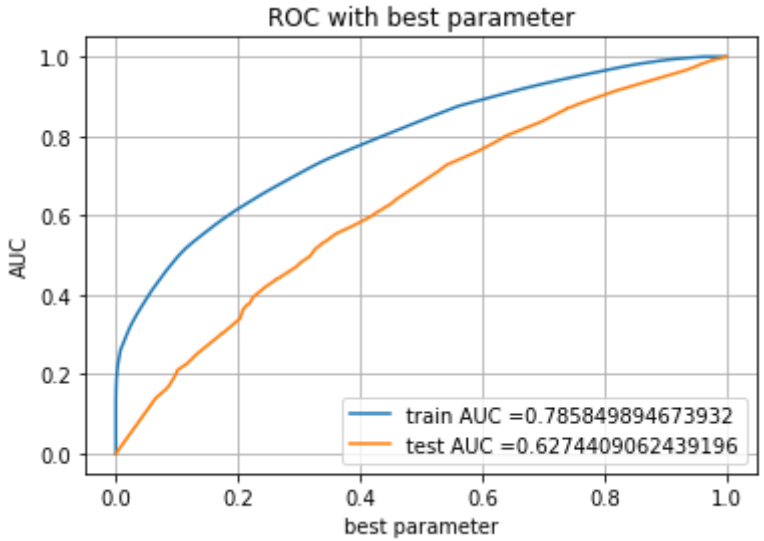
3.1.1 using TFIDF

In [100]:
```
## here i am initializing the model and fit the best value of alpha ###
DTC_model = DecisionTreeClassifier(max_depth=50, min_samples_split= 500)

## fit values in the model ##
DTC_model.fit(X_tr_set1, y_train)

## doing train prediction
y_train_pred = batch_predict(DTC_model, X_tr_set1)

## doing test prediction
y_test_pred = batch_predict(DTC_model, X_te_set1)

ROC_curve_with_best_parameter(y_train_pred,y_test_pred)
```



3.1.2 using TFIDF W2V

In [104]:
```
DTC_model = DecisionTreeClassifier(max_depth=50, min_samples_split= 500)
DTC_model.fit(X_tr_set2, y_train)

y_train_pred = batch_predict(DTC_model, X_tr_set2)
y_test_pred = batch_predict(DTC_model, X_te_set2)
ROC_curve_with_best_parameter(y_train_pred,y_test_pred)
```



3.2 confusion matrix with best hyperparameter

3.2.1 using TFIDF

In [89]:
```
DTC_model = DecisionTreeClassifier(max_depth=50, min_samples_split= 500)
DTC_model.fit(X_tr_set1, y_train)
## finding confusion matrix ###
confusion_matrix(y_test,DTC_model.predict(X_te_set1))
```

Out[89]:
```
array([[  365,  2277],
       [ 1231, 12627]], dtype=int64)
```

3.2.2 using TFIDF W2V

In [88]:
```
## finding confusion matrix ###
DTC_model = DecisionTreeClassifier(max_depth=50, min_samples_split= 500)
DTC_model.fit(X_tr_set2, y_train)
## finding confusion matrix ###
confusion_matrix(y_test,DTC_model.predict(X_te_set2))
```

Out[88]:
```
array([[   62,  2580],
       [  120, 13738]], dtype=int64)
```

3.3 plot WorldCloud with essay data point and false positive point of set_2

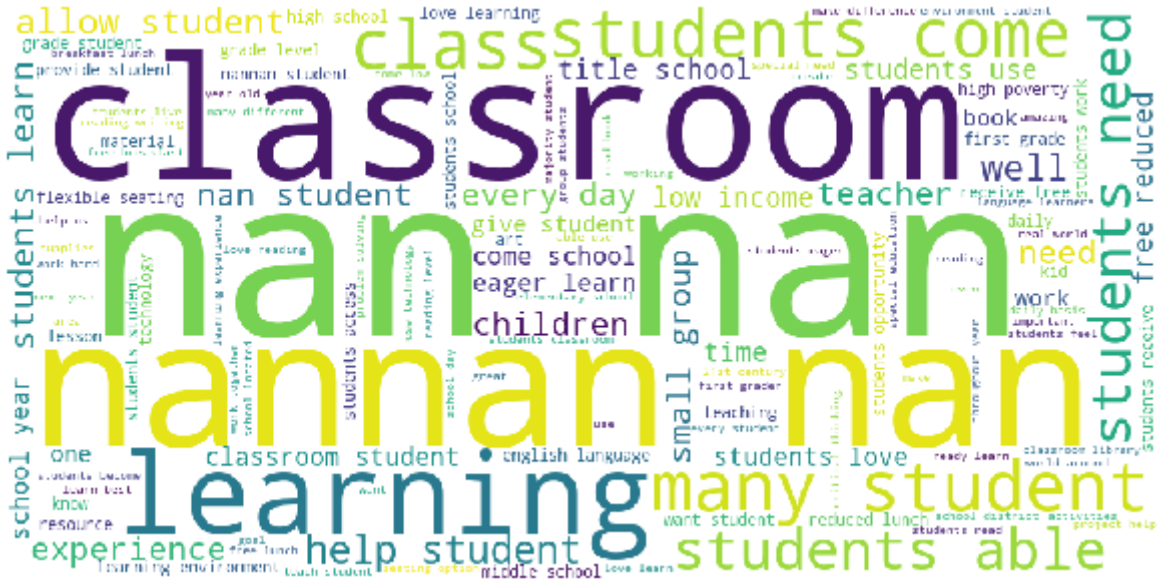
In [203]:
```
indices = []
DTC_model = DecisionTreeClassifier(max_depth=50, min_samples_split= 500)
DTC_model.fit(X_tr_set2, y_train)
y_predict=DTC_model.predict(X_te_set2)
count =0
for i in range(len(y_test)):
    if y_test[i]==0 and y_predict[i]==1:
        indices.append(count)
    count += 1
```

In [135]:

```
##https://www.geeksforgeeks.org/generating-word-cloud-python###

from wordcloud import WordCloud, STOPWORDS #import librarires

essay_words = ' '
stopwords = set(STOPWORDS)
```

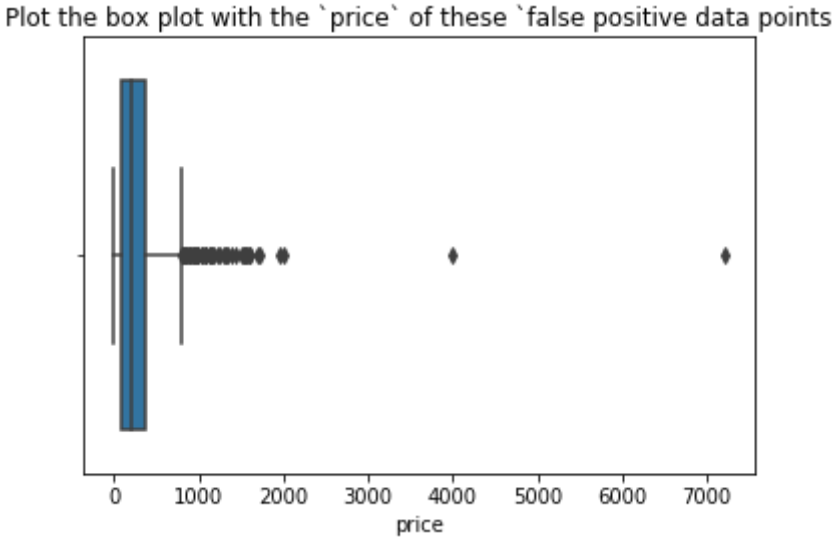
In [147]:

```
for val in (X_train['essay'][indices]):

    value = str(val) #typecast the value

        # split the value
    tokens = value.split()

        # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        essay_words = essay_words + words + ' '


wordcloud = WordCloud(width = 1000, height = 500,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(essay_words)

# here i am plotting the image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



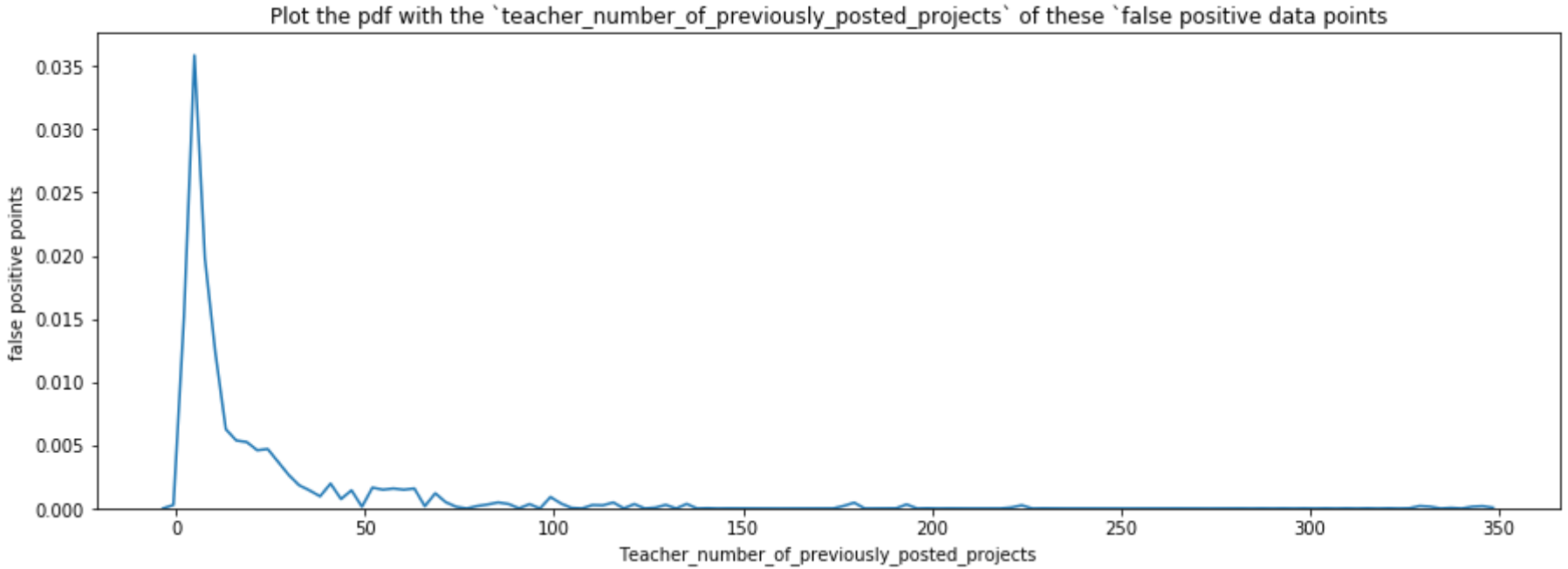
3.4 Plot the box plot with the `price` of these `false positive data points'

In [205]:
```
sns.boxplot(X_train['price'][indices])

plt.title('Plot the box plot with the `price` of these `false positive data points')
```



3.5 Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points

In [146]:
```
plt.figure(figsize=(15,5)) #giving the size of box

#fit he value of column
sns.distplot(X_train['teacher_number_of_previously_posted_projects'][indices], hist=False)

# this line is for title of graph
plt.title('Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points')

plt.xlabel('Teacher_number_of_previously_posted_projects') #denoting X-axis

plt.ylabel('false positive points') #denoting Y-axis
plt.show()
```



4. Task 2: extract feature importance

In [175]:
```
## i take help from below source
# https://stackoverflow.com/questions/47111434/randomforestregressor-and-feature-importances-error

def feature_importance(model, X):
    return X[:,model.feature_importances_.argsort()[::-1]]
```

In [176]:
```
from sklearn.tree import DecisionTreeClassifier

DTC_model= DecisionTreeClassifier()

DTC_model.fit(X_tr_set1,y_train)

X_train_best_feature = feature_importance(DTC_model,X_tr_set1)

X_test_best_feature  = feature_importance(DTC_model, X_te_set1)
```

In [177]:
```
print(X_final_train.shape)
print(X_final_test.shape)
```

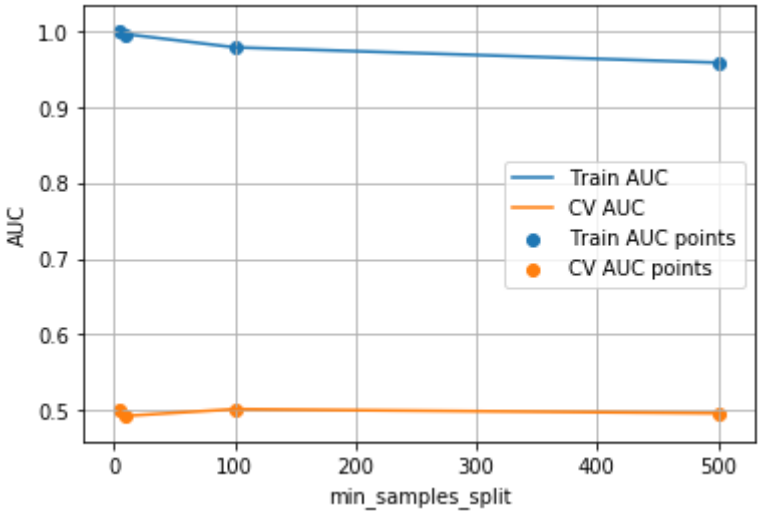
```
(22445, 5000)
(16500, 5000)
```

4.1 find best hyper parameter:

In [179]:

```
from sklearn.tree import DecisionTreeClassifier


min_samples_split = [5, 10, 100, 500]

for j in tqdm(min_samples_split):
    DTC_model = DecisionTreeClassifier(min_samples_split= j) #checking ,model with all values of alpha
    DTC_model.fit(X_train_best_feature, y_train)

    y_train_pred = batch_predict(DTC_model, X_train_best_feature)
    y_cv_pred = batch_predict(DTC_model, X_cr_set1)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
plt.plot(min_samples_split, train_auc, label='Train AUC')
plt.plot(min_samples_split, cv_auc, label='CV AUC')

plt.scatter(min_samples_split, train_auc, label='Train AUC points')
plt.scatter(min_samples_split, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("min_samples_split")
plt.ylabel("AUC")
plt.grid()
plt.show()
```

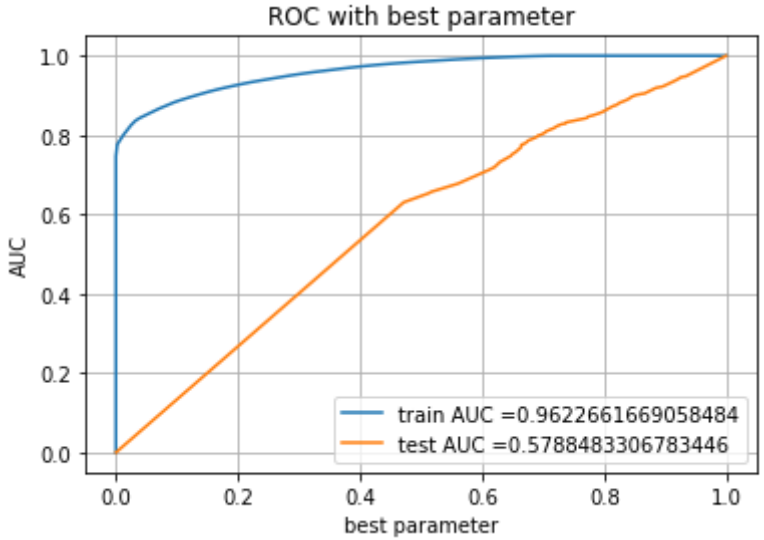
```
  0%|                                                                                            | 0/4 [00:00<?, ?it/
s]

 25%|█████████████████████▋                                                                 | 1/4 [02:02<06:07, 122.42s/
it]

 50%|██████████████████████████████████████████                                          | 2/4 [03:47<03:54, 117.22s/i
t]

 75%|██████████████████████████████████████████████████████████████▌                     | 3/4 [05:37<01:54, 114.95s/
it]

100%|████████████████████████████████████████████████████████████████████████████████████| 4/4 [07:23<00:00, 110.94s/i
t]
```



observation: with the help oh above graph best min_sample_split = 500

4.2 ROC curve with best min_sample_split

In [180]:
```
DTC_model = DecisionTreeClassifier(min_samples_split= 500)

## fit values in the model ##
DTC_model.fit(X_train_best_feature, y_train)

## doing train prediction
y_train_pred = batch_predict(DTC_model, X_train_best_feature)

## doing test prediction
y_test_pred = batch_predict(DTC_model, X_test_best_feature)

ROC_curve_with_best_parameter(y_train_pred,y_test_pred)
```



4.3 confusion matrix with best min_sample_split:

In [181]:
```
DTC_model = DecisionTreeClassifier(min_samples_split= 500)
DTC_model.fit(X_train_best_feature, y_train)
## finding confusion matrix ###
confusion_matrix(y_test,DTC_model.predict(X_test_best_feature))
```

Out[181]:
```
array([[  491,  2151],
       [ 1953, 11905]], dtype=int64)
```

4.4 calculate indices of false positive points

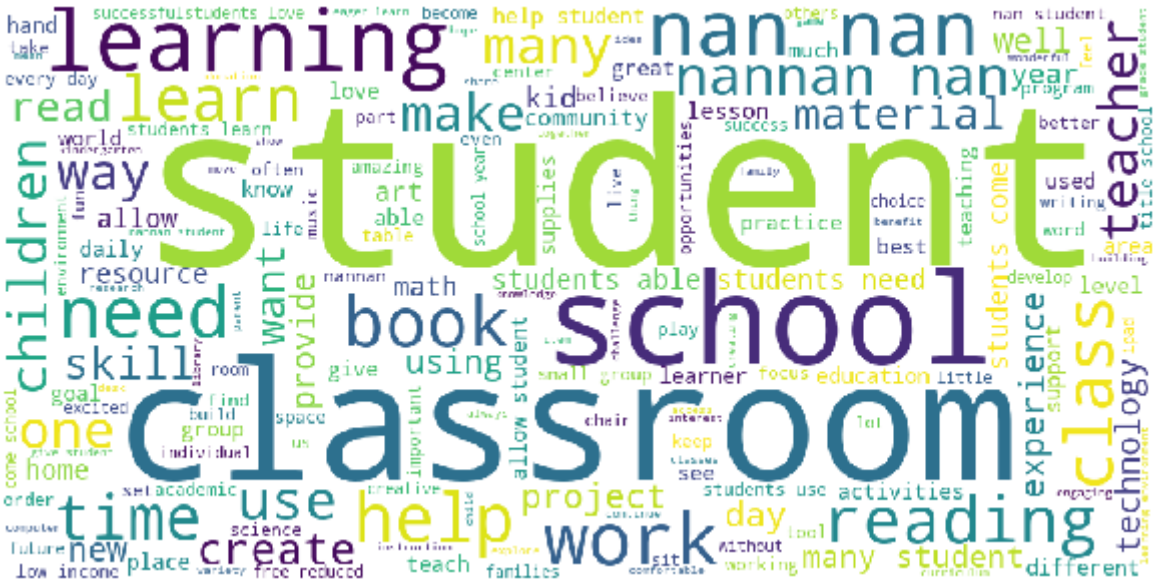
In [206]:
```
indices = [] #this is the list of false positive points


DTC_model = DecisionTreeClassifier(min_samples_split= 500) #male model with best parameter

DTC_model.fit(X_train_best_feature, y_train) # fit the data

y_predict=DTC_model.predict(X_test_best_feature) #predict the data

count =0
for i in range(len(y_test)):

    if y_test[i]==0 and y_predict[i]==1: # condition of false positive points
        indices.append(count)
    count += 1
```

In [183]:
```
##https://www.geeksforgeeks.org/generating-word-cloud-python###

from wordcloud import WordCloud, STOPWORDS #import librarires

essay_words = ' '
stopwords = set(STOPWORDS)
```

4.5 i am ploting WordCloud with feature importance

In [184]:

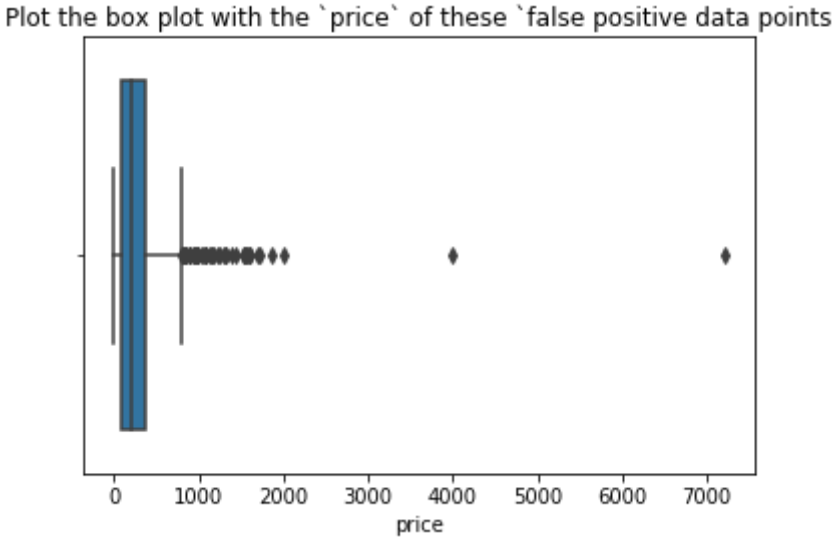
```
for val in (X_train['essay'][indices]):

    value = str(val) #typecast the value

        # split the value
    tokens = value.split()

        # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        essay_words = essay_words + words + ' '


wordcloud = WordCloud(width = 1000, height = 500,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(essay_words)

# here i am plotting the image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



4.6 i am ploting box plot with importance features

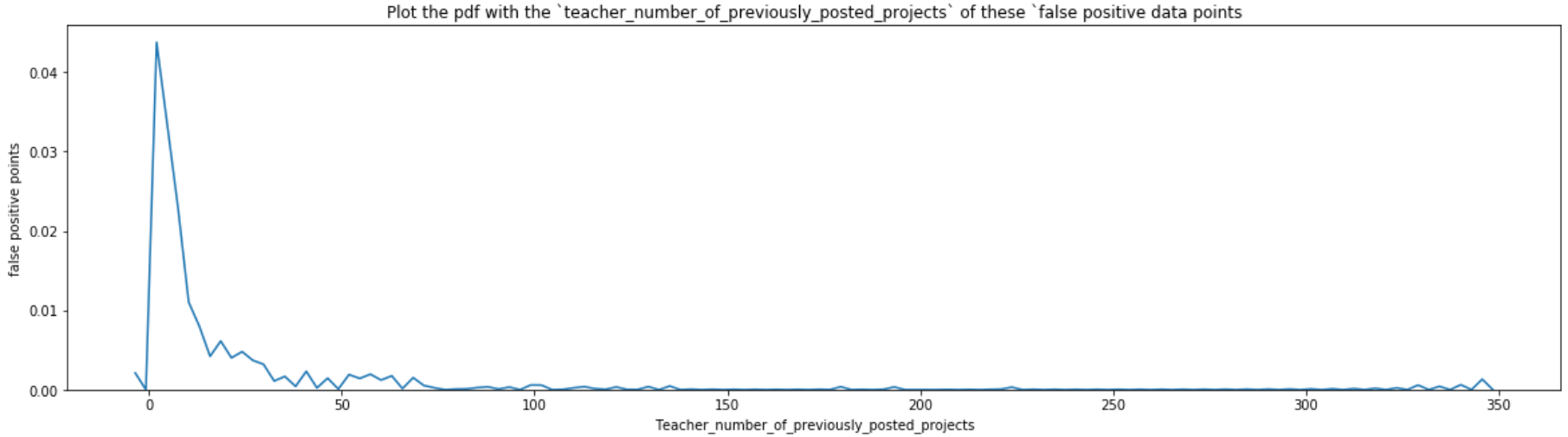
In [218]:
```
sns.boxplot(X_train['price'][indices])

plt.title('Plot the box plot with the `price` of these `false positive data points')
```

Out[218]: Text(0.5, 1.0, 'Plot the box plot with the `price` of these `false positive data points')



4.7 ploting PDF with importance features

In [216]:
```
plt.figure(figsize=(20,5))
sns.distplot(X_train['teacher_number_of_previously_posted_projects'][indices], hist=False)
plt.title('Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points')
plt.xlabel('Teacher_number_of_previously_posted_projects')
plt.ylabel('false positive points')
plt.show()
```



5. Summarize result in table format:

In [213]:
```
#    i have taken help from below link:
#    https://stackoverflow.com/questions/39032720/formatting-lists-into-columns-of-a-table-output-python-3
#    importing libraries
import texttable as tt

# creating model
table = tt.Texttable()
headings = ['Vectorizer','Model','max_depth','min_sample_split','AUC']
table.header(headings)
Vectorizer = ['TFIDF', 'TFIDFW2V','TFIDF(Importance feature)']
Model = ['DecisionTree' , 'DecisionTree','DecisionTree']
max_depth = [50 , 50 , 'NA']
min_sample_split = [500 , 500 , 500]
AUC = [0.574,0.627,0.578]

for row in zip(Vectorizer,Model,max_depth,min_sample_split,AUC):
    table.add_row(row)

summarize_table = table.draw()
print (summarize_table)
```

```
+------------------------+--------------+-----------+------------------+-------+
|       Vectorizer       |    Model     | max_depth | min_sample_split |  AUC  |
+========================+==============+===========+==================+=======+
| TFIDF                  | DecisionTree | 50        | 500              | 0.574 |
+------------------------+--------------+-----------+------------------+-------+
| TFIDFW2V               | DecisionTree | 50        | 500              | 0.627 |
+------------------------+--------------+-----------+------------------+-------+
| TFIDF(Importance       | DecisionTree | NA        | 500              | 0.578 |
| feature)               |              |           |                  |       |
+------------------------+--------------+-----------+------------------+-------+
```

In [ ]: