Apply Decision Tree

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

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

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

In [2]:

```
project_data = pd.read_csv("train_data.csv", nrows = 75000)
resource_data = pd.read_csv("resources.csv", nrows = 75000)
```

In [3]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (75000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

```
# Let's check for any "null" or "missing" values
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 75000 entries, 0 to 74999
Data columns (total 17 columns):
Unnamed: 0                                      75000 non-null int64
id                                              75000 non-null object
teacher_id                                      75000 non-null object
teacher_prefix                                  74997 non-null object
school_state                                    75000 non-null object
project_submitted_datetime                      75000 non-null object
project_grade_category                          75000 non-null object
project_subject_categories                      75000 non-null object
project_subject_subcategories                   75000 non-null object
project_title                                   75000 non-null object
project_essay_1                                 75000 non-null object
project_essay_2                                 75000 non-null object
project_essay_3                                 2558 non-null object
project_essay_4                                 2558 non-null object
project_resource_summary                        75000 non-null object
teacher_number_of_previously_posted_projects    75000 non-null int64
project_is_approved                             75000 non-null int64
dtypes: int64(3), object(14)
memory usage: 9.7+ MB
```

In [5]:

```
project_data['teacher_prefix'].isna().sum()
```

Out[5]:

```
3
```

In [6]:

```
# "teacher_prefix" seems to contain 3 "missing" values, let't use mode replacement strategy to fil
l those missing values
project_data['teacher_prefix'].mode()
```

Out[6]:

```
0    Mrs.
dtype: object
```

In [7]:

```
# Let's replace the missing values with "Mrs." , as it is the mode of the "teacher_prefix"
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('Mrs.')
```

In [8]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [9]:

```
# Let's select only the selected features or columns, dropping "project_resource_summary" as it is
optional
#
project_data.drop(['id','teacher_id','project_submitted_datetime','project_resource_summary'],axis
=1, inplace=True)
project_data.columns
```

Out[9]:

```
Index(['Unnamed: 0', 'teacher_prefix', 'school_state',
       'project_grade_category', 'project_subject_categories',
```

```
       'project_subject_subcategories', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'price', 'quantity'],
      dtype='object')
```

In [10]:

```
# Data seems to be highly imbalanced since the ratio of "class 1" to "class 0" is nearly 5.5
project_data['project_is_approved'].value_counts()
```

Out[10]:

```
1    63632
0    11368
Name: project_is_approved, dtype: int64
```

In [11]:

```
number_of_approved = project_data['project_is_approved'][project_data['project_is_approved'] == 1].
count()
number_of_not_approved = project_data['project_is_approved'][project_data['project_is_approved'] =
= 0].count()

print("Ratio of Project approved to Not approved is:", number_of_approved/number_of_not_approved)
```

```
Ratio of Project approved to Not approved is: 5.597466572836031
```

In [12]:

```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [13]:

```
project_data.head(2)
```

Out[13]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
0	160221	Mrs.	IN	Grades PreK-2	Literacy & Language	ESL, Literacy
1	140945	Mr.	FL	Grades 6-8	History & Civics, Health & Sports	Civics & Government, Team Sports

In [14]:

```
# Let's drop the project essay columns from the dadaset now, as we have captured the essay text da
ta into single "essay" column
project_data.drop(['project_essay_1','project_essay_2','project_essay_3','project_essay_4'],axis=1
, inplace=True)
```

In [15]:

```
y = project_data['project_is_approved'].values
X = project_data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[15]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcateg
0	160221	Mrs.	IN	Grades PreK-2	Literacy & Language	ESL, Literacy

In [16]:

```
# 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)
```

2) Make Data Model Ready: encoding numerical, categorical features

In [17]:

```
def cleaning_text_data(list_text_feature,df,old_col_name,new_col_name):

    # remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
    # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
    # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
    # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
    feature_list = []
    for i in list_text_feature:
        temp = ""
        # consider we have text like this "Math & Science, Warmth, Care & Hunger"
        for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care
& Hunger"]
            if 'The' in j.split(): # this will split each of the catogory based on space "Math & Sc
ience"=> "Math","&", "Science"
                j=j.replace('The','') # if we have the words "The" we are going to replace it with
''(i.e removing 'The')
            j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Sc
ience"=>"Math&Science"
            temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
            temp = temp.replace('&','_') # we are replacing the & value into
        feature_list.append(temp.strip())

    df[new_col_name] = feature_list
    df.drop([old_col_name], axis=1, inplace=True)

    from collections import Counter
    my_counter = Counter()
    for word in df[new_col_name].values:
        my_counter.update(word.split())

    feature_dict = dict(my_counter)
    sorted_feature_dict = dict(sorted(feature_dict.items(), key=lambda kv: kv[1]))
    return sorted_feature_dict
```

In [18]:

```
def clean_project_grade(list_text_feature,df,old_col_name,new_col_name):
```

```
    # remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
    # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
    # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
    # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
    feature_list = []
    for i in list_text_feature:
        temp = i.split(' ')
        last_dig = temp[-1].split('-')
        fin = [temp[0]]
        fin.extend(last_dig)
        feature = '_'.join(fin)
        feature_list.append(feature.strip())

    df[new_col_name] = feature_list
    df.drop([old_col_name], axis=1, inplace=True)

    from collections import Counter
    my_counter = Counter()
    for word in df[new_col_name].values:
        my_counter.update(word.split())

    feature_dict = dict(my_counter)
    sorted_feature_dict = dict(sorted(feature_dict.items(), key=lambda kv: kv[1]))
    return sorted_feature_dict
```

2.1) Text Preprocessing: project_subject_categories

In [19]:

```
x_train_sorted_category_dict = cleaning_text_data(X_train['project_subject_categories'],X_train,'p
roject_subject_categories','clean_categories')
x_test_sorted_category_dict =
cleaning_text_data(X_test['project_subject_categories'],X_test,'project_subject_categories','clean_
categories')
```

2.2) Text Preprocessing : project_subject_subcategories

In [20]:

```
x_train_sorted_subcategories = cleaning_text_data(X_train['project_subject_subcategories'],X_train
,'project_subject_subcategories','clean_subcategories')
x_test_sorted_subcategories = cleaning_text_data(X_test['project_subject_subcategories'],X_test,'p
roject_subject_subcategories','clean_subcategories')
```

2.3) Text Preprocessing: project_grade_category

In [21]:

```
x_train_sorted_grade =
clean_project_grade(X_train['project_grade_category'],X_train,'project_grade_category','clean_grade
')
x_test_sorted_grade =
clean_project_grade(X_test['project_grade_category'],X_test,'project_grade_category','clean_grade'
)
```

2.4) Text Preprocessing (stowords): project_essay, project_title

In [22]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
```

```
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [23]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [24]:

```
# Combining all the above stundents
from tqdm import tqdm
def process_text(df,col_name):
    preprocessed_feature = []
    # tqdm is for printing the status bar
    for sentance in tqdm(df[col_name].values):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\"', ' ')
        sent = sent.replace('\\n', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
        preprocessed_feature.append(sent.lower().strip())
    return preprocessed_feature
```

In [25]:

```
x_train_essay_preprocessed = process_text(X_train,'essay')
x_test_essay_preprocessed = process_text(X_test,'essay')
```

```
100%|████████████████████████████████████████████████████████████████| 50250/50250
[00:37<00:00, 1345.96it/s]
100%|████████████████████████████████████████████████████████████████| 24750/24750
[00:18<00:00, 1365.59it/s]
```

In [26]:

```
x_train_title_preprocessed = process_text(X_train,'project_title')
x_test_title_preprocessed = process_text(X_test,'project_title')
```

```
100%|████████████████████████████████████████████████████████████| 50250/50250
[00:01<00:00, 26559.23it/s]
100%|████████████████████████████████████████████████████████████| 24750/24750
[00:00<00:00, 26245.07it/s]
```

2.5) Vectorizing Categorical Data

project_subject_categories (clean_categories)

In [27]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
def cat_vectorizer(X_train,df,col_name):
    vectorizer = CountVectorizer()
    vectorizer.fit(X_train[col_name].values)
    feature_one_hot = vectorizer.transform(df[col_name].values)
    print(vectorizer.get_feature_names())
    return feature_one_hot, vectorizer.get_feature_names()
```

In [28]:

```
x_train_cat_one_hot, x_train_cat_feat_list = cat_vectorizer(X_train,X_train,'clean_categories')
x_test_cat_one_hot, x_test_cat_feat_list = cat_vectorizer(X_train,X_test,'clean_categories')
```

```
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
```

In [29]:

```
# shape after categorical one hot encoding
print(x_train_cat_one_hot.shape)
print(x_test_cat_one_hot.shape)
```

```
(50250, 9)
(24750, 9)
```

project_subject_subcategory (clean_subcategory)

In [30]:

```
x_train_subcat_one_hot, x_train_subcat_feat_list =
cat_vectorizer(X_train,X_train,'clean_subcategories')
x_test_subcat_one_hot, x_test_subcat_feat_list =
cat_vectorizer(X_train,X_test,'clean_subcategories')
```

```
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
```

```
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
```

In [31]:

```
# shape after categorical one hot encoding
print(x_train_subcat_one_hot.shape)
print(x_test_subcat_one_hot.shape)
```

```
(50250, 30)
(24750, 30)
```

school_state

In [32]:

```
# we use count vectorizer to convert the values into one hot encoding
# CountVectorizer for "school_state"
x_train_state_one_hot, x_train_state_feat_list = cat_vectorizer(X_train,X_train,'school_state')
x_test_state_one_hot, x_test_state_feat_list = cat_vectorizer(X_train,X_test,'school_state')
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', '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']
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', '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']
```

In [33]:

```
# shape after categorical one hot encoding
print(x_train_state_one_hot.shape)
print(x_test_state_one_hot.shape)
```

```
(50250, 51)
(24750, 51)
```

teacher_prefix

In [34]:

```
# we use count vectorizer to convert the values into one hot encoding
# CountVectorizer for teacher_prefix
x_train_teacher_prefix_one_hot,x_train_teacher_prefix_feat_list = cat_vectorizer(X_train,X_train,'
teacher_prefix')
x_test_teacher_prefix_one_hot,x_test_teacher_prefix_feat_list =
cat_vectorizer(X_train,X_test,'teacher_prefix')
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

In [35]:

```
# shape after categorical one hot encoding
print(x_train_teacher_prefix_one_hot.shape)
print(x_test_teacher_prefix_one_hot.shape)
```

```
(50250, 5)
(24750, 5)
```

project_grade_category

In [36]:

```
# using count vectorizer for one-hot encoding of project_grade_category
x_train_grade_one_hot, x_train_grade_feat_list = cat_vectorizer(X_train,X_train,'clean_grade')
x_test_grade_one_hot, x_test_grade_feat_list = cat_vectorizer(X_train,X_test,'clean_grade')
```

```
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

In [37]:

```
# shape after categorical one hot encoding
print(x_train_grade_one_hot.shape)
print(x_test_grade_one_hot.shape)
```

```
(50250, 4)
(24750, 4)
```

2.6) Vectorizing Text Data

2.6.1) TFIDF (essay)

In [38]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
def tfidf_vectorizer(X_train,col_name,df):
    vectorizer = TfidfVectorizer()
    vectorizer.fit(X_train[col_name].values)
    df_tfidf = vectorizer.transform(df[col_name].values)
    return df_tfidf, vectorizer.get_feature_names()
```

In [39]:

```
# Lets vectorize essay
x_train_essay_tfidf, x_train_essay_tfidf_feat = tfidf_vectorizer(X_train,'essay',X_train)
x_test_essay_tfidf, x_test_essay_tfidf_feat = tfidf_vectorizer(X_train,'essay',X_test)
```

In [40]:

```
print(x_train_essay_tfidf.shape)
print(x_test_essay_tfidf.shape)
```

```
(50250, 44506)
(24750, 44506)
```

2.6.4) TFIDF (title)

In [41]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
def tfidf_vectorizer_title(X_train,col_name,df):
    vectorizer = TfidfVectorizer()
    vectorizer.fit(X_train[col_name].values)
    df_tfidf = vectorizer.transform(df[col_name].values)
    return df_tfidf, vectorizer.get_feature_names()
```

In [42]:

```
# Lets vectorize essay
x_train_title_tfidf, x_train_title_tfidf_feat =
tfidf_vectorizer_title(X_train,'project_title',X_train)
x_test_title_tfidf, x_test_title_tfidf_feat =
tfidf_vectorizer_title(X_train,'project_title',X_test)
```

In [43]:

```
print(x_train_title_tfidf.shape)
print(x_test_title_tfidf.shape)
```

```
(50250, 12075)
(24750, 12075)
```

In [44]:

```
# Combining all the above stundents
from tqdm import tqdm
def preprocess_essay(df,col_name):
    preprocessed_essays = []
    # tqdm is for printing the status bar
    for sentance in tqdm(df[col_name].values):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\"', ' ')
        sent = sent.replace('\\n', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(e for e in sent.split() if e not in stopwords)
        preprocessed_essays.append(sent.lower().strip())
    return preprocessed_essays
```

In [45]:

```
# average Word2Vec
# compute average word2vec for each review.
def compute_avg_W2V(preprocessed_feature):
    avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(preprocessed_feature): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        cnt_words =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:
                vector += model[word]
                cnt_words += 1
        if cnt_words != 0:
            vector /= cnt_words
        avg_w2v_vectors.append(vector)
    return avg_w2v_vectors
```

In [46]:

```
x_train_preprocessed_essay = preprocess_essay(X_train,'essay')
x_test_preprocessed_essay = preprocess_essay(X_test,'essay')
```

```
100%|████████████████████████████████████████████████████████████████| 50250/50250
[00:45<00:00, 1116.54it/s]
100%|████████████████████████████████████████████████████████████████| 24750/24750 [00:
26<00:00, 932.62it/s]
```

In [47]:

```
x_train_preprocessed_title = preprocess_essay(X_train,'project_title')
x_test_preprocessed_title = preprocess_essay(X_test,'project_title')
```

```
100%|████████████████████████████████████████████████████████████████| 50250/50250
[00:02<00:00, 18813.17it/s]
100%|████████████████████████████████████████████████████████████████| 24750/24750
[00:01<00:00, 17741.82it/s]
```

2.6.5) Using Pretrained Models: TFIDF Weighted W2V

In [48]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
```

```
# storing variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-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 [49]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
def get_tfidf_dict(preprocessed_feature):
    tfidf_model = TfidfVectorizer()
    tfidf_model.fit(preprocessed_feature)
    # 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())
    return dictionary, tfidf_words
```

In [50]:

```
# average Word2Vec
# compute average word2vec for each review.
def compute_tfidf_w2v_vectors(preprocessed_feature):
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    dictionary, tfidf_words = get_tfidf_dict(preprocessed_feature)
    for sentence in tqdm(preprocessed_feature): # 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_vectors.append(vector)
    return tfidf_w2v_vectors
```

In [51]:

```
x_train_weighted_w2v_essay = compute_tfidf_w2v_vectors(x_train_essay_preprocessed)
x_test_weighted_w2v_essay= compute_tfidf_w2v_vectors(x_test_essay_preprocessed)
```

```
100%|████████████████████████████████████████████████████████████████| 50250/50250 [05:53<00:00, 142.29it/s]
100%|████████████████████████████████████████████████████████████████| 24750/24750 [01:10<00:00, 352.73it/s]
```

In [52]:

```
x_train_weighted_w2v_title = compute_tfidf_w2v_vectors(x_train_title_preprocessed)
x_test_weighted_w2v_title= compute_tfidf_w2v_vectors(x_test_title_preprocessed)
```

```
100%|████████████████████████████████████████████████████████████████| 50250/50250 [00:02<00:00, 22945.42it/s]
100%|████████████████████████████████████████████████████████████████| 24750/24750 [00:01<00:00, 22098.45it/s]
```

2.6.6) Vectorizing Numerical Features

We have 2 numerical features left, "price" and "teacher_number_of_previously_posted_projects". Let's check for the "missing" or "NaN" values present in those numerical features and use "Mean Replacement" for "price" and "Mode Replacement" for "teacher_number_of_previously_posted_projects".

In [53]:

```
print("Total number of \"Missing\" Values present in X_train price:",X_train['price'].isna().sum()
)
print("Total number of \"Missing\" Values present in X_test price:",X_test['price'].isna().sum())
```

```
Total number of "Missing" Values present in X_train price: 47806
Total number of "Missing" Values present in X_test price: 23580
```

In [54]:

```
print("Total number of \"Missing\" Values present in X_train previous teacher number:",X_train['te
acher_number_of_previously_posted_projects'].isna().sum())
print("Total number of \"Missing\" Values present in X_test previous teacher number:",X_test['teac
her_number_of_previously_posted_projects'].isna().sum())
```

```
Total number of "Missing" Values present in X_train previous teacher number: 0
Total number of "Missing" Values present in X_test previous teacher number: 0
```

In [55]:

```
print("Total number of \"Missing\" Values present in X_train quantity:",X_train['quantity'].isna()
.sum())
print("Total number of \"Missing\" Values present in X_test quantity:",X_test['quantity'].isna().s
um())
```

```
Total number of "Missing" Values present in X_train quantity: 47806
Total number of "Missing" Values present in X_test quantity: 23580
```

"teacher_number_of_previously_posted_projects" does not have any "missing" values.

In [56]:

```
X_train['price'].mean()
```

Out[56]:

```
306.590945171848
```

In [57]:

```
X_train['price'] = X_train['price'].fillna(306.5909)
```

In [58]:

```
X_test['price'].mean()
```

Out[58]:

```
275.04324786324764
```

In [59]:

```
X_test['price'] = X_test['price'].fillna(275.0432)
```

In [60]:

```
print(X_train['quantity'].mean())
print(X_test['quantity'].mean())
```

```
18.592880523731587
18.38290598290598
```

In [61]:

```
X_train['quantity'] = X_train['quantity'].fillna(18.5928)
X_test['quantity'] = X_test['quantity'].fillna(18.38291)
```

In [62]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler
def scaler_function(df,col_name):

    scaler = StandardScaler()
    scaler.fit(df[col_name].values.reshape(-1,1)) # finding the mean and standard deviation of this
data

    # Now standardize the data with above maen and variance.
    print(f"Mean : {scaler.mean_[0]}, Standard deviation : {np.sqrt(scaler.var_[0])}")
    scaled = scaler.transform(df[col_name].values.reshape(-1, 1))
    return scaled
```

teacher_number_of_previously_posted_projects

In [63]:

```
x_train_teacher_number = scaler_function(X_train,'teacher_number_of_previously_posted_projects')
x_test_teacher_number = scaler_function(X_test,'teacher_number_of_previously_posted_projects')
```

```
Mean : 11.213810945273632, Standard deviation : 27.988796200272308
Mean : 11.282222222222222, Standard deviation : 27.9328036962718
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

price

In [64]:

```
x_train_price = scaler_function(X_train,'price')
x_test_price = scaler_function(X_test,'price')
```

```
Mean : 306.59090219701494, Standard deviation : 89.75922049893008
Mean : 275.0432022626263, Standard deviation : 62.219442683983296
```

quantity

In [65]:

```
x_train_quantity = scaler_function(X_train,'quantity')
x_test_quantity = scaler_function(X_test,'quantity')
```

```
Mean : 18.592803916417914, Standard deviation : 7.11005442686334
Mean : 18.382909810101008, Standard deviation : 5.477566583949763
```

2.6.7) Calculate Sentiment Score for each Essay (Combined)

In [66]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# import nltk
# nltk.download('vader_lexicon')
def compute_sentiment_score(df):
    score_list = []
    sid = SentimentIntensityAnalyzer()
    for essay in df['essay']:
        ss = sid.polarity_scores(essay)
        score_list.append(ss)
    return score_list
# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
C:\ProgramData\Anaconda3\lib\site-packages\nltk\twitter\__init__.py:20: UserWarning:

The twython library has not been installed. Some functionality from the twitter package will not b
e available.
```

In [67]:

```
x_train_score = compute_sentiment_score(X_train)
x_test_score = compute_sentiment_score(X_test)
```

In [68]:

```
def populate_list(score_dicts):

    neg_score = []
    neu_score = []
    pos_score = []
    compound_score = []
    for dict_ in score_dicts:
        neg_score.append(dict_['neg'])
        neu_score.append(dict_['neu'])
        pos_score.append(dict_['pos'])
        compound_score.append(dict_['compound'])
    return neg_score, neu_score, pos_score, compound_score
```

In [69]:

```
x_train_neg, x_train_neu, x_train_pos, x_train_compound = populate_list(x_train_score)
x_test_neg, x_test_neu, x_test_pos, x_test_compound = populate_list(x_test_score)
```

In [70]:

```
# for training set
X_train['neg'] = x_train_neg
X_train['neu'] = x_train_neu
X_train['pos'] = x_train_pos
X_train['compound'] = x_train_compound

# for testing set
X_test['neg'] = x_test_neg
X_test['neu'] = x_test_neu
X_test['pos'] = x_test_pos
X_test['compound'] = x_test_compound
```

2.7) Merging all the features and building the sets

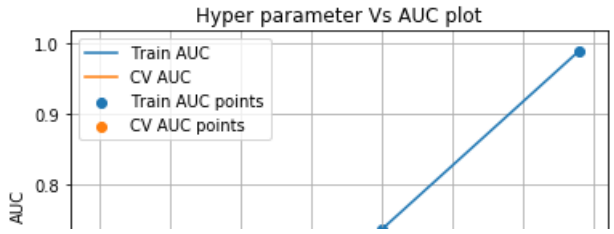
Set 1: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF) + Sentiment Scores

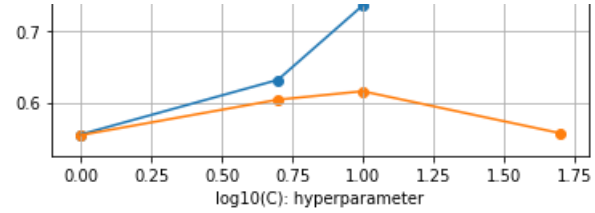
In [71]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_set_1 =
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\

x_train_grade_one_hot,x_train_teacher_number,x_train_price,x_train_quantity,x_train_title_tfidf,x_
train_essay_tfidf,\
                          X_train['neg'].values.reshape(-1,1),X_train['neu'].values.reshape(-1,1),X_
rain['pos'].values.reshape(-1,1),X_train['compound'].values.reshape(-1,1))).tocsr()
X_test_set_1 =
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                         x_test_grade_one_hot,x_test_teacher_number,x_test_price,x_test_quantity,x_t
est_title_tfidf,x_test_essay_tfidf,\
                         X_test['neg'].values.reshape(-1,1),X_test['neu'].values.reshape(-1,1),X_tes
['pos'].values.reshape(-1,1),X_test['compound'].values.reshape(-1,1))).tocsr()
```

In [72]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
import math


DT_ = DecisionTreeClassifier(class_weight = "balanced")
parameters = {'max_depth':[1,5,10,50]}
clf = RandomizedSearchCV(DT_, parameters,n_iter = 4, scoring='roc_auc')
clf.fit(X_train_set_1, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_max_depth'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
max_depth_ =  results['param_max_depth'].apply(lambda x: math.log10(x))

plt.plot(max_depth_, train_auc, label='Train AUC')

plt.plot(max_depth_, cv_auc, label='CV AUC')

plt.scatter(max_depth_, train_auc, label='Train AUC points')

plt.scatter(max_depth_, cv_auc, label='CV AUC points')



plt.legend()
plt.xlabel("log10(C): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```





Out[72]:

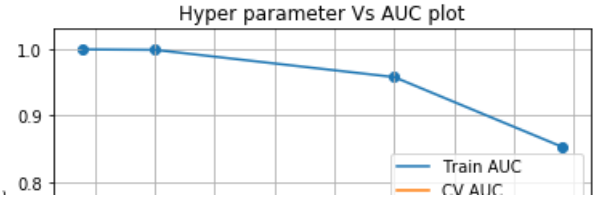
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params	split0_test_score	split1
0	2.004990	0.106749	0.093342	0.008992	1	{'max_depth': 1}	0.558490	0.547
1	6.638010	1.083193	0.110996	0.010621	5	{'max_depth': 5}	0.598721	0.608
2	12.654996	1.646709	0.084012	0.004095	10	{'max_depth': 10}	0.602836	0.627
3	65.679022	0.714597	0.110339	0.020532	50	{'max_depth': 50}	0.546385	0.559

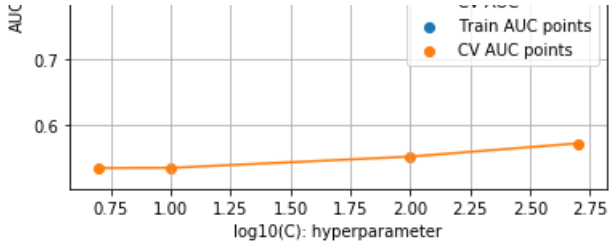
In [74]:

```
# From the AUC plot, we find that the best value for "max_depth" - for the DecisionTreeClassfier i
s 10
best_max_depth = 10
```

In [75]:

```
DT_ = DecisionTreeClassifier(class_weight = "balanced")
parameters = {'min_samples_split':[5,10,100,500]}
clf = RandomizedSearchCV(DT_, parameters,n_iter = 4, scoring='roc_auc')
clf.fit(X_train_set_1, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_min_samples_split'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
min_samples_split_ =  results['param_min_samples_split'].apply(lambda x: math.log10(x))

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("log10(C): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```





Out[75]:

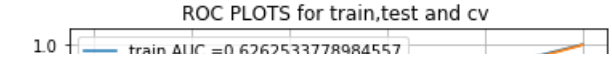
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_split	params	split0_tes
0	123.390033	7.533778	0.156257	0.025519	5	{'min_samples_split': 5}	0.530836
1	130.723176	12.434286	0.133431	0.008314	10	{'min_samples_split': 10}	0.535035
2	99.832593	2.493715	0.135417	0.007349	100	{'min_samples_split': 100}	0.549545
3	56.410905	7.079870	0.145833	0.019492	500	{'min_samples_split': 500}	0.560895

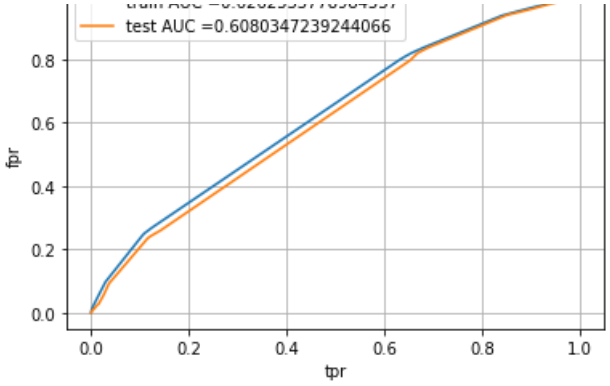
In [76]:

```
# min_samples_split from the graph above is 500
best_min_samples_split = 500
```

In [147]:

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


DT_ = DecisionTreeClassifier(max_depth=best_max_depth, min_samples_split = best_min_samples_split
,class_weight = "balanced")
DT_.fit(X_train_set_1, y_train)


y_train_pred = DT_.predict_proba(X_train_set_1)
y_test_pred =  DT_.predict_proba(X_test_set_1)

y_train_pred_prob = []
y_test_pred_prob = []

for index in range(len(y_train_pred)):
    y_train_pred_prob.append(y_train_pred[index][1])

for index in range(len(y_test_pred)):
    y_test_pred_prob.append(y_test_pred[index][1])


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)



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("tpr")
plt.ylabel("fpr")
plt.title("ROC PLOTS for train,test and cv ")
plt.grid()
plt.show()
```





In [84]:

```
def compute_auc_with_hyper_para_depth(x_tr,y_tr, x_te, y_te, depth_list):
    auc_tr = []
    auc_te = []
    y_train_pred_prob = []
    y_test_pred_prob = []

    for depth in depth_list:
        DT_ = DecisionTreeClassifier(max_depth = depth, class_weight = "balanced")
        DT_.fit(x_tr,y_tr)

        y_train_pred = DT_.predict_proba(x_tr)
        y_test_pred = DT_.predict_proba(x_te)

        for index in range(len(y_train_pred)):
            y_train_pred_prob.append(y_train_pred[index][1])

        for index in range(len(y_test_pred)):
            y_test_pred_prob.append(y_test_pred[index][1])

        train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred_prob)
        test_fpr, test_tpr, tc_thresholds = roc_curve(y_te, y_test_pred_prob)

        y_train_pred_prob = []
        y_test_pred_prob = []



        auc_tr.append(auc(train_fpr,train_tpr))
        auc_te.append(auc(test_fpr,test_tpr))
    plt.plot(depth_list,auc_tr,label="train_auc_with_max_depth")
    plt.plot(depth_list,auc_te,label="test_auc_with_max_depth")

    plt.legend()
    plt.xlabel("Hyper Parameter:(max_depth)")
    plt.ylabel("Area Under ROC Curve")
    plt.title("auc v/s hyper parameters")
    plt.grid()
    plt.show()
```

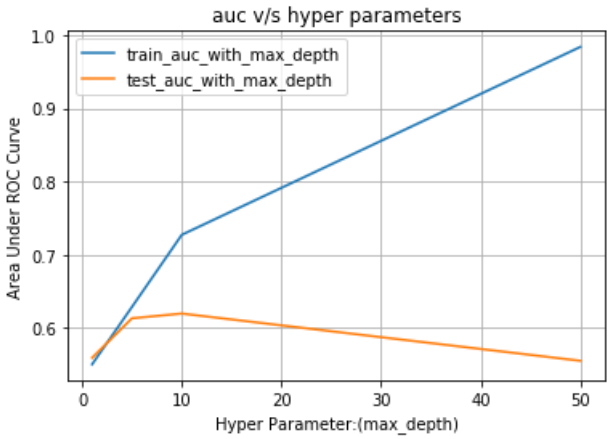
In [85]:

```
def compute_auc_with_hyper_para_split(x_tr,y_tr, x_te, y_te, split_list):
    auc_tr = []
    auc_te = []
    y_train_pred_prob = []
    y_test_pred_prob = []

    for split in split_list:
        DT_ = DecisionTreeClassifier(min_samples_split = split, class_weight = "balanced")
        DT_.fit(x_tr,y_tr)

        y_train_pred = DT_.predict_proba(x_tr)
        y_test_pred = DT_.predict_proba(x_te)

        for index in range(len(y_train_pred)):
            y_train_pred_prob.append(y_train_pred[index][1])
```

```
        for index in range(len(y_test_pred)):
            y_test_pred_prob.append(y_test_pred[index][1])

        train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred_prob)
        test_fpr, test_tpr, tc_thresholds = roc_curve(y_te, y_test_pred_prob)

        y_train_pred_prob = []
        y_test_pred_prob = []



        auc_tr.append(auc(train_fpr,train_tpr))
        auc_te.append(auc(test_fpr,test_tpr))

    plt.plot(split_list,auc_tr,label="train_auc_with_min_split")
    plt.plot(split_list,auc_te,label="test_auc_with_min_split")

    plt.legend()
    plt.xlabel("Hyper Parameter:(min_samples_split)")
    plt.ylabel("Area Under ROC Curve")
    plt.title("auc v/s hyper parameters")
    plt.grid()
    plt.show()
```

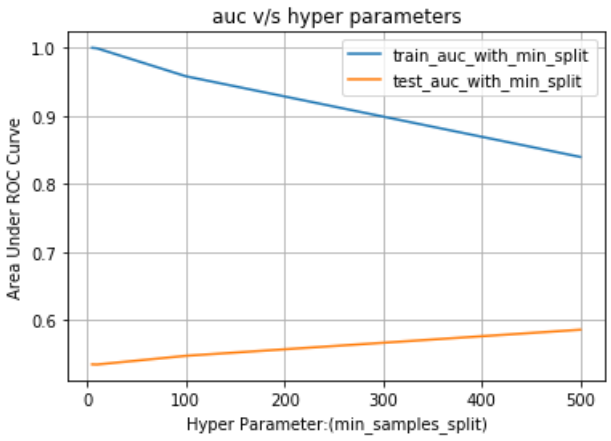
In [91]:

```
depth_list = [1,5,10,50]
compute_auc_with_hyper_para_depth(X_train_set_1,y_train,X_test_set_1, y_test, depth_list)
```



In [92]:

```
split_list = [5,10,100,500]
compute_auc_with_hyper_para_split(X_train_set_1,y_train,X_test_set_1, y_test, split_list)
```



In [148]:

```
# we are writing our own function for predict, with defined thresould
```

```
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

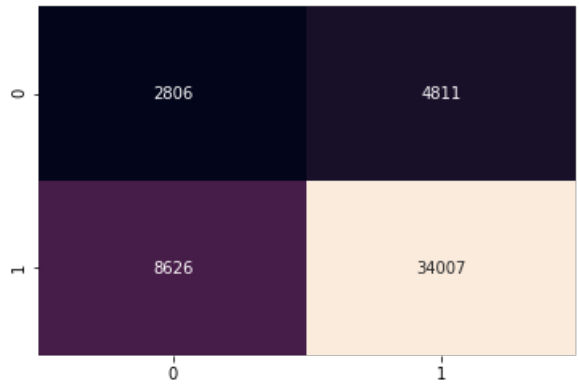
In [149]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred_prob, best_t)),annot = True,
fmt = "d", cbar=False)
```

```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.29385029993681533 for threshold 0.511
Train confusion matrix
```

Out[149]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c397329b70>
```



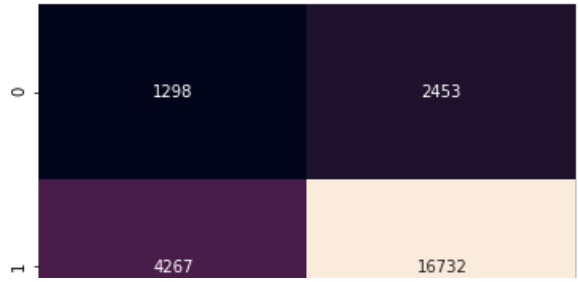
In [150]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred_prob, best_t)), annot = True,
fmt = "d", cbar=False)
```

```
Test confusion matrix
```

Out[150]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c3971fce10>
```

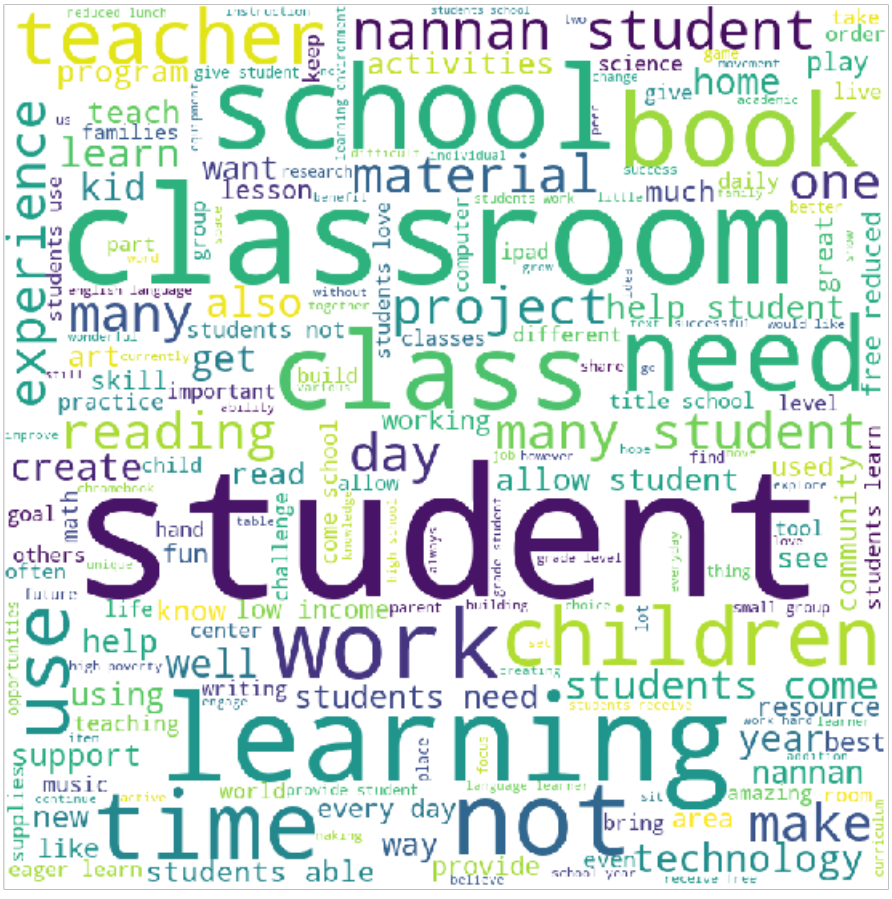


False positive points and WordCloud, box plots, PDF

Wordcloud

In [151]:

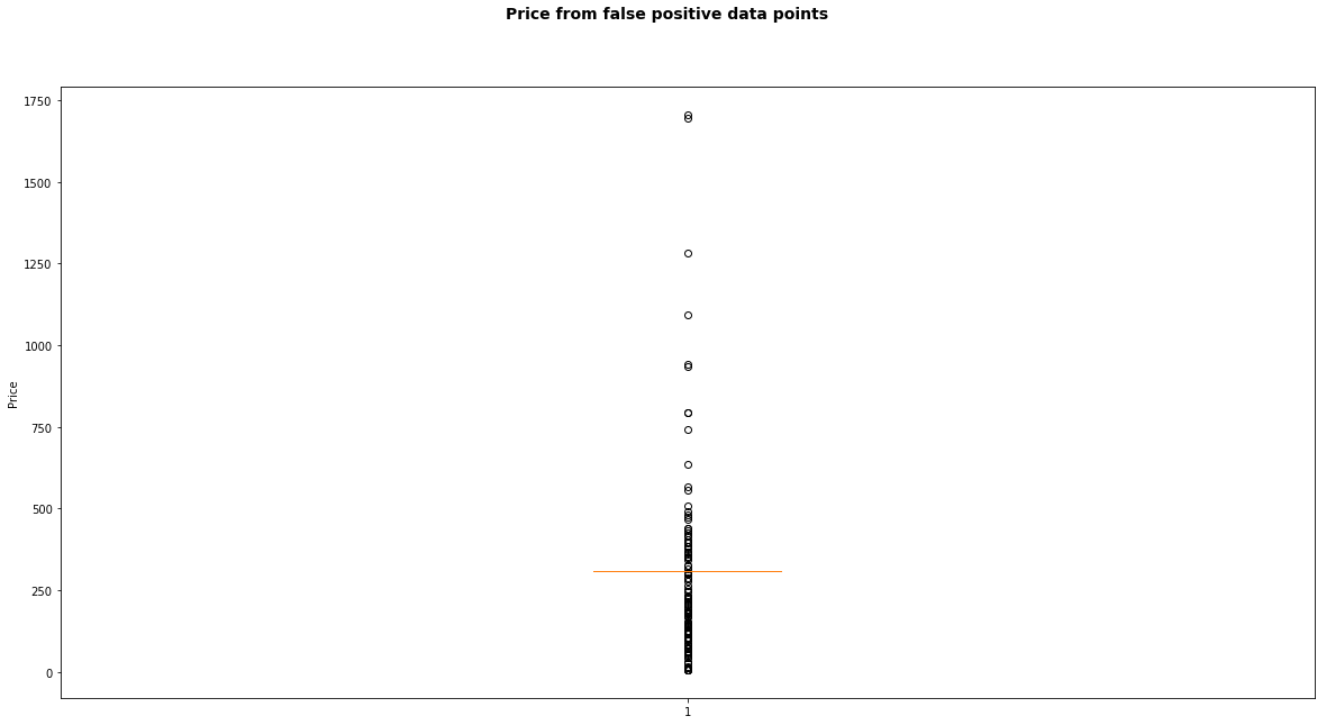
```
# y_test we have as y_test from train_test_split
# https://www.geeksforgeeks.org/generating-word-cloud-python/
from wordcloud import WordCloud
y_actual = y_test
# y_predicted would be
y_pred = predict_with_best_t(y_test_pred_prob, best_t)
essay_words = " "
essay_ = []
for i in range(len(y_pred)):
    if y_pred[i]==1 and y_actual[i]!=y_pred[i]:
            essay_.append(x_train_preprocessed_essay[i])
for essay in essay_:
    essay_words += essay + " "
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(essay_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()
```



Boxplot

In [152]:

```
y_actual = y_test
# y_predicted would be
y_pred = predict_with_best_t(y_test_pred_prob, best_t)
prices = []
for i in range(len(y_pred)):
    if y_pred[i]==1 and y_actual[i]!=y_pred[i]:
            prices.append(X_train['price'].iloc[i])
fig = plt.figure(figsize=(20,10))
fig.suptitle('Price from false positive data points', fontsize=14, fontweight='bold')

ax = fig.add_subplot(111)
ax.boxplot(prices)
ax.set_ylabel('Price')

plt.show()
```

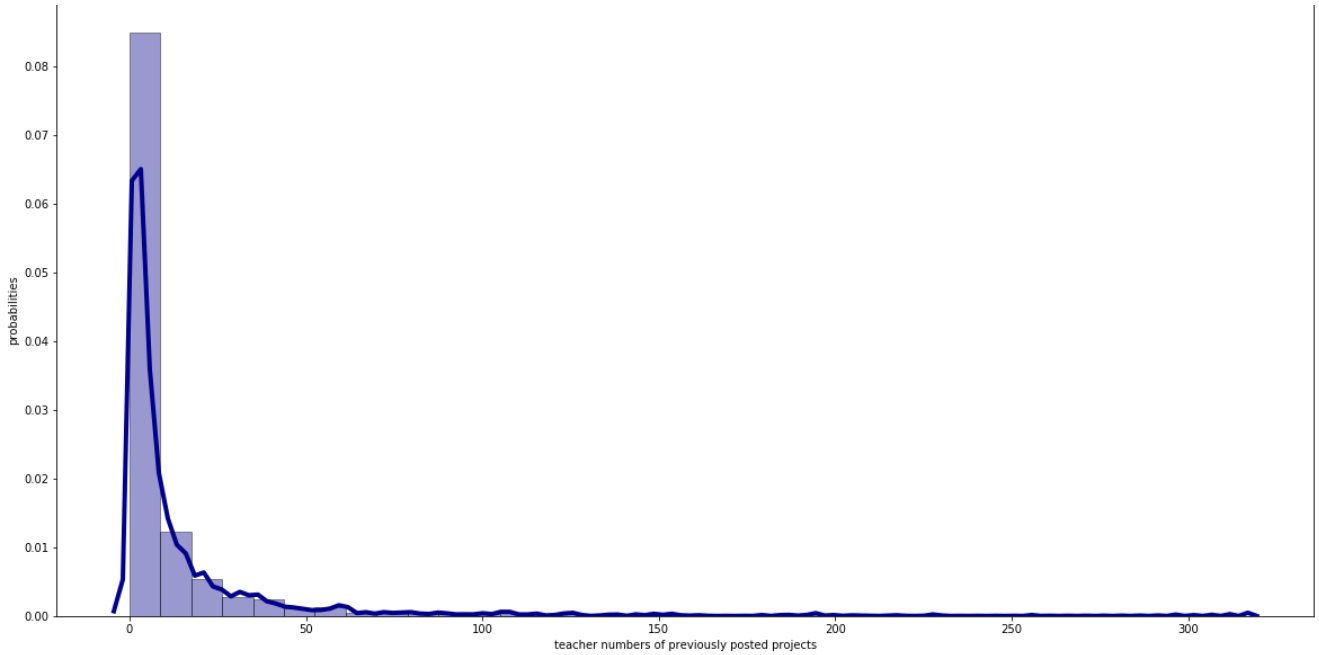


PDF for teacher_number_of_previously_posted_projects

In [153]:

```
y_actual = y_test
# y_predicted would be
y_pred = predict_with_best_t(y_test_pred_prob, best_t)
teacher_number_ = []
for i in range(len(y_pred)):
    if y_pred[i]==1 and y_actual[i]!=y_pred[i]:
            teacher_number_.append(X_train['teacher_number_of_previously_posted_projects'].iloc[i])
fig = plt.figure(figsize=(20,10))
ax = sns.distplot(teacher_number_, hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
ax.set(xlabel = 'teacher numbers of previously posted projects',ylabel = 'probabilities')
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning:

The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
```



In [78]:

```
def enable_plotly_in_cell():
  import IPython
  from plotly.offline import init_notebook_mode
  display(IPython.core.display.HTML('''<script src="/static/components/requirejs/require.js"></scr
ipt>'''))
  init_notebook_mode(connected=False)
```

In [81]:

```
%matplotlib inline
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

def plot_3d_plot(x_tr, y_tr, x_te, y_te, depth_list, split_list):
    auc_tr = []
    auc_te = []
    y_train_pred_prob = []
    y_test_pred_prob = []

    for depth, split in zip(depth_list ,split_list):
        DT_ = DecisionTreeClassifier(max_depth = depth ,min_samples_split = split, class_weight = "
balanced")
        DT_.fit(x_tr,y_tr)

        y_train_pred = DT_.predict_proba(x_tr)
        y_test_pred = DT_.predict_proba(x_te)

        for index in range(len(y_train_pred)):
            y_train_pred_prob.append(y_train_pred[index][1])

        for index in range(len(y_test_pred)):
            y_test_pred_prob.append(y_test_pred[index][1])

        train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred_prob)
        test_fpr, test_tpr, tc_thresholds = roc_curve(y_te, y_test_pred_prob)

        y_train_pred_prob = []
        y_test_pred_prob = []


        auc_tr.append(auc(train_fpr,train_tpr))
        auc_te.append(auc(test_fpr,test_tpr))
    X = split_list
    Y = depth_list
    Z1 = auc_tr
    Z2 = auc_te
    # https://plot.ly/python/3d-axes/
```

```
    trace1 = go.Scatter3d(x=X,y=Y,z=Z1, name = 'train')
    trace2 = go.Scatter3d(x=X,y=Y,z=Z2, name = 'cv')
    data = [trace1,trace2]
    enable_plotly_in_cell()

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

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

In [82]:

```
depth = [1,5,10,50]
split = [5,10,100,500]
plot_3d_plot(X_train_set_1, y_train, X_test_set_1, y_test, depth, split)
```

Feature Importance on feature set 1

In [154]:

```
DT_ = DecisionTreeClassifier(max_depth = None)
clf = DT_.fit(X_train_set_1, y_train)
feat = dict(zip(X_train.columns, clf.feature_importances_))
feat
```

Out[154]:

```
{'Unnamed: 0': 0.0,
 'teacher_prefix': 0.0,
 'school_state': 0.0005922288392830854,
 'project_title': 0.0,
 'teacher_number_of_previously_posted_projects': 0.001221402140250685,
 'price': 0.0001160649484022 1273,
 'quantity': 0.0,
 'essay': 0.0,
 'clean_categories': 0.0,
 'clean_subcategories': 0.0,
```

```
 'clean_grade': 0.0,
 'neg': 0.0,
 'neu': 0.0,
 'pos': 0.0,
 'compound': 0.0}
```

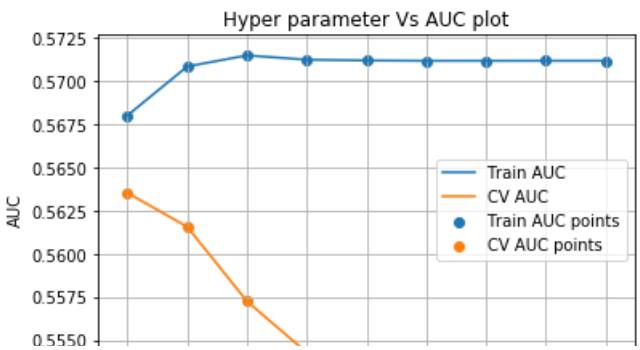
From the feature importance, we found that only "school_state","teacher_number_of_previously_posted_projects" and "price" are the important features. So we are using these features and we'll use LogisticRegression .

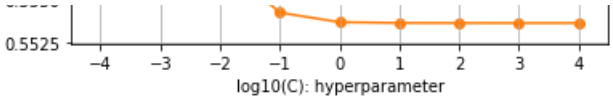
In [156]:

```
# lets prepare new set consisting non_zero feature importance
X_train_new = hstack((x_train_state_one_hot,x_train_teacher_number,x_train_price)).tocsr()
X_test_new = hstack((x_test_state_one_hot, x_test_teacher_number,x_test_price)).tocsr()
```

In [162]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
import math


log_reg = LogisticRegression(class_weight = "balanced")
parameters = {'C':[10**x for x in range(-4,5)]}
clf = RandomizedSearchCV(log_reg, parameters,n_iter = 9, scoring='roc_auc')
clf.fit(X_train_new, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_C'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
max_depth_ =  results['param_C'].apply(lambda x: math.log10(x))

plt.plot(max_depth_, train_auc, label='Train AUC')

plt.plot(max_depth_, cv_auc, label='CV AUC')

plt.scatter(max_depth_, train_auc, label='Train AUC points')

plt.scatter(max_depth_, cv_auc, label='CV AUC points')



plt.legend()
plt.xlabel("log10(C): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```





Out[162]:

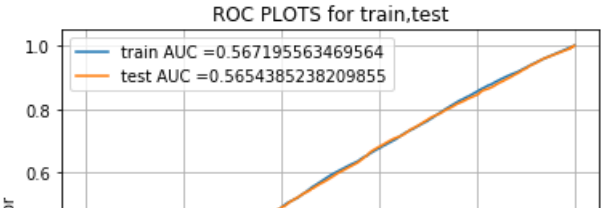
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	s
0	0.061997	0.003744	0.006004	0.000004	0.0001	{'C': 0.0001}	0.549859	0.573168	0
1	0.075011	0.008837	0.004994	0.000005	0.001	{'C': 0.001}	0.548474	0.568441	0
2	0.088333	0.005909	0.005334	0.000474	0.01	{'C': 0.01}	0.546913	0.559731	0
3	0.120657	0.002871	0.005344	0.000482	0.1	{'C': 0.1}	0.545608	0.554696	0
4	0.158999	0.015768	0.005665	0.000473	1	{'C': 1}	0.545286	0.553806	0

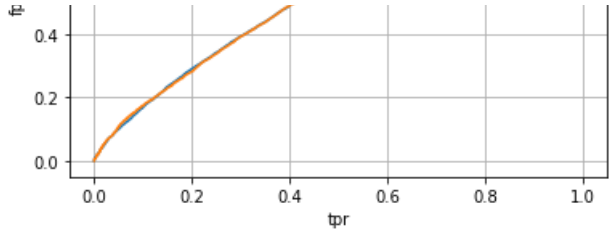
In [163]:

```
# best "c" for Logisticregression from the graph is 0.0001
best_C = 0.0001
```

In [165]:

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


log_reg = LogisticRegression(C = best_C, class_weight = "balanced")
log_reg.fit(X_train_new, y_train)


y_train_pred = log_reg.predict_proba(X_train_new)
y_test_pred =  log_reg.predict_proba(X_test_new)

y_train_pred_prob = []
y_test_pred_prob = []

for index in range(len(y_train_pred)):
    y_train_pred_prob.append(y_train_pred[index][1])

for index in range(len(y_test_pred)):
    y_test_pred_prob.append(y_test_pred[index][1])


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)



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("tpr")
plt.ylabel("fpr")
plt.title("ROC PLOTS for train,test")
plt.grid()
plt.show()
```





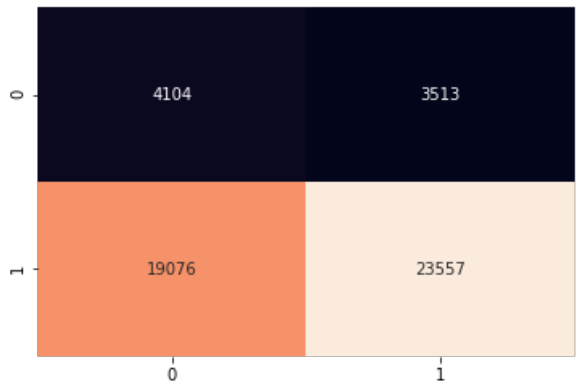
In [166]:

```
print("="*100)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred_prob, best_t)),annot = True,
fmt = "d", cbar=False)
```

```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.29771278421829506 for threshold 0.492
Train confusion matrix
```

Out[166]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c397342358>
```



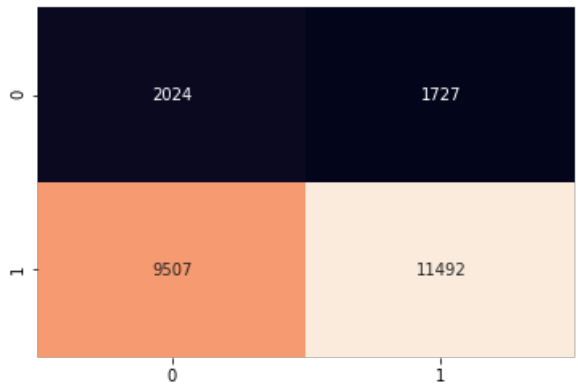
In [167]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred_prob, best_t)), annot = True,
fmt = "d", cbar=False)
```

```
Test confusion matrix
```

Out[167]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c39734b550>
```



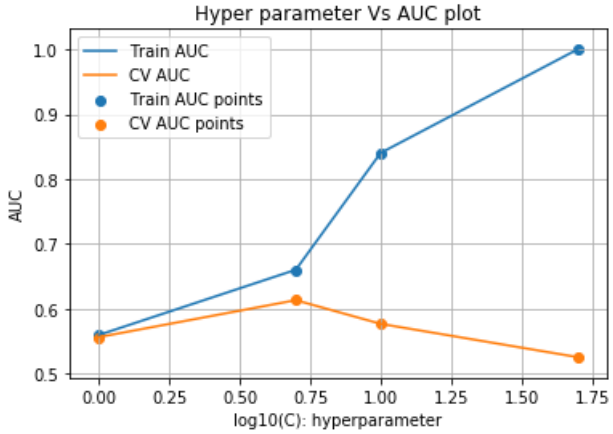
Set 2: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V) + Sentiment Scores

In [87]:

```
X_train_set_2 =
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\

x_train_grade_one_hot,x_train_teacher_number,x_train_price,x_train_quantity,x_train_weighted_w2v_ti
tle,x_train_weighted_w2v_essay,\
                         X_train['neg'].values.reshape(-1,1),X_train['neu'].values.reshape(-1,1),X_
rain['pos'].values.reshape(-1,1),X_train['compound'].values.reshape(-1,1))).tocsr()
X_test_set_2 =
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                        x_test_grade_one_hot,x_test_teacher_number,x_test_price,x_test_quantity,x_t
est_weighted_w2v_title,x_test_weighted_w2v_essay,\
                        X_test['neg'].values.reshape(-1,1),X_test['neu'].values.reshape(-1,1),X_tes
['pos'].values.reshape(-1,1),X_test['compound'].values.reshape(-1,1))).tocsr()
```

In [98]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

DT_ = DecisionTreeClassifier(class_weight = "balanced")
parameters = {'max_depth':[1,5,10,50]}
clf = RandomizedSearchCV(DT_, parameters,n_iter = 4, scoring='roc_auc')
clf.fit(X_train_set_2, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_max_depth'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
max_depth_ =  results['param_max_depth'].apply(lambda x: math.log10(x))

plt.plot(max_depth_, train_auc, label='Train AUC')

plt.plot(max_depth_, cv_auc, label='CV AUC')

plt.scatter(max_depth_, train_auc, label='Train AUC points')

plt.scatter(max_depth_, cv_auc, label='CV AUC points')



plt.legend()
plt.xlabel("log10(C): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



Out[98]:

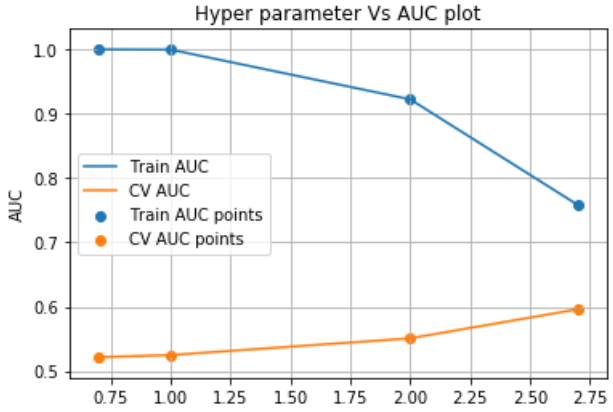
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params	split0_test_score	split1
0	4.481789	0.623423	0.417473	0.071193	1	{'max_depth': 1}	0.561054	0.552
1	23.182604	0.313140	0.379841	0.026202	5	{'max_depth': 5}	0.616883	0.614
2	54.990560	3.132931	0.385835	0.048892	10	{'max_depth': 10}	0.572849	0.578
3	130.046158	1.458867	0.437501	0.038270	50	{'max_depth': 50}	0.524337	0.531

In [92]:

```
# From the AUC plot, we find that the best value for "max_depth" - for the DecisionTreeClassfier i
s 5
best_max_depth = 5
```

In [100]:

```
DT_ = DecisionTreeClassifier(class_weight = "balanced")
parameters = {'min_samples_split':[5,10,100,500]}
clf = RandomizedSearchCV(DT_, parameters,n_iter = 4, scoring='roc_auc')
clf.fit(X_train_set_2, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_min_samples_split'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
min_samples_split_ =  results['param_min_samples_split'].apply(lambda x: math.log10(x))

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("log10(C): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results.head()
```



log10(C): hyperparameter

Out[100]:

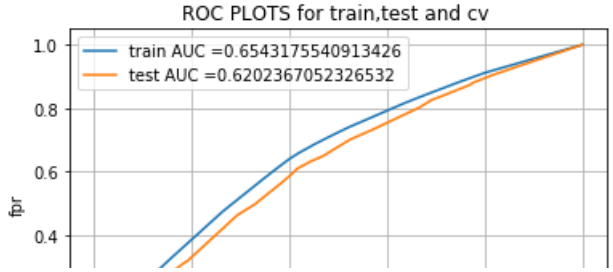
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_min_samples_split	params	split0_tes
0	152.174489	3.603958	0.458848	0.044371	5	{'min_samples_split': 5}	0.523728
1	158.490840	7.177979	0.458237	0.053756	10	{'min_samples_split': 10}	0.523984
2	148.489609	16.852794	0.531154	0.040452	100	{'min_samples_split': 100}	0.547472
3	72.619644	3.664973	0.546881	0.022095	500	{'min_samples_split': 500}	0.602943

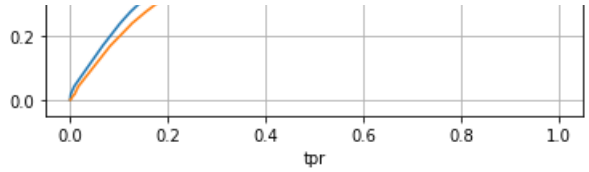
In [91]:

```
# min_samples_split from the graph above is 500
best_min_samples_split = 500
```

In [93]:

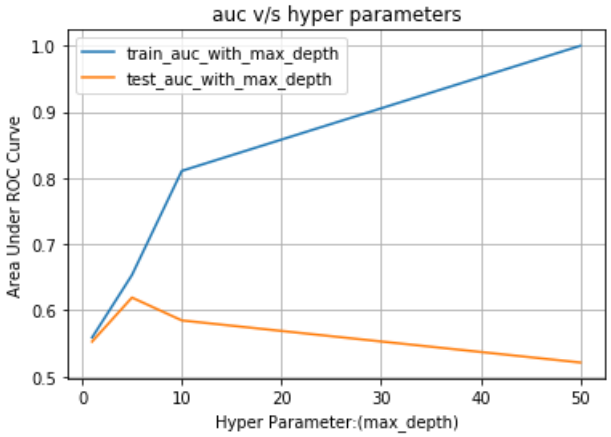
```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


DT_ = DecisionTreeClassifier(max_depth=best_max_depth, min_samples_split = best_min_samples_split
,class_weight = "balanced")
DT_.fit(X_train_set_2, y_train)


y_train_pred = DT_.predict_proba(X_train_set_2)
y_test_pred =  DT_.predict_proba(X_test_set_2)

y_train_pred_prob = []
y_test_pred_prob = []

for index in range(len(y_train_pred)):
    y_train_pred_prob.append(y_train_pred[index][1])

for index in range(len(y_test_pred)):
    y_test_pred_prob.append(y_test_pred[index][1])


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)



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("tpr")
plt.ylabel("fpr")
plt.title("ROC PLOTS for train,test and cv ")
plt.grid()
plt.show()
```





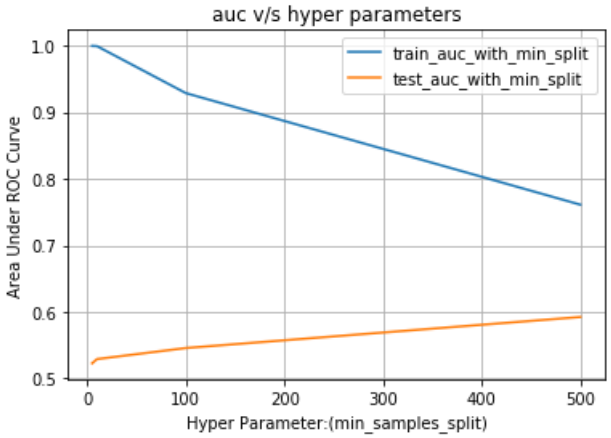
In [103]:

```
depth_list = [1,5,10,50]
compute_auc_with_hyper_para_depth(X_train_set_2,y_train,X_test_set_2, y_test, depth_list)
```



In [88]:

```
split_list = [5,10,100,500]
compute_auc_with_hyper_para_split(X_train_set_2,y_train,X_test_set_2, y_test, split_list)
```



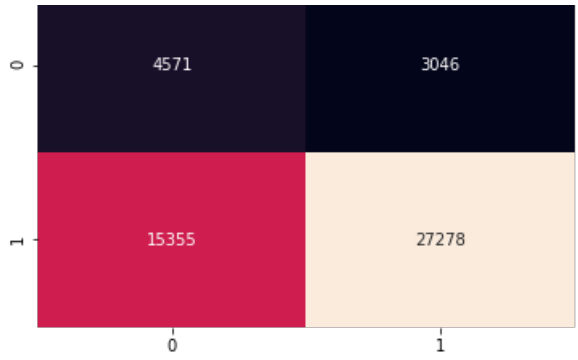
In [96]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred_prob, best_t)),annot = True,
fmt = "d", cbar=False)
```

```
====================================================================================================

the maximum value of tpr*(1-fpr) 0.3839669964571573 for threshold 0.536
Train confusion matrix
```

Out[96]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c397974358>
```



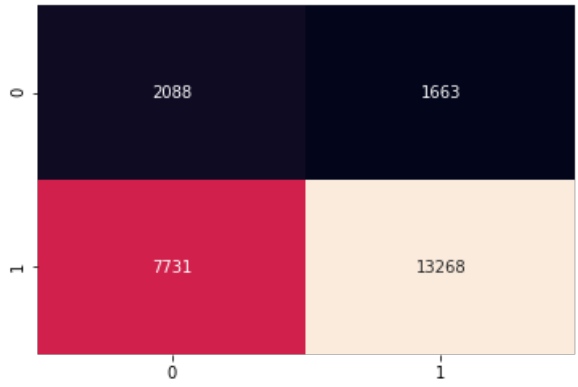
In [97]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred_prob, best_t)), annot = True,
fmt = "d", cbar=False)
```

```
Test confusion matrix
```

Out[97]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c397981438>
```



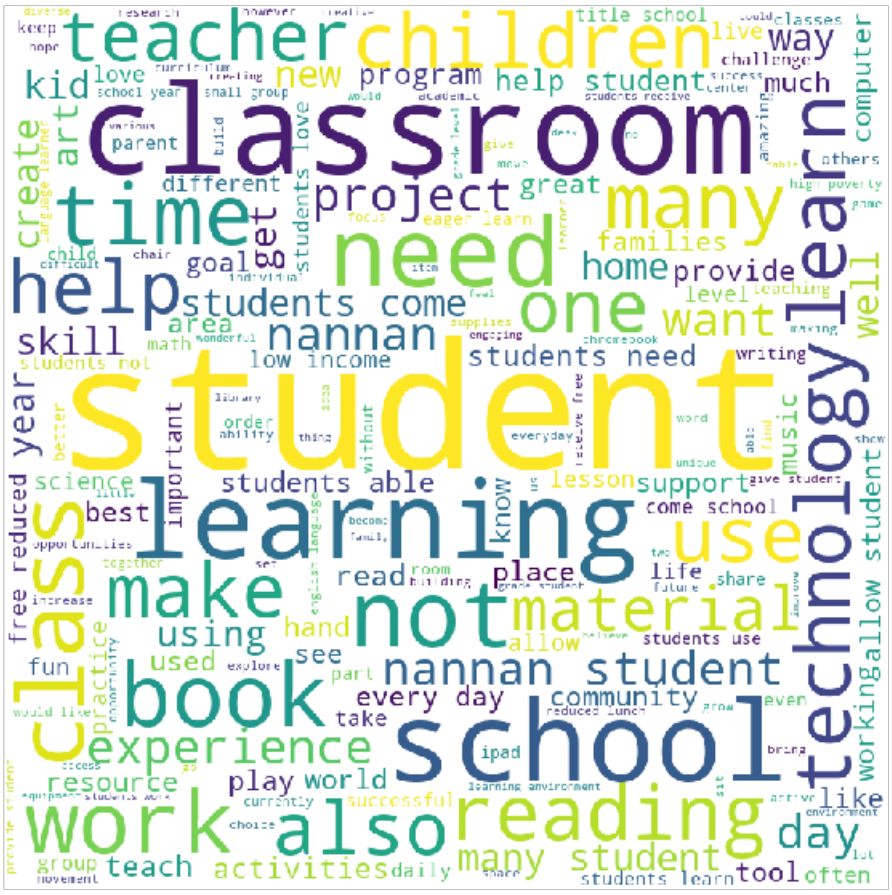
False positive points and WordCloud, box plots

wordcloud

In [134]:

```
# y_test we have as y_test from train_test_split
# https://www.geeksforgeeks.org/generating-word-cloud-python/
from wordcloud import WordCloud
y_actual = y_test
# y_predicted would be
y_pred = predict_with_best_t(y_test_pred_prob, best_t)
essay_words = " "
essay_ = []
for i in range(len(y_pred)):
    if y_pred[i]==1 and y_actual[i]!=y_pred[i]:
            essay_.append(x_train_preprocessed_essay[i])
for essay in essay_:
    essay_words += essay + " "
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(essay_words)
# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
```

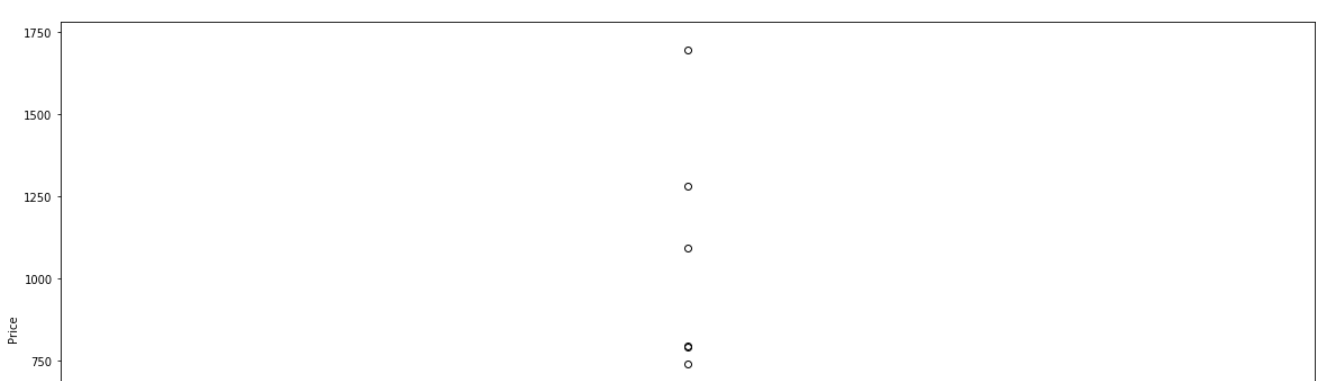
```
plt.show()
```

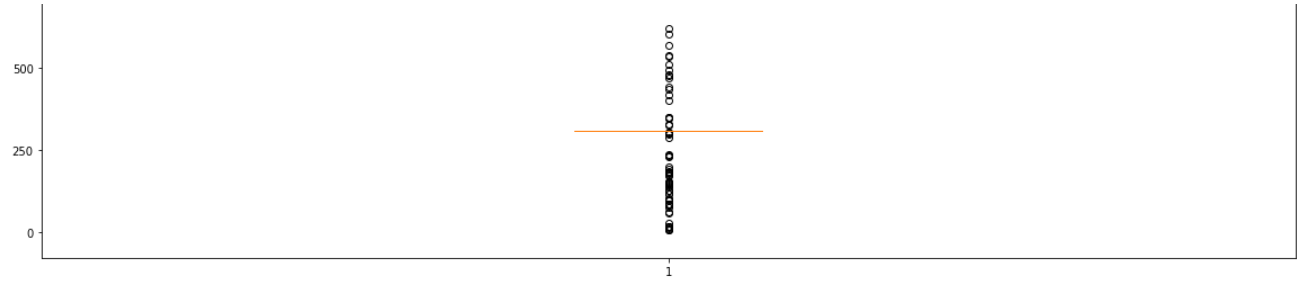


boxplot

In [143]:

```
y_actual = y_test
# y_predicted would be
y_pred = predict_with_best_t(y_test_pred_prob, best_t)
prices = []
for i in range(len(y_pred)):
    if y_pred[i]==1 and y_actual[i]!=y_pred[i]:
            prices.append(X_train['price'].iloc[i])
fig = plt.figure(figsize=(20,10))
fig.suptitle('Price from false positive data points', fontsize=14, fontweight='bold')

ax = fig.add_subplot(111)
ax.boxplot(prices)
ax.set_ylabel('Price')

plt.show()
```



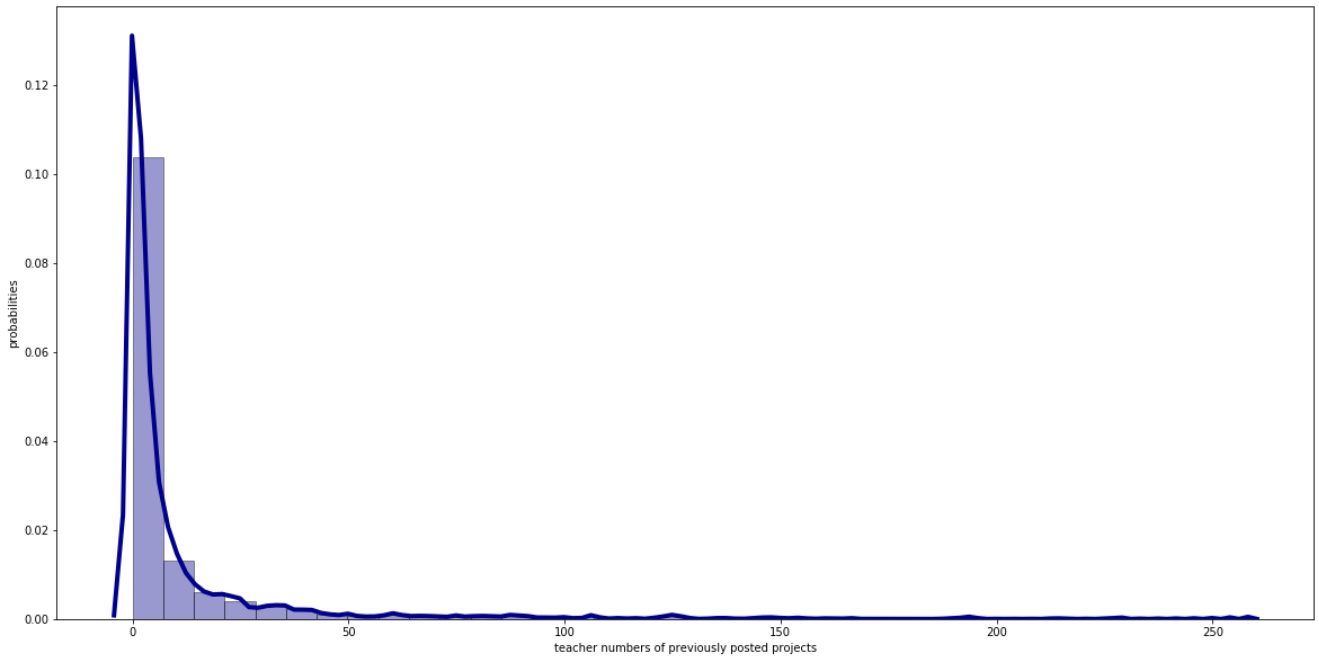


PDF for teacher_number_of_previously_posted_projects

In [146]:

```
y_actual = y_test
# y_predicted would be
y_pred = predict_with_best_t(y_test_pred_prob, best_t)
teacher_number_ = []
for i in range(len(y_pred)):
    if y_pred[i]==1 and y_actual[i]!=y_pred[i]:
            teacher_number_.append(X_train['teacher_number_of_previously_posted_projects'].iloc[i])
fig = plt.figure(figsize=(20,10))
ax = sns.distplot(teacher_number_, hist=True, kde=True,
             bins=int(180/5), color = 'darkblue',
             hist_kws={'edgecolor':'black'},
             kde_kws={'linewidth': 4})
ax.set(xlabel = 'teacher numbers of previously posted projects',ylabel = 'probabilities')
plt.show()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: UserWarning:

The 'normed' kwarg is deprecated, and has been replaced by the 'density' kwarg.
```



In [89]:

```
depth = [1,5,10,50]
split = [5,10,100,500]
plot_3d_plot(X_train_set_2, y_train, X_test_set_2, y_test, depth, split)
```

In [171]:

```
from prettytable import PrettyTable
summarizer = PrettyTable()
summarizer.field_names = ["vectorizer","Model","Hyper Parameter (max_depth)","Hyper Parameter
(min_split)","AUC test"]
summarizer.add_row(["TFIDF","Decision Tree Classifier","10","500","0.60"])
summarizer.add_row(["TFIDF W2V","Decision Tree Classifier","5","500","0.62"])
summarizer.add_row(["Feature Importance","Logistic Regression","0.0001 (reg. parameter)","NA","0.5
6"])
print(summarizer)
```

```
+--------------------+--------------------------+------------------------------+--------------------
-----+----------+
|     vectorizer     |          Model           | Hyper Parameter (max_depth) | Hyper Parameter (mi
_split) | AUC test |
+--------------------+--------------------------+------------------------------+--------------------
-----+----------+
|       TFIDF        | Decision Tree Classifier |              10              |             500
|   0.60   |
|     TFIDF W2V      | Decision Tree Classifier |              5               |             500
|   0.62   |
| Feature Importance |   Logistic Regression    |   0.0001 (reg. parameter)    |              NA
|   0.56   |
+--------------------+--------------------------+------------------------------+--------------------
-----+----------+
```