

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



## ▼ READING DATA

```
dft = pd.read_csv('train_data.csv', nrows=60000)
dfr = pd.read_csv('resources.csv')

print("Number of data points in train data", dft.shape)
print("Number of data points in resources", dfr.shape)
```

```
print('---*50)
print("The attributes of data :", dft.columns.values)

❸ Number of data points in train data (60000, 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']
```

```
print("Number of data points in train data", dfr.shape)
print(dfr.columns.values)
dfr.head(2)
```

❸ Number of data points in train data (1541272, 4)  
 ['id' 'description' 'quantity' 'price']

	<b>id</b>	<b>description</b>	<b>quantity</b>	<b>price</b>
<b>0</b>	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
<b>1</b>	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(dft.columns)]
```

```
#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
dft['Date'] = pd.to_datetime(dft['project_submitted_datetime'])
dft.drop('project_submitted_datetime', axis=1, inplace=True)
dft.sort_values(by=[ 'Date'], inplace=True)
```

```
# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
dft = dft[cols]
```

```
dft.head(2)
```



## ▼ TEXT PROCESSING

55660 8393 p205479 2bf07ba08945e5d8b2a3f269b2b3cfe5 Mrs. CA

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

```
dft.head(2)
```

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
--	---------------	----	------------	----------------	--------------

55660 8393 p205479 2bf07ba08945e5d8b2a3f269b2b3cfe5 Mrs. CA

51140 74477 p189804 4a97f3a390bfe21b99cf5e2b81981c73 Mrs. CA

```
# 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
```

```
# 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", "yo
    "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
    'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they',
    'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll"
    'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'h
    'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'unt
    'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'dur
    'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
    'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'bo
    'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'ver
    's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd
    've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'does
    "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
    "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
    'won', "won't", 'wouldn', "wouldn't"]
```

## ▼ Preprocessing of project\_subject\_categories

```
catogories = list(dft['project_subject_categories'].values)

cat_list = []
for i in catogories:
    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", "Ca
        if 'The' in j.split(): # this will split each of the category based on space "Math &
            j=j.replace('The','') # if we have the words "The" we are going to replace it wit
        j = j.replace(' ','') # we are placing all the ' '(space) with ''(empty) ex:"Math & S
        temp+=j.strip()+" "# abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

dft['clean_categories'] = cat_list
dft.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in dft['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

## ▼ Preprocessing of project\_subject\_subcategories

```

sub_cat_list = []
for i in sub_cat_list:
    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", "Ca
        if 'The' in j.split():# this will split each of the category based on space "Math &
            j=j.replace('The','') # if we have the words "The" we are going to replace it wit
        j = j.replace(' ','') # we are placing all the ' '(space) with ''(empty) ex:"Math & S
        temp +=j.strip()+" "# abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

```

```

dft['clean_subcategories'] = sub_cat_list
dft.drop(['project_subject_subcategories'], axis=1, inplace=True)

```

```

# count of all the words in corpus python:
#https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in dft['clean_subcategories'].values:
    my_counter.update(word.split())

```

```

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))

```

```

# we have to remove the grades from every row
print(dft['project_grade_category'][:20])

```

55660	Grades PreK-2
51140	Grades PreK-2
473	Grades PreK-2
41558	Grades 3-5
29891	Grades 3-5
23374	Grades PreK-2
49228	Grades PreK-2
7176	Grades PreK-2
35006	Grades 3-5
5145	Grades 3-5
48237	Grades 9-12
52282	Grades 9-12
46375	Grades 3-5
36468	Grades PreK-2
36358	Grades PreK-2
39438	Grades PreK-2
2521	Grades PreK-2
58794	Grades PreK-2
40180	Grades PreK-2
53562	Grades 9-12

Name: project\_grade\_category, dtype: object

```

d= list(dft['project_grade_category'].values)
# remove special characters from list of strings python:
# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

grade_cat_list = []
for i in d:
    # consider we have text like this:
    for j in i.split(' '): # # split by space
        j=j.replace('Grades','')# clean grades from the row
    grade_cat_list.append(j.strip())

dft['clean_grade'] = grade_cat_list
dft.drop(['project_grade_category'], axis=1, inplace=True)

my_counter = Counter()
for word in dft['clean_grade'].values:
    my_counter.update(word.split())
project_grade_category_dict= dict(my_counter)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lam

```

## Preparing data for the models

### ▼ Test - Train Split

```

# train test split

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(dft, dft['project_is_approved'], stratify=y_train)
X_cv, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_cv)

X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, stratify= y_train, test_size=0.33, stratify=y_cv)

print(y_train.value_counts())
print(y_test.value_counts())
print(y_cv.value_counts())

```

 1 15295  
 0 2750  
 Name: project\_is\_approved, dtype: int64  
 1 16782  
 0 3018  
 Name: project\_is\_approved, dtype: int64  
 1 7535  
 0 1354  
 Name: project\_is\_approved, dtype: int64

```
#dropping the y labels
https://stackoverflow.com/questions/13411544/delete-column-from-pandas-dataframe-by-column-name

X_train.drop(["project_is_approved"], axis = 1, inplace = True)

X_test.drop(["project_is_approved"], axis = 1, inplace = True)

X_cv.drop(["project_is_approved"], axis = 1, inplace = True)
```

## ▼ Text preprocessing

```
#Processing for essay
# Combining all the above students
from tqdm import tqdm
preprocessed_essays_train = []
# tqdm is for printing the status bar
for sentence in tqdm(X_train['essay'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\r', ' ')
    sent = sent.replace('\n', ' ')
    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_essays_train.append(sent.lower().strip())
```

100% |  | 18045/

```
#Processing for essay
# Combining all the above students
from tqdm import tqdm
preprocessed_essays_test = []
# tqdm is for printing the status bar
for sentence in tqdm(X_test['essay'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\r', ' ')
    sent = sent.replace('\n', ' ')
    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_essays_test.append(sent.lower().strip())
```

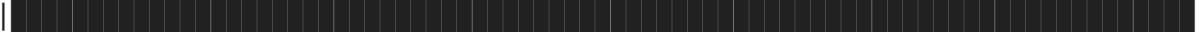
100% |  | 19800/

#Processing for essay

```

# Combining all the above students
from tqdm import tqdm
preprocessed_essays_cv = []
# tqdm is for printing the status bar
for sentance in tqdm(X_cv['essay'].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_essays_cv.append(sent.lower().strip())

```

100% |  | 8889

```

#Proprocessing for essay
# Combining all the above students
from tqdm import tqdm
preprocessed_titles_cv = []
# tqdm is for printing the status bar
for sentance in tqdm(X_cv['project_title'].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_titles_cv.append(sent.lower().strip())

```

100% |  | 8889/8

```

#Proprocessing for essay
# Combining all the above students
from tqdm import tqdm
preprocessed_titles_train = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['project_title'].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_titles_train.append(sent.lower().strip())

```



100%

| 18045/18

```
#Preprocessing for essay
# Combining all the above students
from tqdm import tqdm
preprocessed_titles_test = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['project_title'].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_titles_test.append(sent.lower().strip())
```



100%

| 19800/19

## Encoding numerical, Categorical features

### ▼ vectorize categorical data

```
#project_subject_categories convert categorical to vectors

# convert train, cv and test data of clean_categories into vectors
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer1 = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True)
vectorizer1.fit(X_train['clean_categories'].values)
# firstly convert fit the train data into the vectorizer then it learns the vocabulary
# we use the fitted CountVectorizer to convert the text to vector
X_train_cat = vectorizer1.transform(X_train['clean_categories'].values)
X_cv_cat = vectorizer1.transform(X_cv['clean_categories'].values)
X_test_cat = vectorizer1.transform(X_test['clean_categories'].values)
print(vectorizer1.get_feature_names())

f1=vectorizer1.get_feature_names()
print("After vectorizations")
print(X_train_cat.shape, y_train.shape)
print(X_cv_cat.shape, y_cv.shape)
print(X_test_cat.shape, y_test.shape)
print("=*100")
```

After vectorizations  
 (18045, 9) (18045,)  
 (8889, 9) (8889,)  
 (19800, 9) (19800,)  
=====

```
##project_subject_subcategories convert categorical to vectors
# convert train, cv and test data of clean_categories into vectors
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer2 = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, b
vectorizer2.fit(X_train['clean_subcategories'].values)
# firstly convert fit the train data into the vectorizer then it learns the vocabulary
# we use the fitted CountVectorizer to convert the text to vector
X_train_subcat = vectorizer2.transform(X_train['clean_subcategories'].values)
X_cv_subcat = vectorizer2.transform(X_cv['clean_subcategories'].values)
X_test_subcat = vectorizer2.transform(X_test['clean_subcategories'].values)
print(vectorizer2.get_feature_names())
```

['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurric

```
print("After vectorizations")
print(X_train_subcat.shape, y_train.shape)
print(X_cv_subcat.shape, y_cv.shape)
print(X_test_subcat.shape, y_test.shape)
print("=*100)
```

After vectorizations  
 (18045, 30) (18045,)  
 (8889, 30) (8889,)  
 (19800, 30) (19800,)  
=====

```
# school_state convert categorical to vectors
# now time to count the each words
from collections import Counter
my_counter = Counter()
for word in dft['school_state'].values:
    my_counter.update(word.split())# count the words
school_state_dict = dict(my_counter)# store in dictionary
sorted_school_state_dict = dict(sorted(school_state_dict.items(), key=lambda kv: kv[1]))
print(sorted_school_state_dict)
```

{'VT': 40, 'WY': 58, 'ND': 78, 'MT': 120, 'RI': 148, 'NH': 175, 'NE': 176, 'SD': 177, 'D

```
# convert train, cv and test data of clean_categories into vectors
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
```

```
vectorizer3 = CountVectorizer(vocabulary=list(sorted_school_state_dict.keys()), lowercase=False)
vectorizer3.fit(dft['school_state'].values)
# firstly convert fit the train data into the vector then it learn the vocabulary
# we use the fitted CountVectorizer to convert the text to vector
X_train_school_state = vectorizer3.transform(X_train['school_state'].values)
X_cv_school_state = vectorizer3.transform(X_cv['school_state'].values)
X_test_school_state = vectorizer3.transform(X_test['school_state'].values)
print(vectorizer3.get_feature_names())
```

👤 ['VT', 'WY', 'ND', 'MT', 'RI', 'NH', 'NE', 'SD', 'DE', 'AK', 'WV', 'HI', 'ME', 'DC', 'NM']

```
print("After vectorizations")
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("=*100)
```

👤 After vectorizations  
 (18045, 51) (18045,)  
 (8889, 51) (8889,)  
 (19800, 51) (19800,)

---

```
#project_grade_category categorical to vectors
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-sp
dft['clean_grade']=dft['clean_grade'].fillna("")# fill the null values with space
# convert train, cv and test data of clean_categories into vectors
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer4 = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()), lowercase=False)
vectorizer4.fit(dft['clean_grade'].values)
# firstly convert fit the train data into the vector then it learn the vocabulary
# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade_category = vectorizer4.transform(X_train['clean_grade'].values)
X_cv_project_grade_category = vectorizer4.transform(X_cv['clean_grade'].values)
X_test_project_grade_category = vectorizer4.transform(X_test['clean_grade'].values)
print(vectorizer4.get_feature_names())
```

👤 ['9-12', '6-8', '3-5', 'PreK-2']

```
print("After vectorizations")
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("=*100)
```

👤

After vectorizations

```
(18045, 4) (18045, )
(8889, 4) (8889, )
(19800, 4) (19800, )
```

---

```
#teacher_prefix categorical to vectors
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-sp
dft['teacher_prefix']=dft['teacher_prefix'].fillna(" ")# filll the null valueswith space
my_counter = Counter()
for word in dft['teacher_prefix'].values:
    my_counter.update(word.split())
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
teacher_cat_dict = dict(my_counter)
sorted_teacher_prefix_dict = dict(sorted(teacher_cat_dict.items(), key=lambda kv: kv[1]))
```

# convert train, cv and test data of clean\_categories into vectors

# we use count vectorizer to convert the values into one

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer5 = CountVectorizer(vocabulary=list(sorted_teacher_prefix_dict.keys()), lowercase=False)
vectorizer5.fit(dft['teacher_prefix'].values.astype('U'))
# firstly convert fit the train data into the vectorizer
# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix = vectorizer5.transform(X_train['teacher_prefix'].values.astype('U'))
X_cv_teacher_prefix = vectorizer5.transform(X_cv['teacher_prefix'].values.astype('U'))
X_test_teacher_prefix = vectorizer5.transform(X_test['teacher_prefix'].values.astype('U'))
print(vectorizer5.get_feature_names())
```

 ['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs. ']

```
print("After vectorizations")
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("=*100)
```

 After vectorizations

```
(18045, 5) (18045, )
(8889, 5) (8889, )
(19800, 5) (19800, )
```

---

## ▼ Encoding essay, and Project\_title

```
#bow featurization essay
```

```
X_train_essay=preprocessed_essays_train
X_cv=preprocessed_essays_cv
```

```
^_cv_essay-pi_cv preprocess_cv_essays_cv
X_test_essay=preprocessed_essays_test
```

```
X_train_title=preprocessed_titles_train
X_cv_title=preprocessed_titles_cv
X_test_title=preprocessed_titles_test
# We are considering only the words which appeared in at least 10 documents(rows or projects)
vectorizer6 = CountVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))
vectorizer6.fit(X_train_essay)# that is learned from trained data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer6.transform(X_train_essay)
X_cv_bow = vectorizer6.transform(X_cv_essay)
X_test_bow = vectorizer6.transform(X_test_essay)

#bow featurization title
vectorizer7 = CountVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))
vectorizer7.fit(X_train_title)# that is learned from trained data
# we use the fitted CountVectorizer to convert the text to vector
X_train_bow_title = vectorizer7.transform(X_train_title)
X_cv_bow_title= vectorizer7.transform(X_cv_title)
X_test_bow_title = vectorizer7.transform(X_test_title)
print("After vectorizations")
print(X_train_bow_title.shape, y_train.shape)
print(X_cv_bow_title.shape, y_cv.shape)
print(X_test_bow_title.shape, y_test.shape)
print("=*100)
```



After vectorizations  
 (18045, 1327) (18045,)  
 (8889, 1327) (8889,)  
 (19800, 1327) (19800,)  
 =====

## ▼ TfIdf featurization

```
#for titles
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects)
vectorizer8 = TfidfVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))
vectorizer8.fit(X_train_title)# that is learned from trained data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tf_title = vectorizer8.transform(X_train_title)
X_cv_tf_title= vectorizer8.transform(X_cv_title)
X_test_tf_title = vectorizer8.transform(X_test_title)
print("After vectorizations")
print('X train tf title shape v train shape')
```

```
print(X_cv_tf_title.shape, y_cv.shape)
print(X_test_tf_title.shape, y_test.shape)
print("=*100)
```



After vectorizations  
 (18045, 1327) (18045,)  
 (8889, 1327) (8889,)  
 (19800, 1327) (19800,)

```
=====
```

```
#for essay
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects)
vectorizer9 = TfidfVectorizer(min_df=10,max_features=5000,ngram_range=(1, 2))
vectorizer9.fit(X_train_essay)# that is learned from trained data
# we use the fitted CountVectorizer to convert the text to vector
X_train_tf_essay = vectorizer9.transform(X_train_essay)
X_cv_tf_essay= vectorizer9.transform(X_cv_essay)
X_test_tf_essay = vectorizer9.transform(X_test_essay)
print("After vectorizations")
print(X_train_tf_essay.shape, y_train.shape)
print(X_cv_tf_essay.shape, y_cv.shape)
print(X_test_tf_essay.shape, y_test.shape)
print("=*100)
```



After vectorizations  
 (18045, 5000) (18045,)  
 (8889, 5000) (8889,)  
 (19800, 5000) (19800,)

```
=====
```

## ▼ Using Pretrained Models : AVG W2V

```
# 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')
```

 Loading Glove Model  
1917495it [09:14, 3459.20it/s]  
Done. 1917495 words loaded!

```
glove_words = set(model.keys())
```

```
#for essay  
# average Word2Vec  
# compute average word2vec for each review.  
def func(wordlist):
```

```
train_avg_w2v_vectors = [] # the avg-w2v for each sentence/review is stored in this list  
for sentence in tqdm(wordlist): # for each review/sentence  
    vector = np.zeros(300) # as word vectors are of zero length # we are taking the 300dimens  
    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  
    train_avg_w2v_vectors.append(vector)
```

```
print(len(train_avg_w2v_vectors))  
print(len(train_avg_w2v_vectors[0]))  
return train_avg_w2v_vectors
```

```
train_avg_w2v_vectors=func(preprocessed_essays_train)  
test_avg_w2v_vectors=func(preprocessed_essays_test)  
cv_avg_w2v_vectors=func(preprocessed_essays_cv)  
#for titles  
cv_avg_w2v_vectors_title=func(preprocessed_titles_cv)  
test_avg_w2v_vectors_title=func(preprocessed_titles_test)  
train_avg_w2v_vectors_title=func(preprocessed_titles_train)
```



```

100%|██████████| 18045/1
18045
300
100%|██████████| 19800/1
19800
300
100%|██████████| 8889/1
8889
300
100%|██████████| 8889/8
8889
300
100%|██████████| 19800/19
19800
300
100%|██████████| 18045/18
18045
300

```

## ▼ Using Pretrained Models: TFIDF weighted W2V

```

tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays_train)
# 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())

# average Word2Vec
# compute average word2vec for each review.
def tf_idf_done(word_list):
    train_title_tfidf_w2v_vectors = [] # the avg-w2v for each sentence/review is stored in t
    for sentence in tqdm(word_list): # 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():#.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words):
                #vec = model.wv[word]
                vec = model[word] # getting the vector for each word
        # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/
        tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())))
        vector += (vec * tf_idf) # calculating tfidf weighted w2v
        tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        train_title_tfidf_w2v_vectors.append(vector)
    print(len(train_title_tfidf_w2v_vectors))
    print(len(train_title_tfidf_w2v_vectors[0]))
    return train_title_tfidf_w2v_vectors

```

```
train_tfidf_w2v_vectors=tf_idf_done(preprocessed_essays_train)
test_tfidf_w2v_vectors=tf_idf_done(preprocessed_essays_test)
cv_tfidf_w2v_vectors=tf_idf_done(preprocessed_essays_cv)
```

100% |  | 18045  
 18045  
 300  
 100% |  | 19800/  
 19800  
 300  
 100% |  | 8889  
 8889  
 300

```
train_title_tfidf_w2v_vectors=tf_idf_done(preprocessed_titles_train)
test_title_tfidf_w2v_vectors=tf_idf_done(preprocessed_titles_test)
cv_title_tfidf_w2v_vectors=tf_idf_done(preprocessed_titles_cv)
```

100% |  | 18045/18  
 18045  
 300  
 100% |  | 19800/19  
 19800  
 300  
 100% |  | 8889/8  
 8889  
 300

## ‐ Vectorizing Numerical features

```
price_data = dfr.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
dft = pd.merge(dft, price_data, on='id', how='left')
print(price_data.head(2))
# we also have to do this in train, test and cv
# so also merge the resource data with the train, cv and test
X_train = pd.merge(X_train, price_data, on = "id", how = "left")
#print(x_train.columns)
X_test = pd.merge(X_test, price_data, on = "id", how = "left")
X_cv = pd.merge(X_cv, price_data, on = "id", how = "left")
```

100% |  | 18045  
 18045  
 300  
 100% |  | 19800/  
 19800  
 300  
 100% |  | 8889/8  
 8889  
 300

```
#standardization
# check this one: https://www.youtube.com/watch?v=0H0q0cIn3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler
```

```

from sklearn.preprocessing import MinMaxScaler
from sklearn import preprocessing

price_scalar = StandardScaler()
price_scalar.fit(X_train['price'].values.reshape(-1,1)) # finding the mean and standard deviation
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")
# Now standardize the data with above mean and variance.
train_price_standar = price_scalar.transform(X_train['price'].values.reshape(-1, 1))
# Now standardize the data with above mean and variance.
test_price_standar = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
# Now standardize the data with above mean and variance.
cv_price_standar = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))

# previous_year_projects
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")
# Now standardize the data with above mean and variance.
train_prev_proj_standar = price_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
# Now standardize the data with above mean and variance.
test_prev_proj_standar = price_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
# Now standardize the data with above mean and variance.
cv_prev_proj_standar = price_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))

price_scalar.fit(X_train['quantity'].values.reshape(-1,1)) # finding the mean and standard deviation
#print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")
# Now standardize the data with above mean and variance.
train_qnty_standar = price_scalar.transform(X_train['quantity'].values.reshape(-1, 1))
# Now standardize the data with above mean and variance.
cv_qnty_standar = price_scalar.transform(X_cv['quantity'].values.reshape(-1, 1))
# Now standardize the data with above mean and variance.
test_qnty_standar = price_scalar.transform(X_test['quantity'].values.reshape(-1, 1))

```

## ▼ merging

```

from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matrix
X_set1_train = hstack((X_train_bow_title,X_train_bow,# all bows
                      X_train_teacher_prefix,X_train_cat,X_train_subcat
                      ,X_train_project_grade_category,X_train_school_state,
                      train_qnty_standar,train_price_standar,train_prev_proj_standar))
print(X_set1_train.shape, y_train.shape)

```

 (18045, 6429) (18045,)

```

from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matrix
https://colab.research.google.com/drive/1HntpTYRfAtFc9pp5itR0DrxnYQQoRRgx#scrollTo=WoapUtdOn4Yt&printMode=true

```

```
X_set1_cv = hstack((X_cv_bow_title,X_cv_bow,
                     X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                     X_cv_project_grade_category,X_cv_school_state,
                     cv_qnty_standar,cv_price_standar,cv_prev_proj_standar))
print(X_set1_cv.shape, y_cv.shape)
```

👤 (8889, 6429) (8889,)

```
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matirx
X_set1_test = hstack((X_test_bow_title,X_test_bow,
                      X_test_teacher_prefix,X_test_cat,X_test_subcat,
                      X_test_project_grade_category,X_test_school_state,
                      test_qnty_standar,test_price_standar,test_prev_proj_standar))
print(X_set1_test.shape, y_test.shape)
```

👤 (19800, 6429) (19800,)

```
xtr = X_set2_train.tocsr() # Here I have just applied kind of trail and logic. It was in coom
```

```
xtr
```

👤 <18045x6429 sparse matrix of type '<class 'numpy.float64'>'  
with 2359759 stored elements in Compressed Sparse Row format>

```
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :)
X_set2_train = hstack((X_train_tf_essay,X_train_tf_title,
                      X_train_teacher_prefix,X_train_cat,X_train_subcat,
                      X_train_project_grade_category,X_train_school_state,
                      train_qnty_standar,train_price_standar,train_prev_proj_standar))
print(X_set2_train.shape, y_train.shape)
```

👤 (18045, 6429) (18045,)

```
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :)
X_set2_cv = hstack((X_cv_tf_essay,X_cv_tf_title,
                     X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                     X_cv_project_grade_category,X_cv_school_state,
                     cv_qnty_standar,cv_price_standar,cv_prev_proj_standar))
print(X_set2_cv.shape, y_cv.shape)
```

👤 (8889, 6429) (8889,)

```
# You are getting error coomatrix which is not accessible. For this reason, you are getting s
# Overall, you want things in sparse form. I just checked the type of data and it is coomatri
```

```
# So we have converted coomatrix type to sparse type using csr
```

```
# Here is the coomatrix type.
```

```
#type(X_set2_test) # This is in coomatrix which we don't want and is not accessible.
```

```
xte = X_set2_test.tocsr() # We want in sparse type and so we are convertin it to sparse mat
type(xte)
```

```
#Instead of renamed everything just add an extension of .tocsr() wherever there is coomatrix
```

 `scipy.sparse.csr.csr_matrix`

```
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :)
X_set2_test = hstack((X_test_tf_essay,X_test_tf_title,
                      X_test_teacher_prefix,X_test_cat,X_test_subcat,
                      X_test_project_grade_category,X_test_school_state,
                      test_qty_standar,test_price_standar,test_prev_proj_standar)).tocsr()
print(X_set2_test.shape, y_test.shape)
```

 (19800, 6429) (19800,)

```
import numpy
s=numpy.array(train_avg_w2v_vectors)
print(X_train_project_grade_category.shape)
print(s.shape)
```

 (18045, 4)
 (18045, 300)

```
from scipy.sparse import hstack
import numpy
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :)
X_set3_train = hstack((numpy.array(train_avg_w2v_vectors),numpy.array(train_avg_w2v_vectors_t
                           X_train_teacher_prefix,X_train_cat,X_train_subcat,
                           X_train_project_grade_category,X_train_school_state))
print(X_set3_train.shape, y_train.shape)
```

 (18045, 702) (18045,)

```
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :)
X_set3_cv =hstack((cv_avg_w2v_vectors,cv_avg_w2v_vectors_title,cv_prev_proj_standar,cv_price_
                     X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                     X_cv_project_grade_category,X_cv_school_state))
print(X_set3_cv.shape, y_cv.shape)
```

 (8889, 702) (8889,)

```
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :
X_set3_test =hstack((test_avg_w2v_vectors,test_avg_w2v_vectors_title,test_prev_proj_standar,t
                     test_qty_standar,
                     X_test_teacher_prefix,X_test_cat,X_test_subcat,
                     X_test_project_grade_category,X_test_school_state))
print(X_set3_test.shape, y_test.shape)
```

 (19800, 702) (19800,)

```
import numpy
s=numpy.array(train_tfidf_w2v_vectors)
print(X_train_project_grade_category.shape)
print(s.shape)
```

 (18045, 4)
(18045, 300)

```
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :
X_set4_train =hstack((train_tfidf_w2v_vectors, train_title_tfidf_w2v_vectors,train_prev_proj_
                     train_price_standar,train_qty_standar,
                     X_train_teacher_prefix,X_train_cat,X_train_subcat,
                     X_train_project_grade_category,X_train_school_state))
```

```
print(X_set4_train.shape, y_train.shape)
```

 (18045, 702) (18045,)

```
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :
X_set4_cv =hstack((cv_tfidf_w2v_vectors, cv_title_tfidf_w2v_vectors, cv_prev_proj_standar,
                     cv_price_standar, cv_qty_standar,
                     X_cv_teacher_prefix,X_cv_cat,X_cv_subcat,
                     X_cv_project_grade_category,X_cv_school_state))
```

```
print(X_set4_cv.shape, y_cv.shape)
```

 (8889, 702) (8889,)

```
from scipy.sparse import hstack
# with the same hstack function we are concatenating a sparse matrix and a dense matirx :
X_set4_test = hstack((test_title_tfidf_w2v_vectors,test_tfidf_w2v_vectors,test_prev_proj_stan
                     X_test_project_grade_category,X_test_school_state))
print(X_set4_test.shape, y_test.shape)
```



(19800, 702) (19800, )

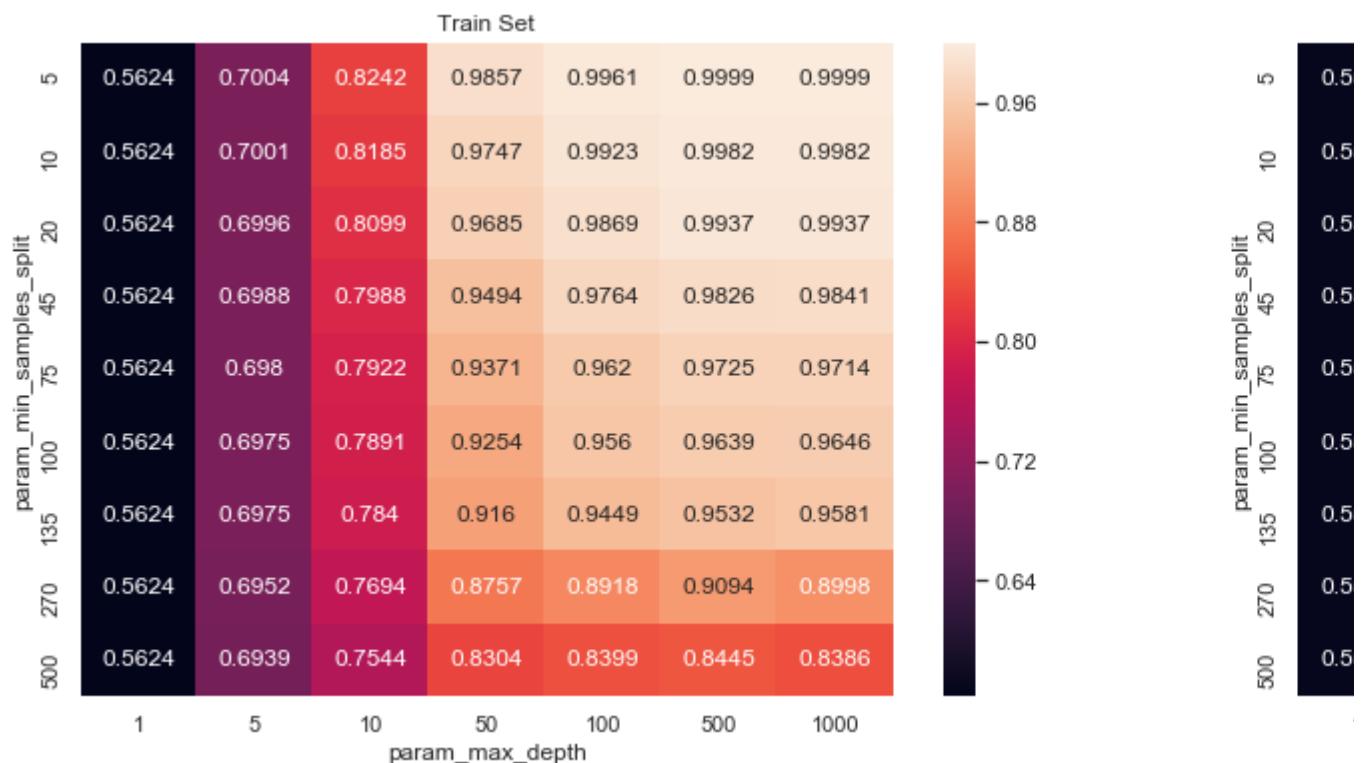
## ▼ Decision trees on BOW

```

from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
dt1 = DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50, 100, 500, 1000], 'min_samples_split': [5, 10, 20, 40, 75, 100, 135, 270, 500]}
clf1 = GridSearchCV(dt1, parameters, cv=3, scoring='roc_auc', return_train_score=True)
se1 = clf1.fit(X_set1_train, y_train)

import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf1.cv_results_).groupby(['param_min_samples_split', 'param_max_depth'])
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='%.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='%.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()

```



## ▼ Best Estimator and Best tune parameters

```

print(clf1.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf1.score(X_set1_train,y_train))
print(clf1.score(X_set1_test,y_test))

❸ DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=10,
                         max_features=None, max_leaf_nodes=None,
                         min_impurity_decrease=0.0, min_impurity_split=None,
                         min_samples_leaf=1, min_samples_split=500,
                         min_weight_fraction_leaf=0.0, presort=False,
                         random_state=None, splitter='best')

0.7511722190852625
0.6719906398813649

# Best tune parameters
best_tune_parameters=[{'max_depth':[10], 'min_samples_split':[500] } ]

clf1.get_params().keys()

❸ dict_keys(['cv', 'error_score', 'estimator__class_weight', 'estimator__criterion', 'esti

```

Fit Model to Hyper-Parameter Curve -> Best Max\_depth-> 1

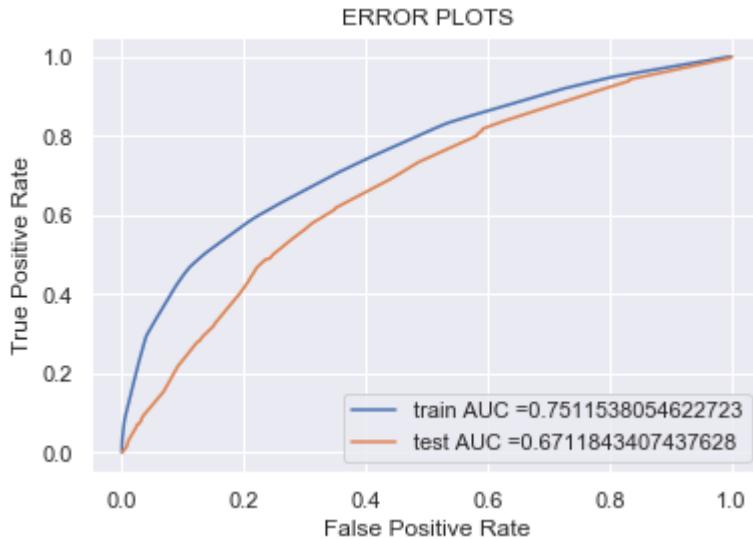
Min\_sample\_split-> 100

```

# https://scikitlearn.org/stable/modules/generated/sklearn.metrics.roc\_curve.html#sklearn.metrics.roc\_curve
from sklearn.metrics import roc_curve, auc
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve, auc
clf11= GridSearchCV( DecisionTreeClassifier(class_weight = 'balanced'),best_tune_parameters)
clfV1=DecisionTreeClassifier (class_weight = 'balanced',max_depth=10,min_samples_split=500)
clf11.fit(X_set1_train, y_train)
# for visualization
clfV1.fit(X_set1_train, y_train)
#https://scikitlearn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier.html#sklearn.linear\_model.SGDClassifier
y_train_pred1 = clf11.predict_proba(X_set1_train) [:,1]
y_test_pred1 = clf11.predict_proba(X_set1_test) [:,1]
train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)
plt.plot(train_fpr1, train_tpr1, label="train AUC =" +str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC =" +str(auc(test_fpr1, test_tpr1)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")

```

```
plt.grid(True)
plt.show()
```



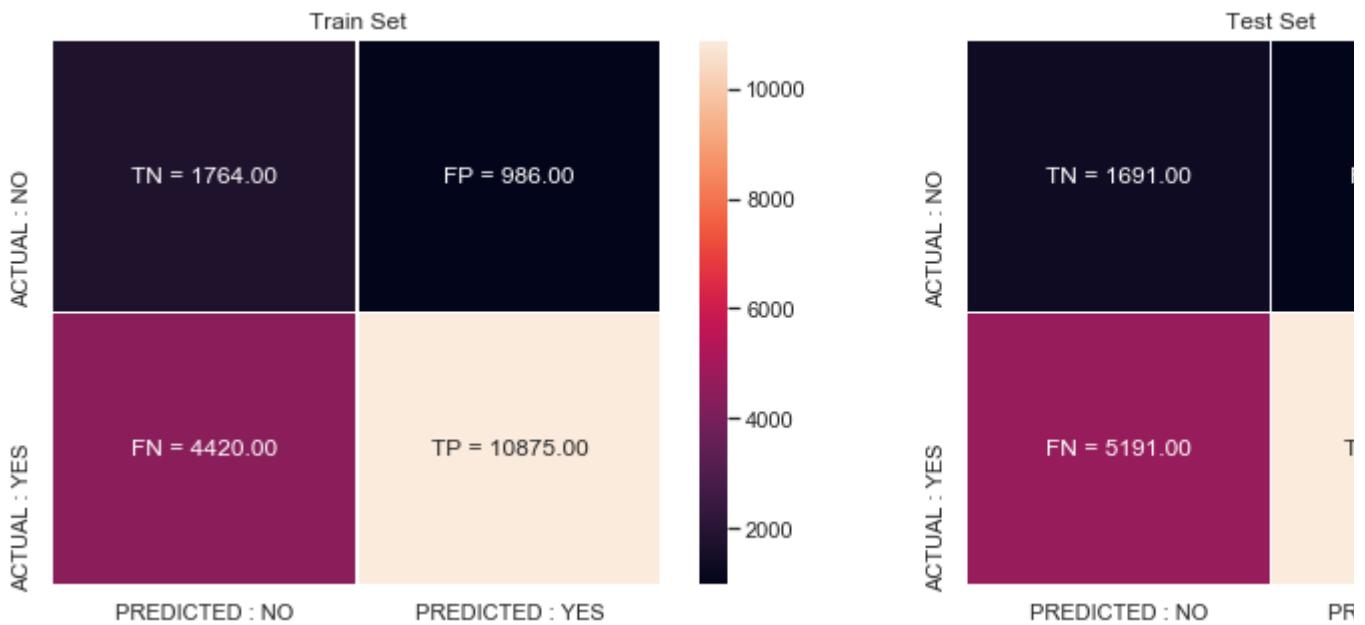
## ▼ Confusion Matrix

```
def predict(proba, threshold, fpr, tpr):
    t = threshold[np.argmax(fpr*(1-tpr))]
    print("the maximum value of tpr*(1-fpr)", np.round(max(tpr*(1-fpr)),2) , "for threshold",
    predictions = []
    global predictions1 # make it global
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1= predictions
    return predictions
```

```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()
con_m_train = confusion_matrix(y_train, predict(y_train_pred1, tr_thresholds1, train_fpr1, tr_t))
con_m_test = confusion_matrix(y_test, predict(y_test_pred1, te_thresholds1, test_fpr1, test_t))
key = (np.asarray([['TN','FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(15,5))
labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flat, con_m_train)]))
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flat, con_m_test)]))
sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], yticklabels=labels_train)
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], yticklabels=labels_test)
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
plt.show()
```



the maximum value of  $tpr * (1 - fpr)$  0.47 for threshold 0.44  
 the maximum value of  $tpr * (1 - fpr)$  0.4 for threshold 0.45



## ▼ Visualizing Decision Tree

```
#Feature aggregation
f1=vectorizer1.get_feature_names()
f2=vectorizer2.get_feature_names()
f3=vectorizer3.get_feature_names()
f4=vectorizer4.get_feature_names()
f5=vectorizer5.get_feature_names()
fb=vectorizer6.get_feature_names()
ft=vectorizer7.get_feature_names()
fb1=vectorizer8.get_feature_names()
ft1=vectorizer9.get_feature_names()

feature_agg_bow = f1 + f2 + f3 + f4 + f5 + fb + ft
feature_agg_tfidf = f1 + f2 + f3 + f4 + f5 + fb1 + ft1
# p is price, q is quantity, t is teacher previous year projects
feature_agg_bow.append('price')
feature_agg_tfidf.append('price')
feature_agg_bow.append('quantity')
feature_agg_tfidf.append('quantity')
feature_agg_bow.append('teacher_previous_projects')
feature_agg_tfidf.append('teacher_previous_projects')

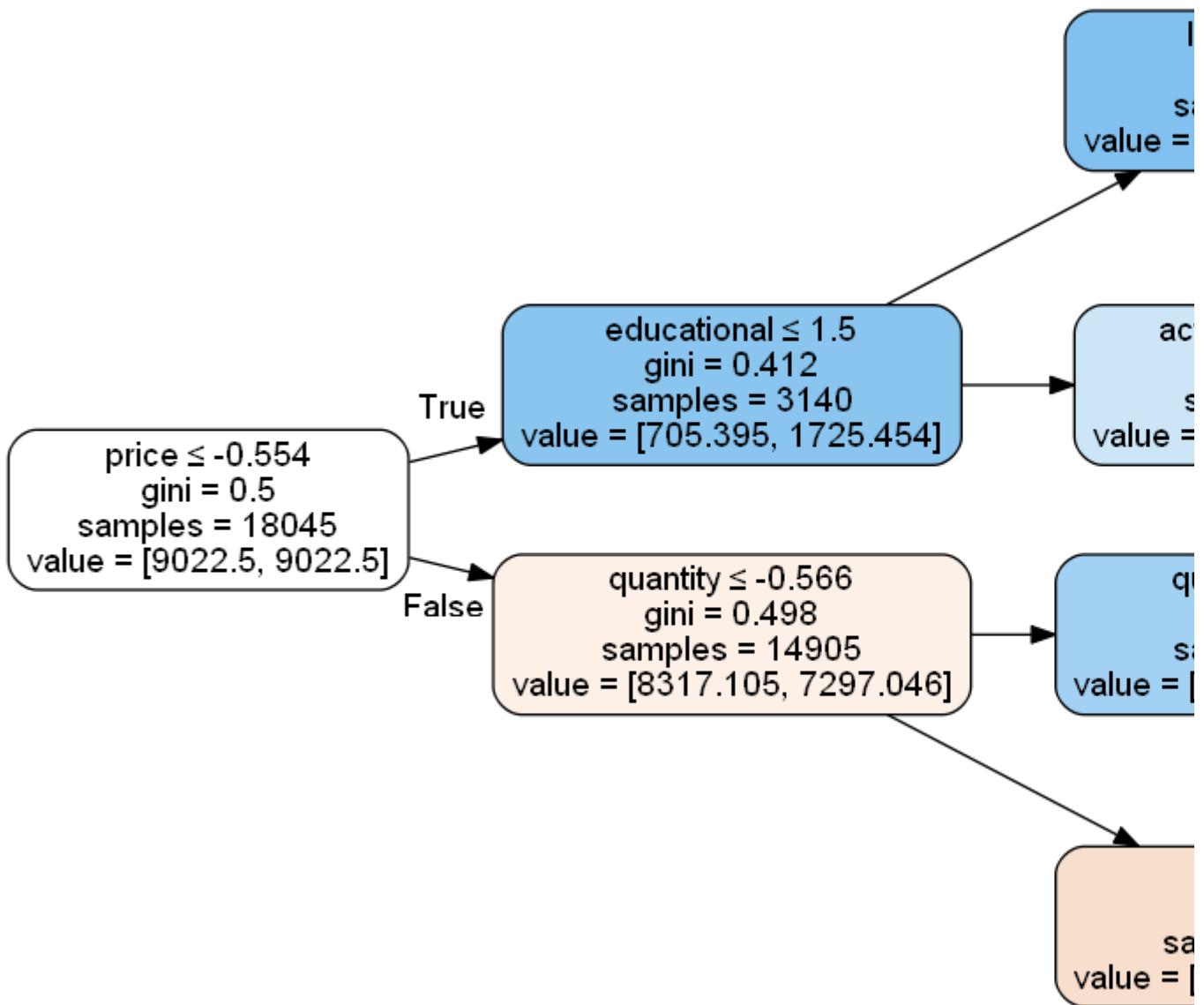
pip install pydotplus
```



Requirement already satisfied: pydotplus in c:\users\hn\anaconda3\lib\site-packages (2.0.0)

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(clfV1, out_file=dot_data, filled=True, rounded=True, special_characters=True,
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```





## ▼ Analysis on the False positives

```
#Get the False positives datapoints  
X_test['essay'].values[1]
```

 "My classroom is filled with fun-loving special education students that are happy, activ

```
#https://www.google.com/search?q=geeks+for+geeks+false+positive&rlz=1C1SQJL_enIN849IN849&oq=g
#https://github.com/pskadasi/DecisionTrees_DonorsChoose/blob/master/Copy_of_8_DonorsChoose_DT
fpi = []
for i in range(len(y_test)) :
    if (y_test.values[i] == 0) & (predictions1[i] == 1) :
        fpi.append(i)
fp_essay1 = []
for i in fpi :
    fp_essay1.append(X_test['essay'].values[i])
```

WORD CLOUD OF ESSAY: Word Cloud is a data visualization technique used for representing text data and its frequency or importance. Significant textual data points can be highlighted using a word cloud. We can get data from social network websites.

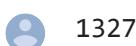
```
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for val in fp_essay1 :
    val = str(val)
    tokens = val.split()
for i in range(len(tokens)):
    tokens[i] = tokens[i].lower()
for words in tokens :
    comment_words = comment_words + words + ' '
wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = stopwords,
                      random_state = 21, max_font_size = 36)
plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```





## ▼ DataFrame of False Positives

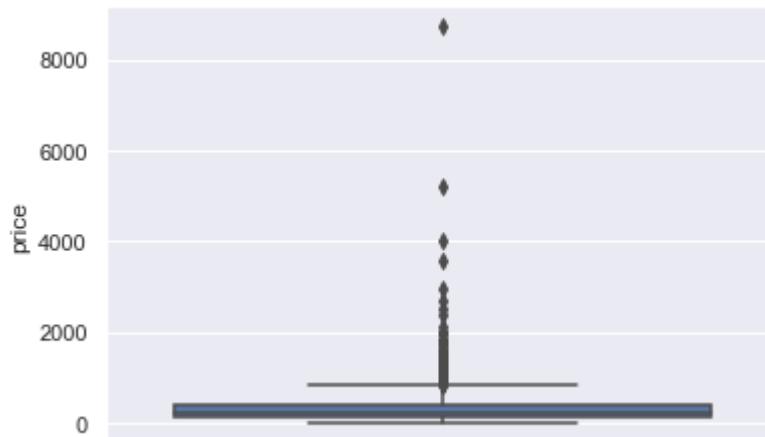
```
# first get the columns:  
cols = X_test.columns  
X_test_falsePos1 = pd.DataFrame(columns=cols)  
  
# get the data of the false pisitives  
for i in fpi : # (in fpi all the false positives data points indexes)  
    X_test_falsePos1 = X_test_falsePos1.append(X_test.filter(items=[i], axis=0))  
  
X_test_falsePos1.head(1)  
len(X_test_falsePos1)
```



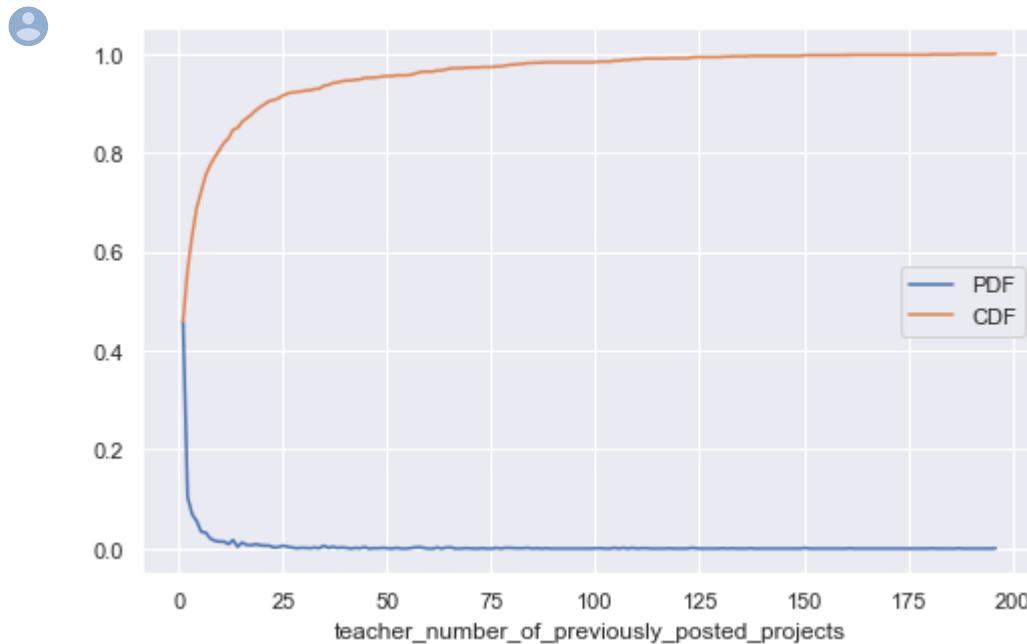
```
##Box Plot (FP 'price')
sns.boxplot(y='price', data=X_test_falsePos1)
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x16d2c8e7588>
```



```
##PDF (FP ,teacher_number_of_previously_posted_projects)
plt.figure(figsize=(8,5))
counts, bin_edges = np.histogram(X_test_falsePos1[ 'teacher_number_of_previously_posted_projects'],
bins='auto', density=True)
pdf = counts/sum(counts)
cdf = np.cumsum(pdf)
pdfP, = plt.plot(bin_edges[1:], pdf)
cdfP, = plt.plot(bin_edges[1:], cdf)
plt.legend([pdfP, cdfP], ["PDF", "CDF"])
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.show()
```



## ▼ Applying Decision trees on TFIDF

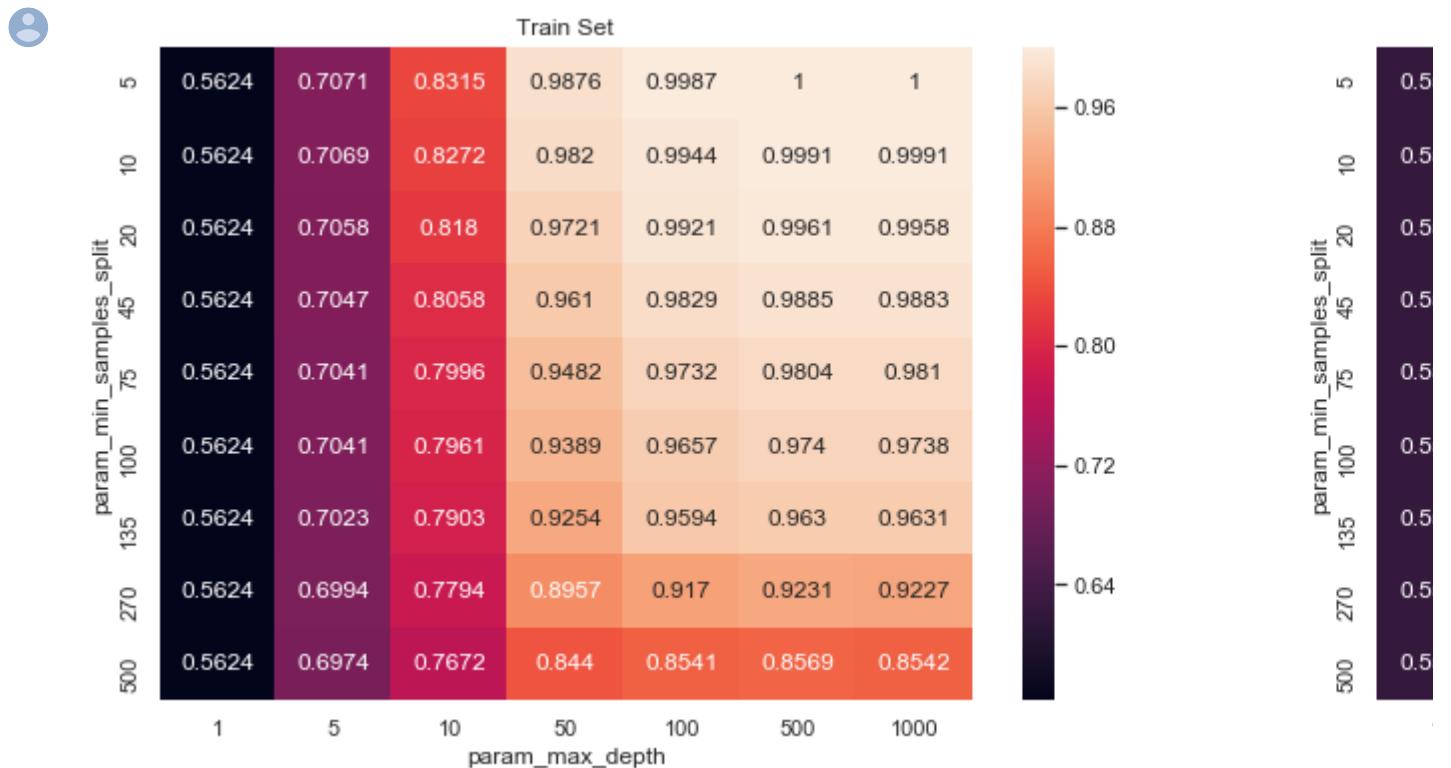
```

from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
dt2 = DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50, 100, 500, 1000], 'min_samples_split': [5, 10, 20, 40, 50, 75, 100, 135, 270, 500]}
clf2 = GridSearchCV(dt2, parameters, cv=3, scoring='roc_auc', return_train_score=True)
se2 = clf2.fit(X_set2_train, y_train)

import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf2.cv_results_).groupby(['param_min_samples_split', 'param_max_depth'])

fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt=' .4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt=' .4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()

```



## ▼ Best Estimator and Best tune parameters

```

print(clf2.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf2.score(X_set2_train,y_train))
print(clf2.score(X_set2_test,y_test))

```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=10,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=500,
                      min_weight_fraction_leaf=0.0, presort=False,
                      random_state=None, splitter='best')

0.7628998900413088
0.6658739514606636

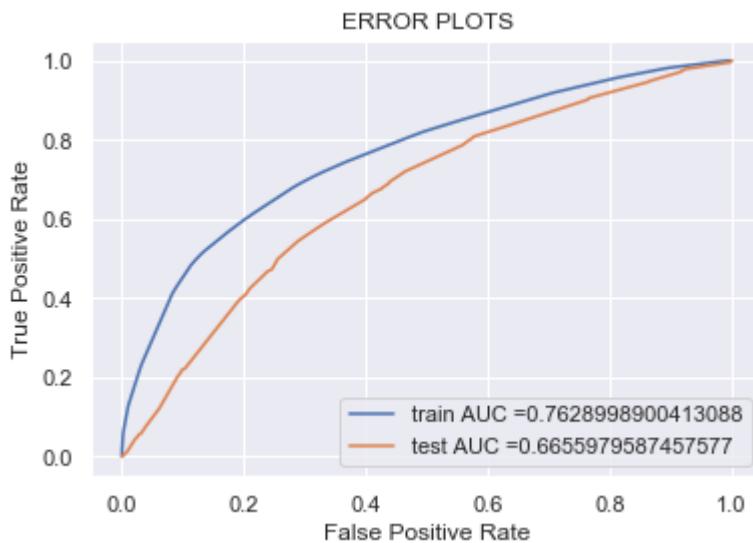
# Best tune parameters
best_tune_parameters=[{'max_depth':[10], 'min_samples_split':[500]}]

##Fitting Model to Hyper-Parameter Curve
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc\_curve.html#sklearn.metrics.roc\_curve
from sklearn.metrics import roc_curve, auc
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve, auc
clf11= GridSearchCV( DecisionTreeClassifier(class_weight = 'balanced'),best_tune_parameters)
clfV1=DecisionTreeClassifier (class_weight = 'balanced',max_depth=10,min_samples_split=500)
clf11.fit(X_set2_train, y_train)
# for visulation
clfV1.fit(X_set2_train, y_train)
#https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier.html#sklearn.linear\_model.SGDClassifier
y_train_pred1 = clf11.predict_proba(X_set2_train) [:,1]
y_test_pred1 = clf11.predict_proba(X_set2_test) [:,1]

train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)

plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid(True)
plt.show()
```



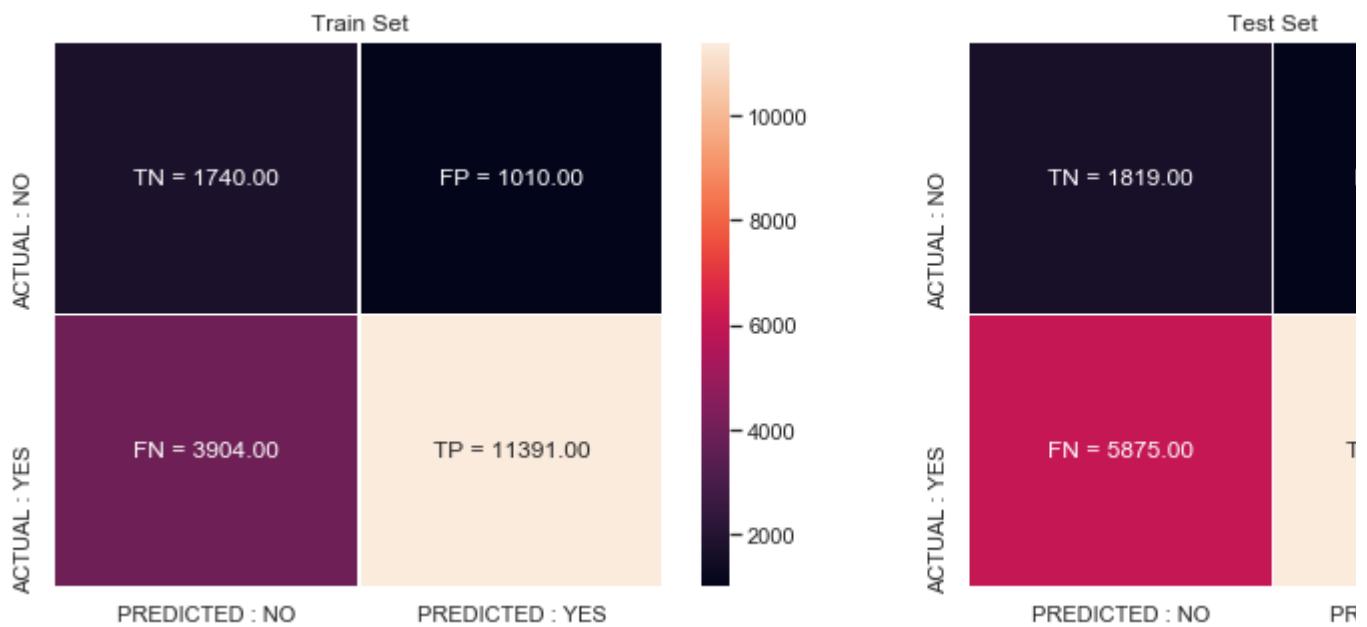


## ▼ Confusion matrix

```
import seaborn as sns; sns.set()
con_m_train = confusion_matrix(y_train, predict(y_train_pred1, tr_thresholds1, train_fpr1, tr_t
con_m_test = confusion_matrix(y_test, predict(y_test_pred1, te_thresholds1, test_fpr1, test_t
key = (np.asarray([['TN', 'FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(15,5))
labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flat
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flat
con_m_test.flatten()))])).reshape(2,2)
sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], yti
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytic
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
plt.show()
```



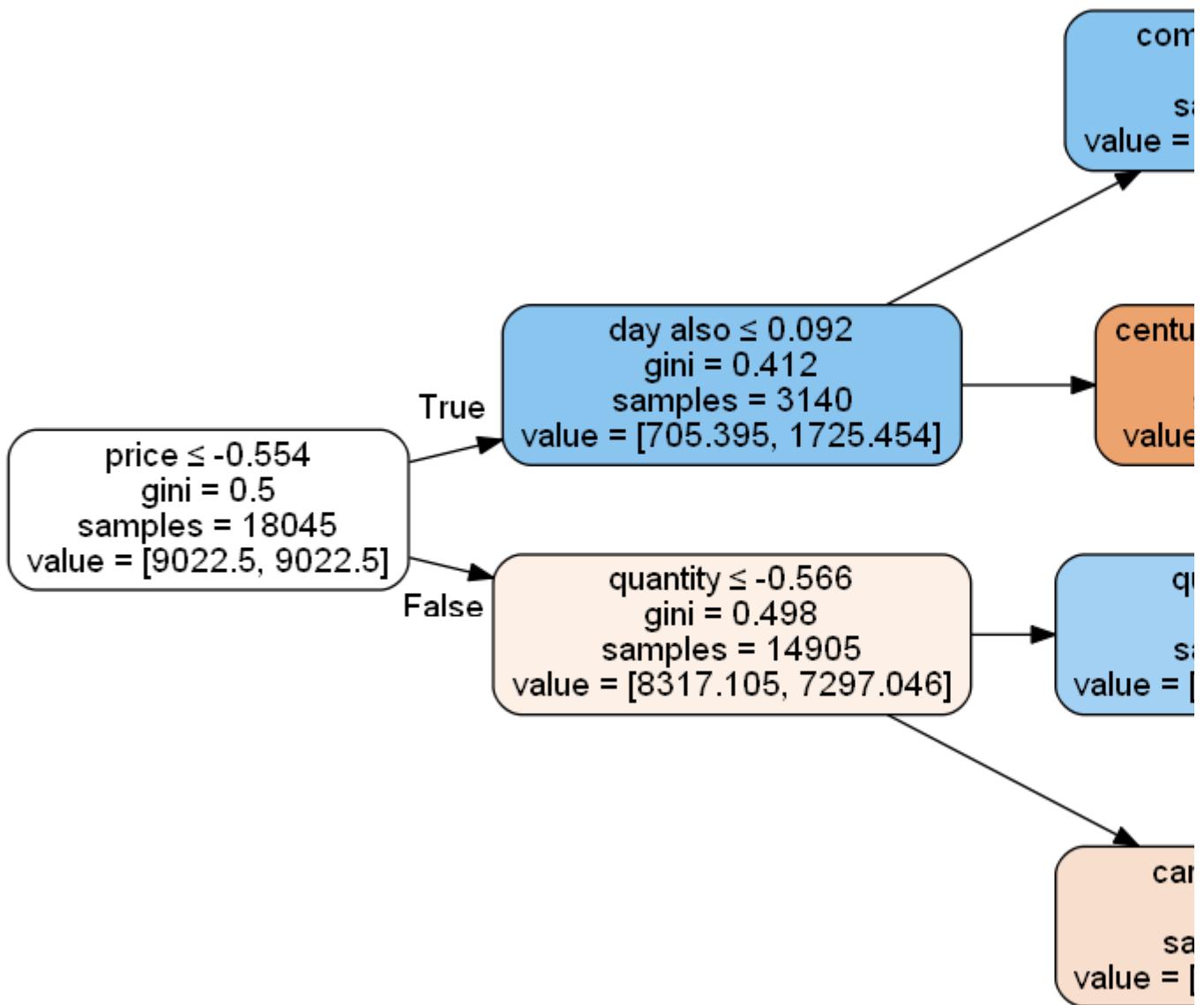
the maximum value of  $tpr*(1-fpr)$  0.49 for threshold 0.39  
 the maximum value of  $tpr*(1-fpr)$  0.39 for threshold 0.5



## ▼ Visualizing Decision Tree

```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(clfV1, out_file=dot_data, filled=True, rounded=True, special_characters=True,
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```





## ▼ Analysis on the False positives

```

#https://www.google.com/search?q=geeks+for+geeks+false+positive&rlz=1C1SQJL_enIN849IN849&oq=g
#https://github.com/pskadasi/DecisionTrees_DonorsChoose/blob/master/Copy_of_8_DonorsChoose_DT
fpi = []
for i in range(len(y_test)) :
    if (y_test.values[i] == 0) & (predictions1[i] == 1) :
        fpi.append(i)
  
```

```
fp.append(i)
fp_essay1 = []
for i in fpi :
    fp_essay1.append(X_test['essay'].values[i])

# Word cloud of essay
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for val in fp_essay1 :
    val = str(val)
    tokens = val.split()
for i in range(len(tokens)):
    tokens[i] = tokens[i].lower()
for words in tokens :
    comment_words = comment_words + words + ' '

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

plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

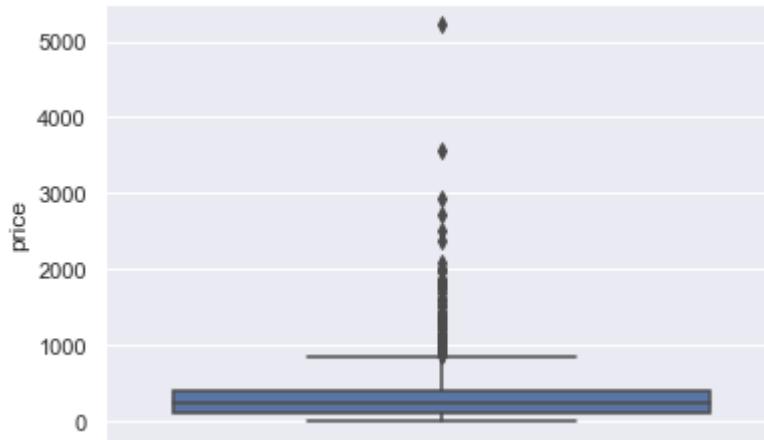


## ▼ DataFrame of False Positives

```
# first get the columns:
cols = X_test.columns
X_test_falsePos1 = pd.DataFrame(columns=cols)
# get the data of the false positives
for i in fpi : # (in fpi all the false positives data points indexes)
    X_test_falsePos1 = X_test_falsePos1.append(X_test.filter(items=[i], axis=0))
```

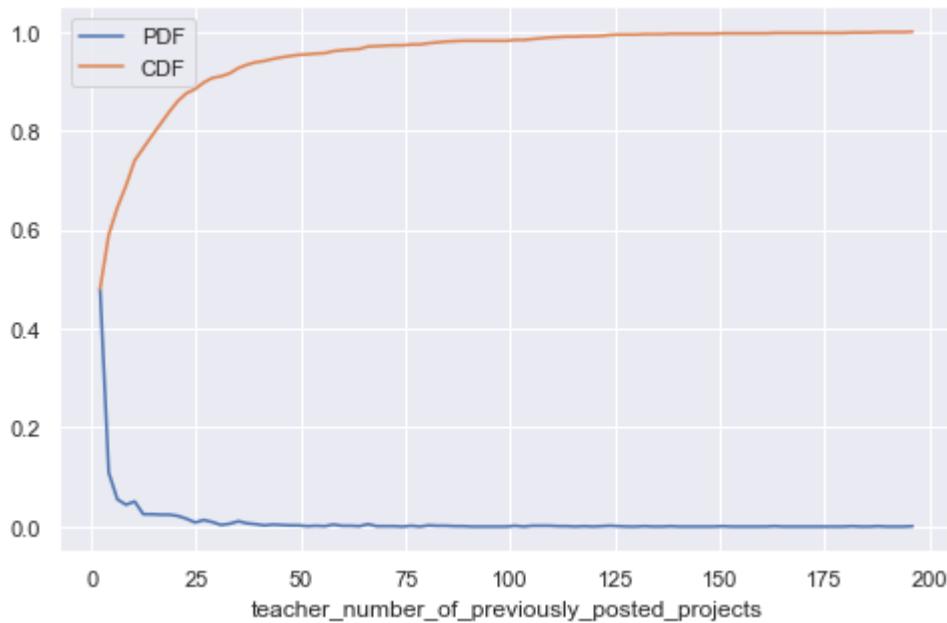
```
#Box Plot (FP 'price')
sns.boxplot(y='price', data=X_test_falsePos1)
```

👤 <matplotlib.axes.\_subplots.AxesSubplot at 0x16bc09070b8>



```
#PDF (FP ,teacher_number_of_previously_posted_projects)
plt.figure(figsize=(8,5))
counts, bin_edges = np.histogram(X_test_falsePos1['teacher_number_of_previously_posted_projects'])
pdf = counts/sum(counts)
cdf = np.cumsum(pdf)
pdfP, = plt.plot(bin_edges[1:], pdf)
cdfP, = plt.plot(bin_edges[1:], cdf)
plt.legend([pdfP, cdfP], ["PDF", "CDF"])
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.show()
```

👤



## ▼ Applying Decision trees on AVG W2V

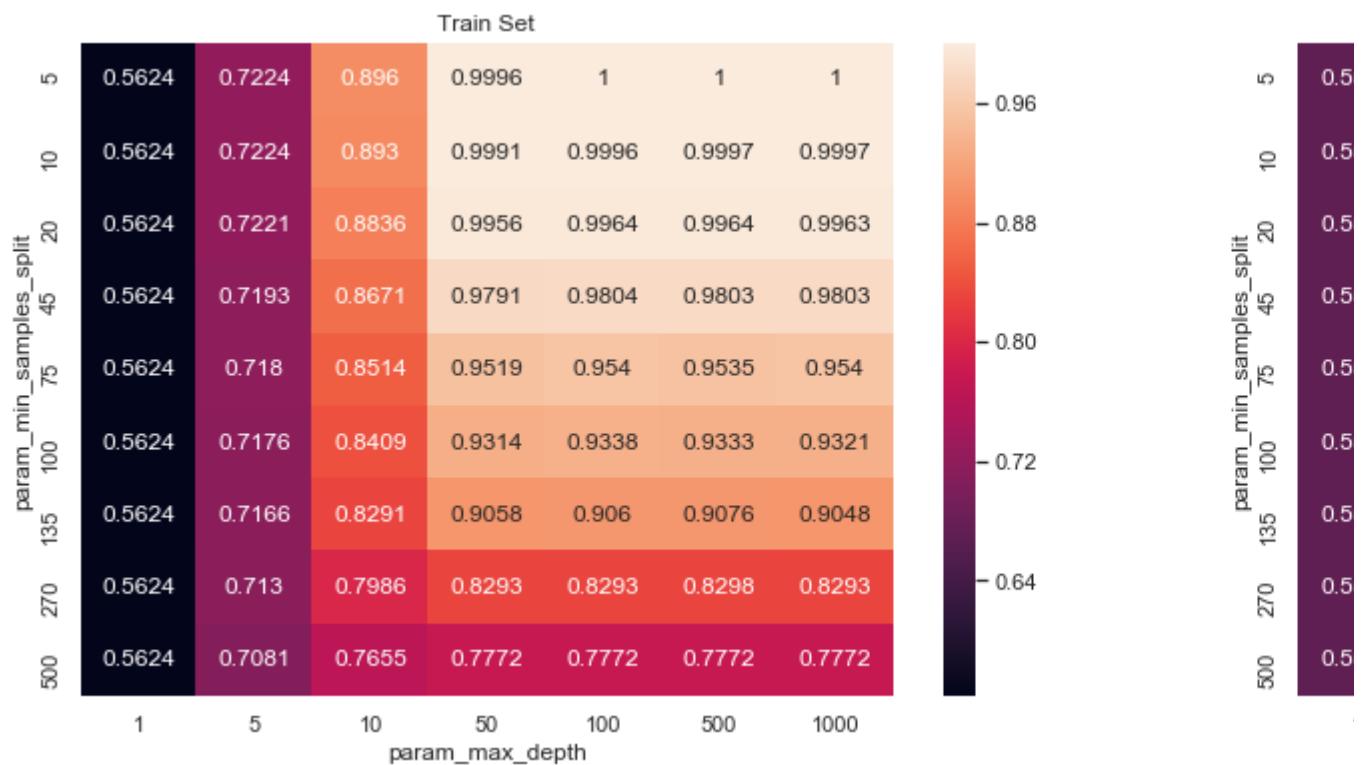
```

from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
dt3= DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50, 100, 500, 1000], 'min_samples_split': [5, 10, 20, 4
clf3 = GridSearchCV(dt3, parameters, cv=3, scoring='roc_auc',n_jobs=4,return_train_score=True
se3 = clf3.fit(X_set3_train, y_train)

import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf3.cv_results_).groupby(['param_min_samples_split', 'param_max_d
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt=' .4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt=' .4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()

```





```
#Best Estimator and Best tune parameters
```

```
print(clf3.best_estimator_)
```

```
#Mean cross-validated score of the best_estimator
```

```
print(clf3.score(X_set3_train,y_train))
```

```
print(clf3.score(X_set3_test,y_test))
```



```
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=5,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=500,
min_weight_fraction_leaf=0.0, presort=False,
random_state=None, splitter='best')
```

```
0.6965402121905555
```

```
0.6437607521359746
```

```
# Best tune parameters
```

```
best_tune_parameters=[{'max_depth':[5], 'min_samples_split':[500] } ]
```

```
#Fitting Model to Hyper-Parameter Curve
```

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

```
from sklearn.metrics import roc_curve, auc
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.metrics import roc_curve, auc
```

```
clf11= GridSearchCV( DecisionTreeClassifier(class_weight = 'balanced'),best_tune_parameters)
```

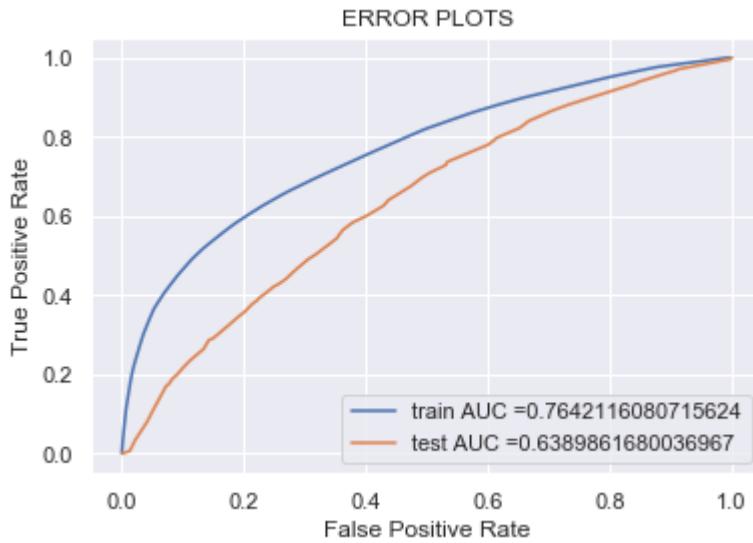
```
clfV1=DecisionTreeClassifier (class_weight = 'balanced',max_depth=5,min_samples_split=500)
```

```
clf11.fit(X_set3_train, y_train)
```

```
# for visulation
```

```
clfV1.fit(X_set3_train, y_train)
```

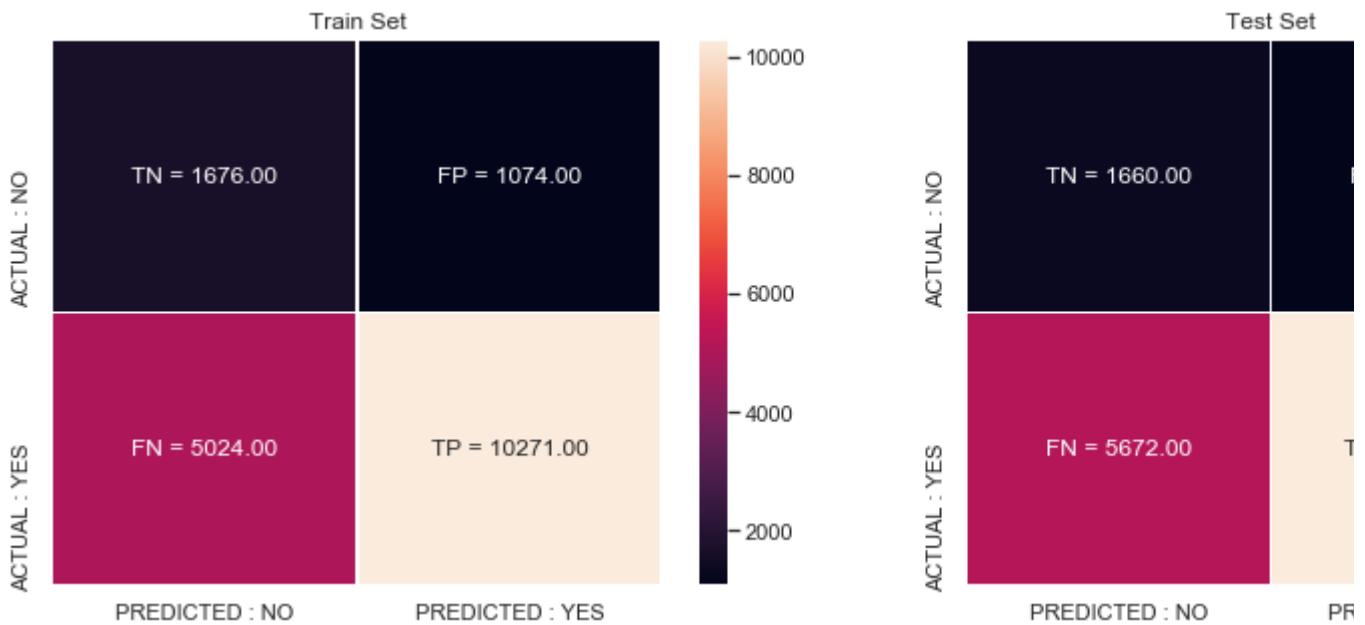
```
#https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SVCClassifier.html#skl
y_train_pred1 = clf11.predict_proba(X_set3_train) [:,1]
y_test_pred1 = clf11.predict_proba(X_set3_test) [:,1]
train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)
plt.plot(train_fpr1, train_tpr1, label="train AUC =" + str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC =" + str(auc(test_fpr1, test_tpr1)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid(True)
plt.show()
```



```
#confusion matrix test data
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()
con_m_train = confusion_matrix(y_train, predict(y_train_pred1, tr_thresholds1, train_fpr1, tr
con_m_test = confusion_matrix(y_test, predict(y_test_pred1, te_thresholds1, test_fpr1, test_t
key = (np.asarray([['TN', 'FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(15,5))
labels_train = (np.asarray(["{} = {:.2f}"] .format(key, value) for key, value in zip(key.fl
labels_test = (np.asarray(["{} = {:.2f}"] .format(key, value) for key, value in zip(key.flat
sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_train, fmt = ' ', ax=ax[0])
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'],
yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], annot = labels_test, fmt = ' ', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
plt.show()
```



the maximum value of  $tpr*(1-fpr)$  0.41 for threshold 0.47  
 the maximum value of  $tpr*(1-fpr)$  0.37 for threshold 0.47



##Analysis on the False positives

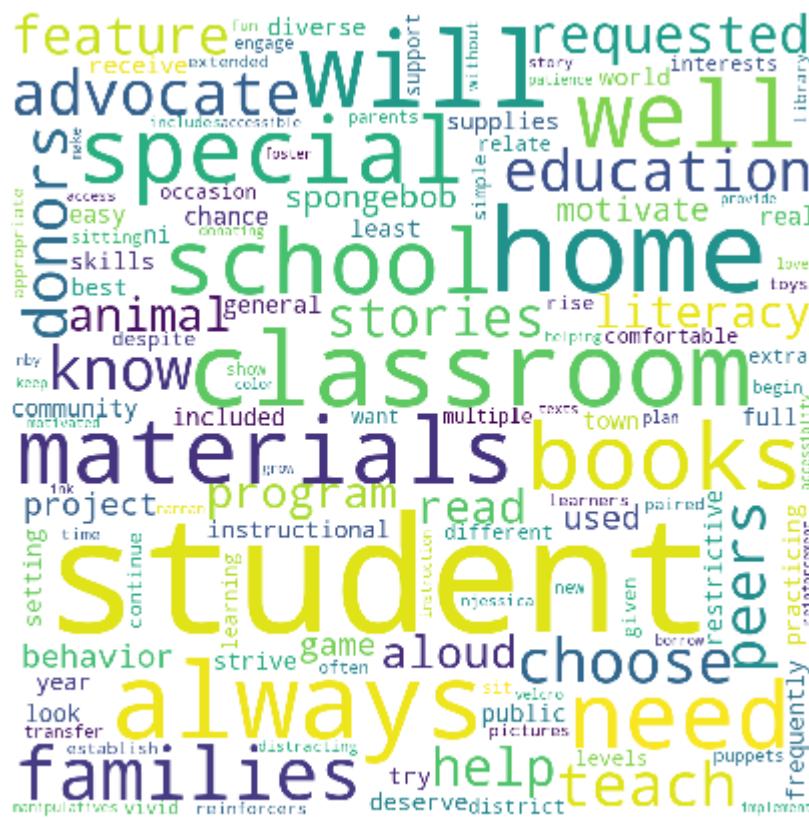
```
fpi = []
for i in range(len(y_test)) :
    if (y_test.values[i] == 0) & (predictions1[i] == 1) :
        fpi.append(i)
fp_essay1 = []
for i in fpi :
    fp_essay1.append(X_test['essay'].values[i])
```

pip install wordcloud

Requirement already satisfied: wordcloud in c:\users\hp\anaconda3\lib\site-packages (1.5)
 Requirement already satisfied: numpy>=1.6.1 in c:\users\hp\anaconda3\lib\site-packages (from wordcloud)
 Requirement already satisfied: pillow in c:\users\hp\anaconda3\lib\site-packages (from wordcloud)
 Note: you may need to restart the kernel to use updated packages.

```
#Word cloud of essay
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for val in fp_essay1 :
    val = str(val)
    tokens = val.split()
for i in range(len(tokens)):
    tokens[i] = tokens[i].lower()
for words in tokens :
    comment_words = comment_words + words + ' '
wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = stopwords, random_state = 21, collocations=False)
plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
```

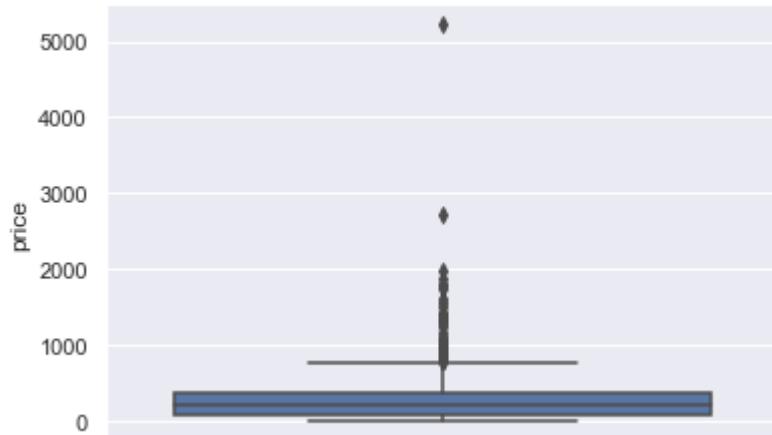
```
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



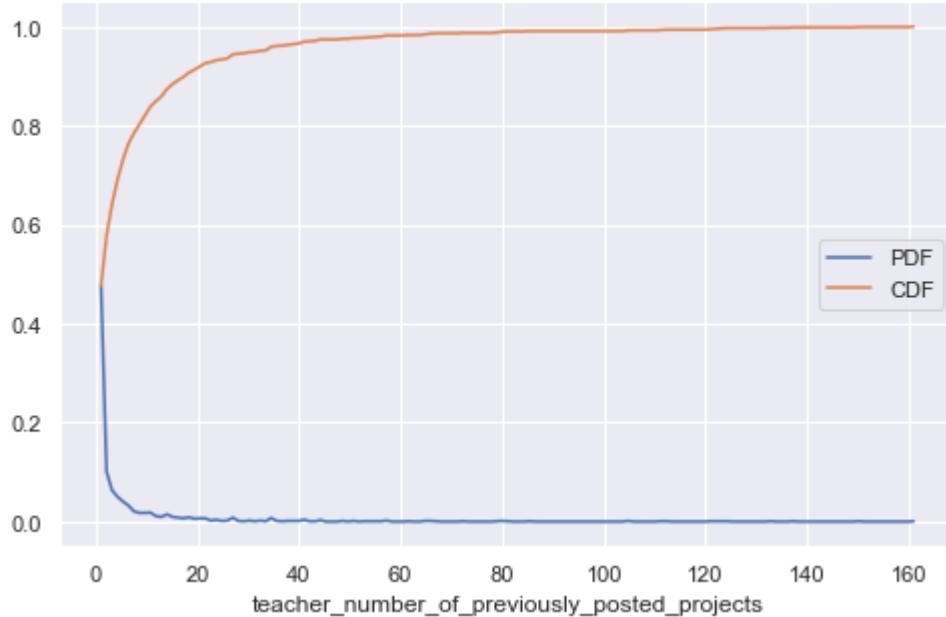
```
#DataFrame of False Positives
# first get the columns:
cols = X_test.columns
X_test_falsePos1 = pd.DataFrame(columns=cols)
# get the data of the false pisitives
for i in fpi : # (in fpi all the false positives data points indexes)
    X_test_falsePos1 = X_test_falsePos1.append(X_test.filter(items=[i], axis=0))

#Box Plot (FP 'price')
sns.boxplot(y='price', data=X_test_falsePos1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x16bb87b9240>
```



```
#PDF (FP ,teacher_number_of_previously_posted_projects)
plt.figure(figsize=(8,5))
counts, bin_edges = np.histogram(X_test_falsePos1[ 'teacher_number_of_previously_posted_projects'],
bins='auto', density=True)
pdf = counts/sum(counts)
cdf = np.cumsum(pdf)
pdfP, = plt.plot(bin_edges[1:], pdf)
cdfP, = plt.plot(bin_edges[1:], cdf)
plt.legend([pdfP, cdfP], ["PDF", "CDF"])
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.show()
```



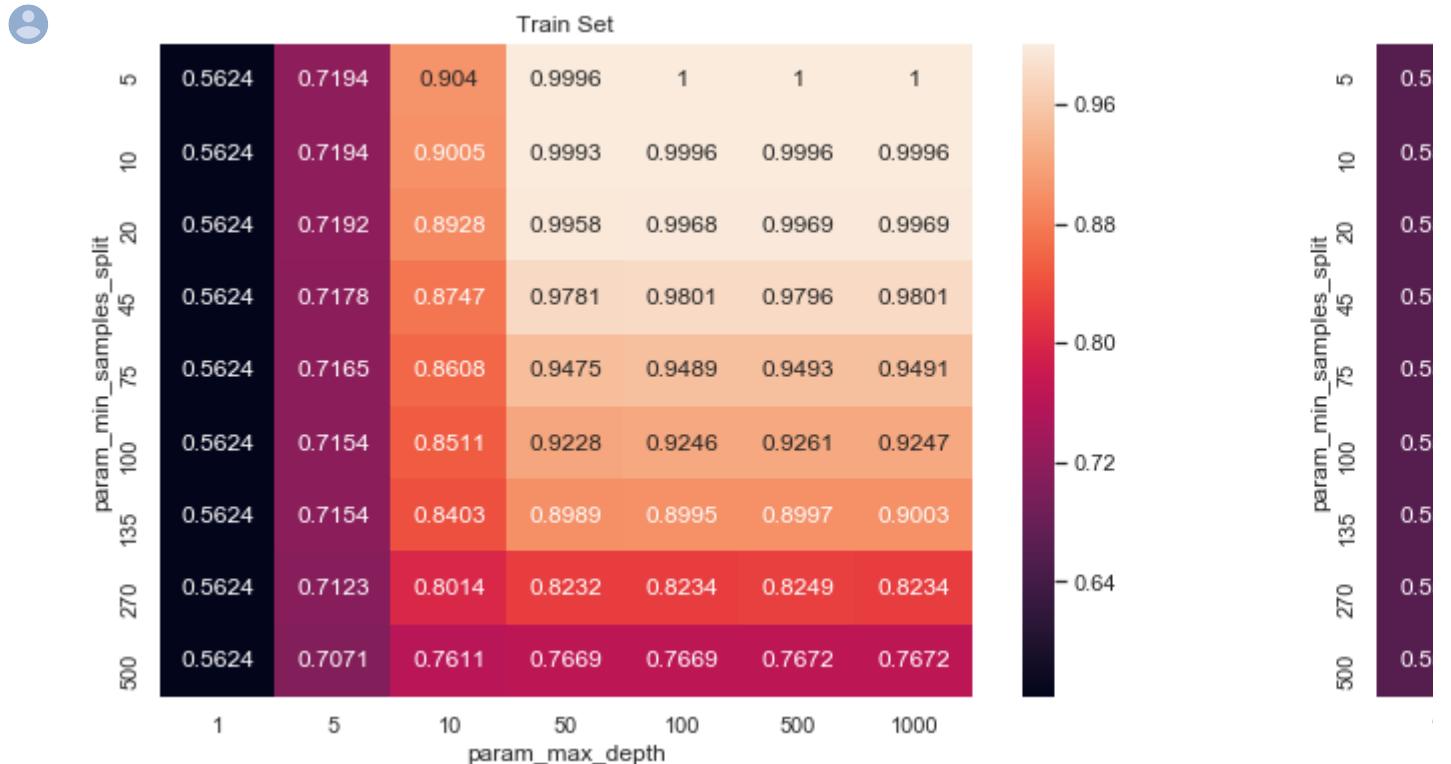
## ▼ Applying Decision trees on td\_idf W2V

```

from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
dt4= DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50, 100, 500, 1000], 'min_samples_split': [5, 10, 20, 45, 75, 100, 135, 270, 500]}
clf4 = GridSearchCV(dt4, parameters, cv=3, scoring='roc_auc', return_train_score=True)
set4= clf4.fit(X_set4_train, y_train)

import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf4.cv_results_).groupby(['param_min_samples_split', 'param_max_depth'])
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()

```



```

#Best Estimator and Best tune parameters
print(clf4.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf4.score(X_set4_train,y_train))
print(clf4.score(X_set4_test,y_test))

```

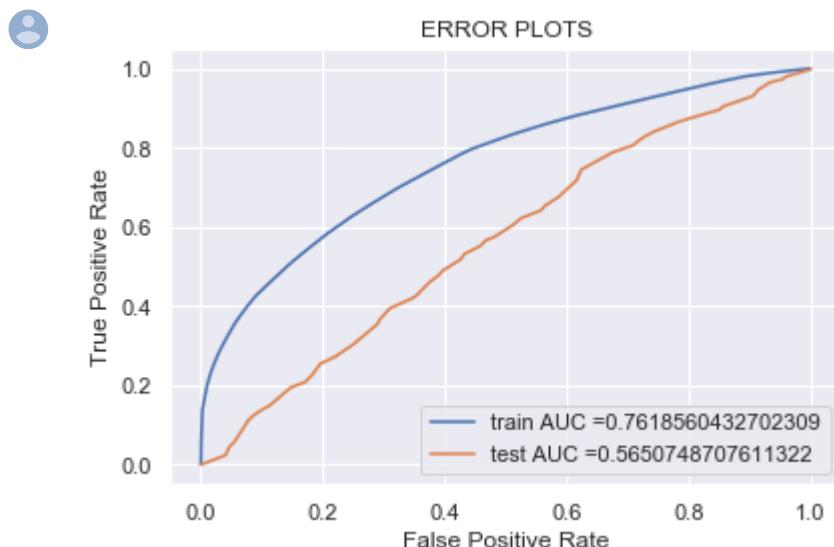


```
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=5,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=500,
                      min_weight_fraction_leaf=0.0, presort=False,
                      random_state=None, splitter='best')

0.6969748759249904
0.6138512151971972
```

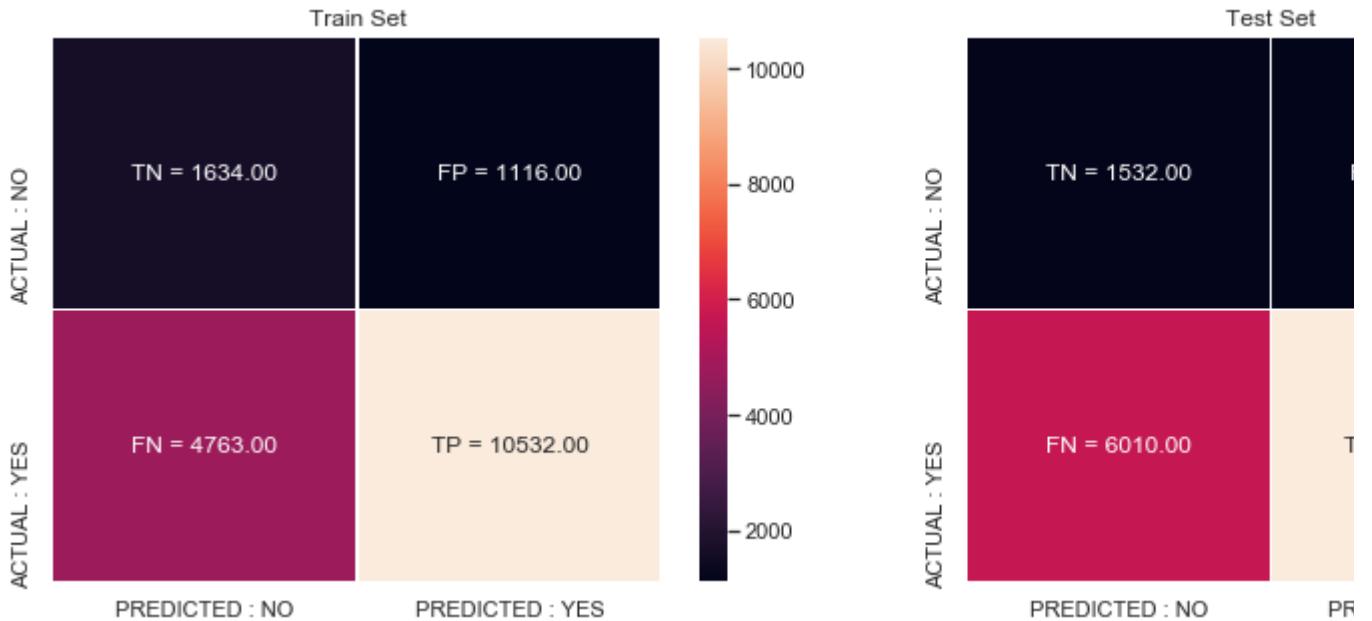
```
best_tune_parameters= [{ 'max_depth': [5], 'min_samples_split':[500] }]
```

```
#Fitting Model to Hyper-Parameter Curve
from sklearn.metrics import roc_curve, auc
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve, auc
clf11= GridSearchCV( DecisionTreeClassifier(class_weight = 'balanced'),best_tune_parameters)
clfV1=DecisionTreeClassifier (class_weight = 'balanced',max_depth=5,min_samples_split=500)
clf11.fit(X_set4_train, y_train)
# for visulation
clfV1.fit(X_set4_train, y_train)
#https://scikitlearn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier.html#skl
y_train_pred1 = clf11.predict_proba(X_set4_train) [:,1]
y_test_pred1 = clf11.predict_proba(X_set4_test) [:,1]
train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)
plt.plot(train_fpr1, train_tpr1, label="train AUC =" +str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC =" +str(auc(test_fpr1, test_tpr1)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid(True)
plt.show()
```



```
#CONFUSION MATRIX
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()
con_m_train = confusion_matrix(y_train, predict(y_train_pred1, tr_thresholds1, train_fpr1, tr_t))
con_m_test = confusion_matrix(y_test, predict(y_test_pred1, te_thresholds1, test_fpr1, test_t))
key = (np.asarray([['TN', 'FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(15,5))
labels_train = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flat, con_m_train)]))
labels_test = (np.asarray(["{0} = {1:.2f}" .format(key, value) for key, value in zip(key.flat, con_m_test)]))
sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], ax=ax[0])
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], yticklabels=['ACTUAL : NO', 'ACTUAL : YES'], ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
plt.show()
```

 the maximum value of  $tpr * (1 - fpr)$  0.42 for threshold 0.45  
the maximum value of  $tpr * (1 - fpr)$  0.33 for threshold 0.5



```
#Analysis on the False positives
fpi = []
for i in range(len(y_test)) :
    if (y_test.values[i] == 0) & (predictions1[i] == 1) :
        fpi.append(i)
fp_essay1 = []
for i in fpi :
    fp_essay1.append(X_test['essay'].values[i])
```

```
#WORD CLOUD OF ESSAY
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for val in fp_essay1 :
    val = str(val)
    tokens = val.split()
```

```
for i in range(len(tokens)):
    tokens[i] = tokens[i].lower()
for words in tokens :
    comment_words = comment_words + words + ' '
wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = stopw
min_font_size = 10).generate(comment_words)

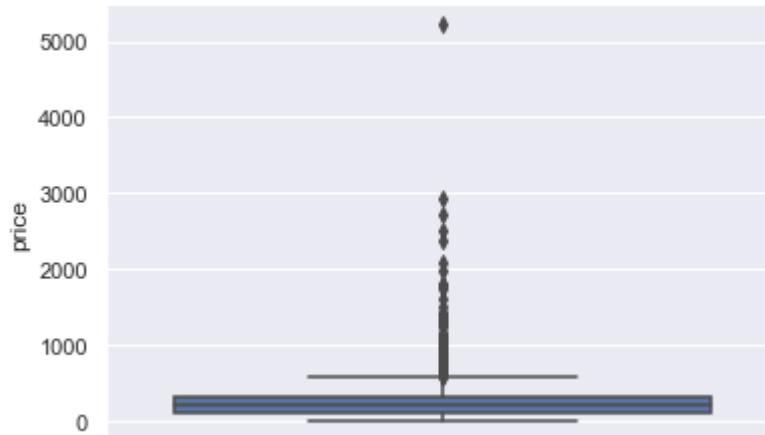
plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



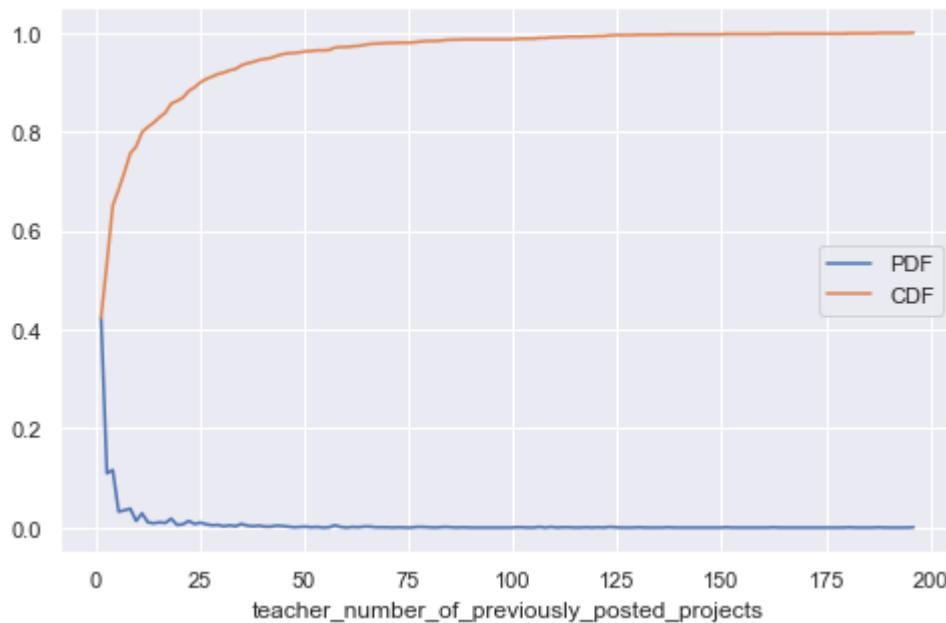
```
#Box Plot (FP 'price')
# first get the columns:
cols = X_test.columns
X_test_falsePos1 = pd.DataFrame(columns=cols)
# get the data of the false positives
for i in fpi : # (in fpi all the false positives data points indexes)
    X_test_falsePos1 = X_test_falsePos1.append(X_test.filter(items=[i], axis=0))
sns.boxplot(y='price', data=X_test_falsePos1)
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x16d281599b0>
```



```
#PDF (FP ,teacher_number_of_previously_posted_projects)
plt.figure(figsize=(8,5))
counts, bin_edges = np.histogram(X_test_falsePos1[ 'teacher_number_of_previously_posted_projects']
pdf = counts/sum(counts)
cdf = np.cumsum(pdf)
pdfP, = plt.plot(bin_edges[1:], pdf)
cdfP, = plt.plot(bin_edges[1:], cdf)
plt.legend([pdfP, cdfP], ["PDF", "CDF"])
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.show()
```



Select 5k best features from features of Set 2 using feature\_importances, discard all the other remain model of your choice i.e.(Dession tree, Logistic Regression, Linear SVM), you need to do hyperparamete selected and procedure in step 2 and step 3

```
#https://stackoverflow.com/questions/47111434/randomforestregressor-and-feature-importances-e
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
def selectKImportance(model, X, k=5):
    return X[:,model.best_estimator_.feature_importances_.argsort()[:-1][:-k]]

# for tf-idf set 2
X_set5_train = selectKImportance(clf2, xtr,5000)
X_set5_test = selectKImportance(clf2, X_set2_test, 5000)

print(X_set5_train.shape)
print(X_set5_test.shape)
```

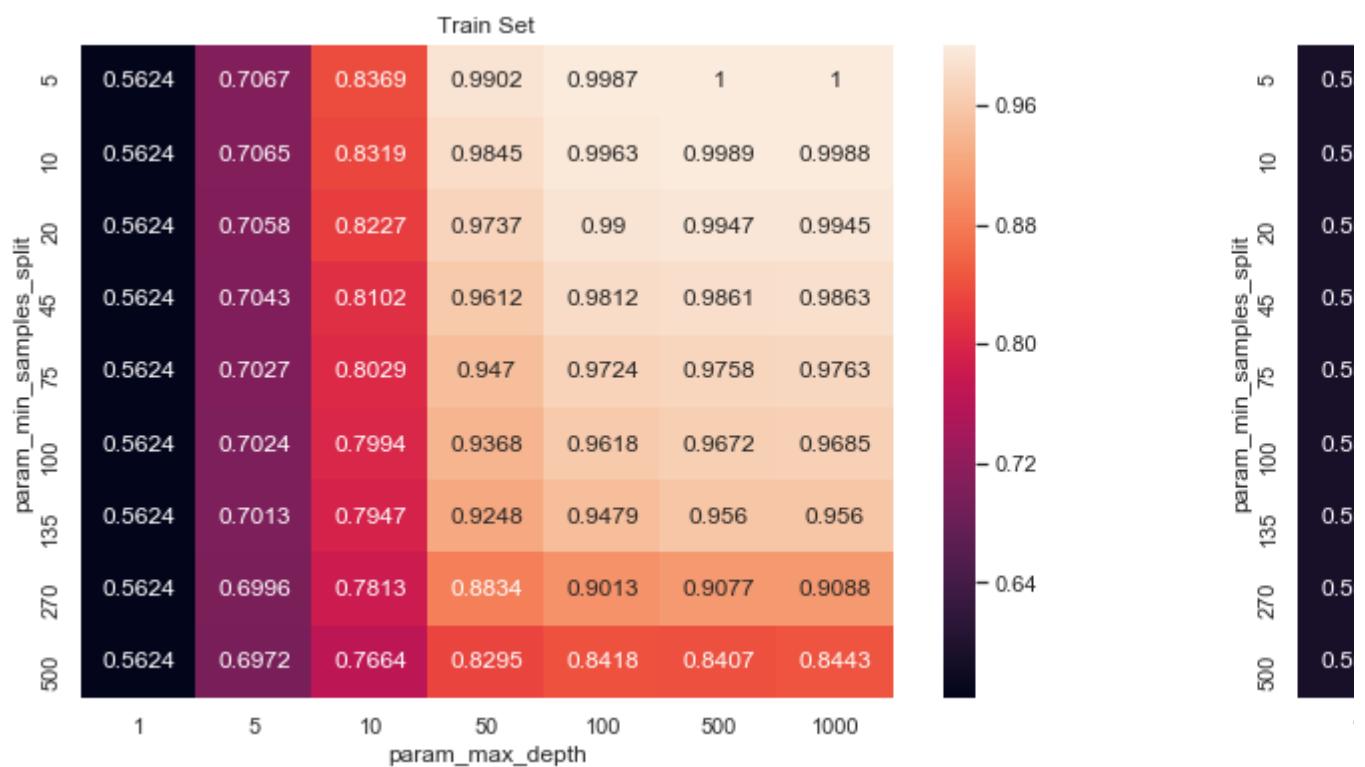
👤 (18045, 5000)  
 (19800, 5000)

## ▼ Decision tree on Important features

```
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
dt5= DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50, 100, 500, 1000], 'min_samples_split': [5, 10, 20, 40, 50, 100, 200, 500, 1000]}
clf5 = GridSearchCV(dt5, parameters, cv=3, scoring='roc_auc', return_train_score=True)
set5= clf5.fit(X_set5_train, y_train)

import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(clf5.cv_results_).groupby(['param_min_samples_split', 'param_max_depth'])
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='%.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='%.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```

👤



```
#Best Estimator and Best tune parameters
```

```
print(clf5.best_estimator_)
```

```
#Mean cross-validated score of the best_estimator
```

```
print(clf5.score(X_set5_train,y_train))
```

```
print(clf5.score(X_set5_test,y_test))
```



```
DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=10,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=500,
min_weight_fraction_leaf=0.0, presort=False,
random_state=None, splitter='best')
```

```
0.7628998900413088
```

```
0.6659494725920092
```

```
# Best tune parameters
```

```
best_tune_parameters=[{'max_depth': [10], 'min_samples_split':[500] } ]
```

```
# train with best hyperparameter
```

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

```
from sklearn.metrics import roc_curve, auc
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.metrics import roc_curve, auc
```

```
clf11= GridSearchCV( DecisionTreeClassifier(class_weight = 'balanced'),best_tune_parameters)
```

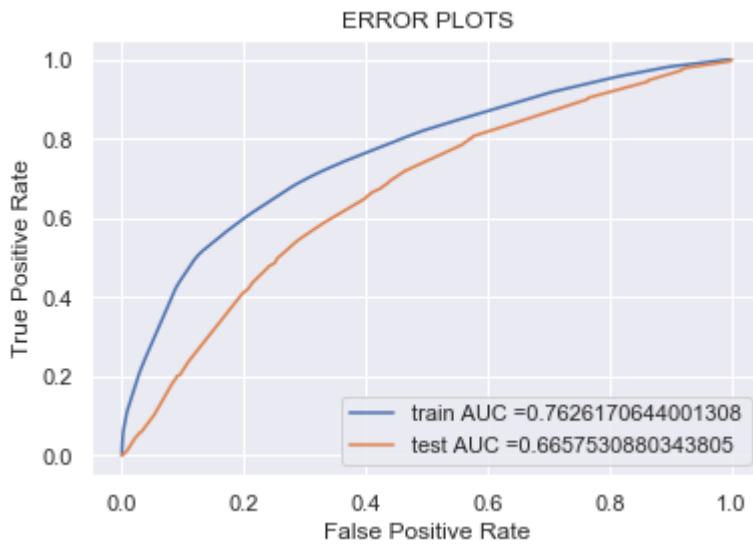
```
clfV1=DecisionTreeClassifier (class_weight = 'balanced',max_depth=10,min_samples_split=500)
```

```
clf11.fit(X_set5_train, y_train)
```

```
# for visulation
```

```
clfV1.fit(X_set5_train, y_train)
```

```
#https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SVCClassifier.html#skl
y_train_pred1 = clf11.predict_proba(X_set5_train)[:,1]
y_test_pred1 = clf11.predict_proba(X_set5_test)[:,1]
train_fpr1, train_tpr1, tr_thresholds1 = roc_curve(y_train, y_train_pred1)
test_fpr1, test_tpr1, te_thresholds1 = roc_curve(y_test, y_test_pred1)
plt.plot(train_fpr1, train_tpr1, label="train AUC ="+str(auc(train_fpr1, train_tpr1)))
plt.plot(test_fpr1, test_tpr1, label="test AUC ="+str(auc(test_fpr1, test_tpr1)))
plt.legend()
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ERROR PLOTS")
plt.grid(True)
plt.show()
```

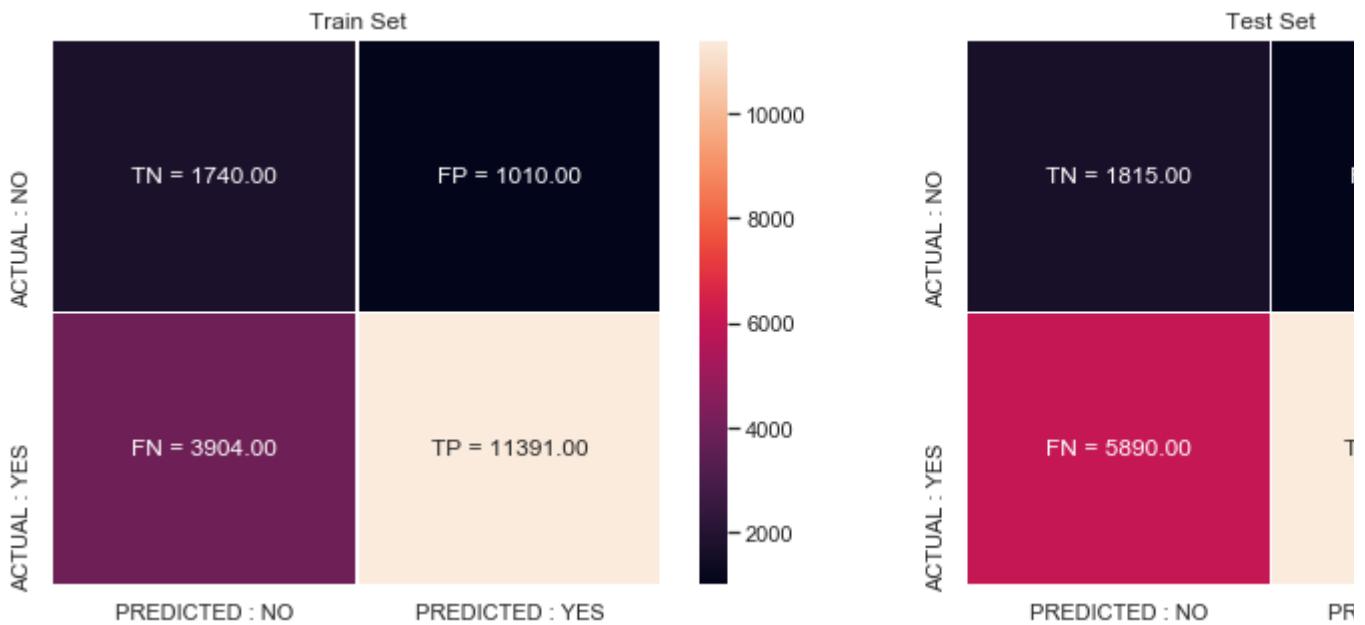


```
#CONFUSION MATRIX
```

```
#https://www.quantinsti.com/blog/creating-heatmap-using-python-seaborn
import seaborn as sns; sns.set()
con_m_train = confusion_matrix(y_train, predict(y_train_pred1, tr_thresholds1, train_fpr1, tr
con_m_test = confusion_matrix(y_test, predict(y_test_pred1, te_thresholds1, test_fpr1, test_t
key = (np.asarray([['TN', 'FP'], ['FN', 'TP']]))
fig, ax = plt.subplots(1,2, figsize=(15,5))
labels_train = (np.asarray(["{} = {:.2f}" .format(key, value) for key, value in zip(key.fl
labels_test = (np.asarray(["{} = {:.2f}" .format(key, value) for key, value in zip(key.flat
con_m_test.flatten()))])).reshape(2,2)
sns.heatmap(con_m_train, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], yti
sns.heatmap(con_m_test, linewidths=.5, xticklabels=['PREDICTED : NO', 'PREDICTED : YES'], ytic
ax[0].set_title('Train Set')
ax[1].set_title('Test Set')
plt.show()
```



the maximum value of  $tpr*(1-fpr)$  0.49 for threshold 0.39  
 the maximum value of  $tpr*(1-fpr)$  0.39 for threshold 0.5



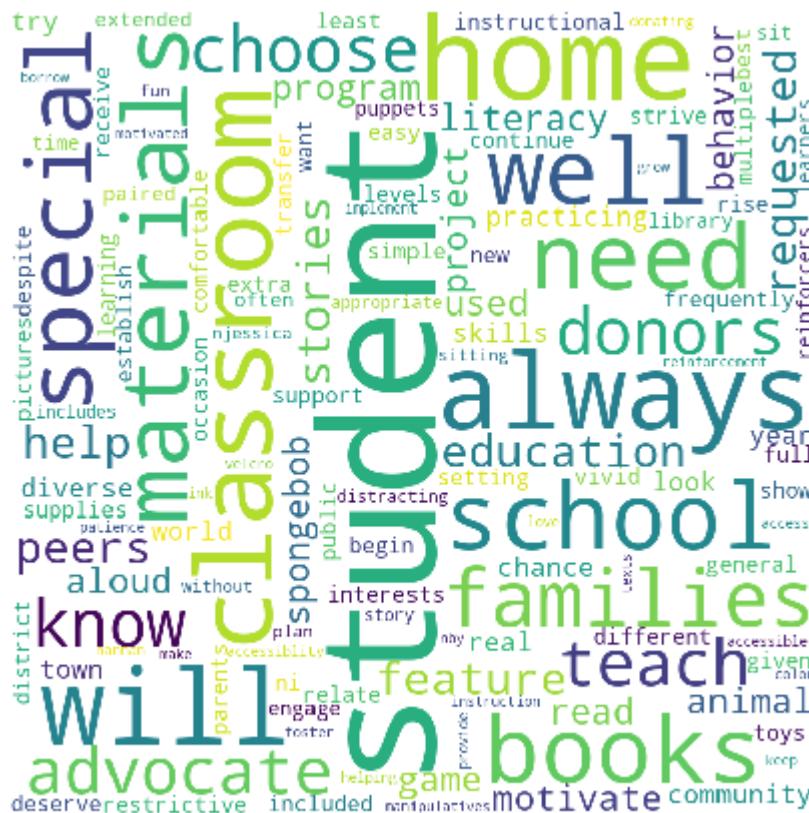
#Analysis on the False positives

```
fpi = []
for i in range(len(y_test)) :
    if (y_test.values[i] == 0) & (predictions1[i] == 1) :
        fpi.append(i)
fp_essay1 = []
for i in fpi :
    fp_essay1.append(X_test['essay'].values[i])

# Word cloud of essay
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for val in fp_essay1 :
    val = str(val)
    tokens = val.split()
for i in range(len(tokens)):
    tokens[i] = tokens[i].lower()
for words in tokens :
    comment_words = comment_words + words + ' '

wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = stopw

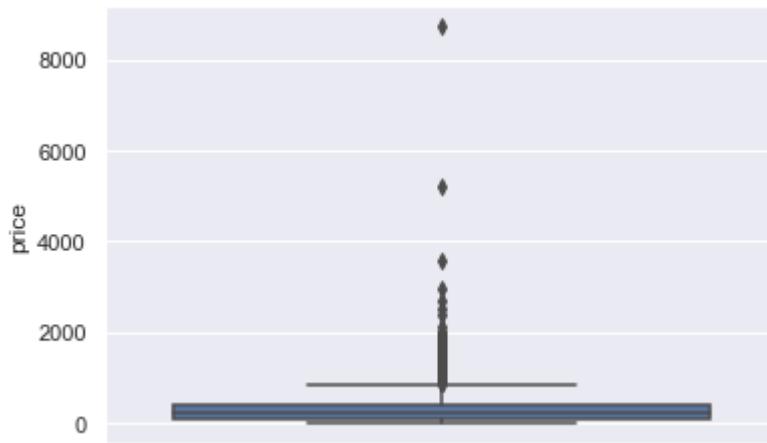
plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



```
#Box Plot (FP 'price')
# first get the columns:
cols = X_test.columns
X_test_falsePos1 = pd.DataFrame(columns=cols)
# get the data of the false positives
for i in fpi : # (in fpi all the false positives data points indexes)
    X_test_falsePos1 = X_test_falsePos1.append(X_test.filter(items=[i], axis=0))
sns.boxplot(y='price', data=X_test_falsePos1)
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x16bc088ad68>
```

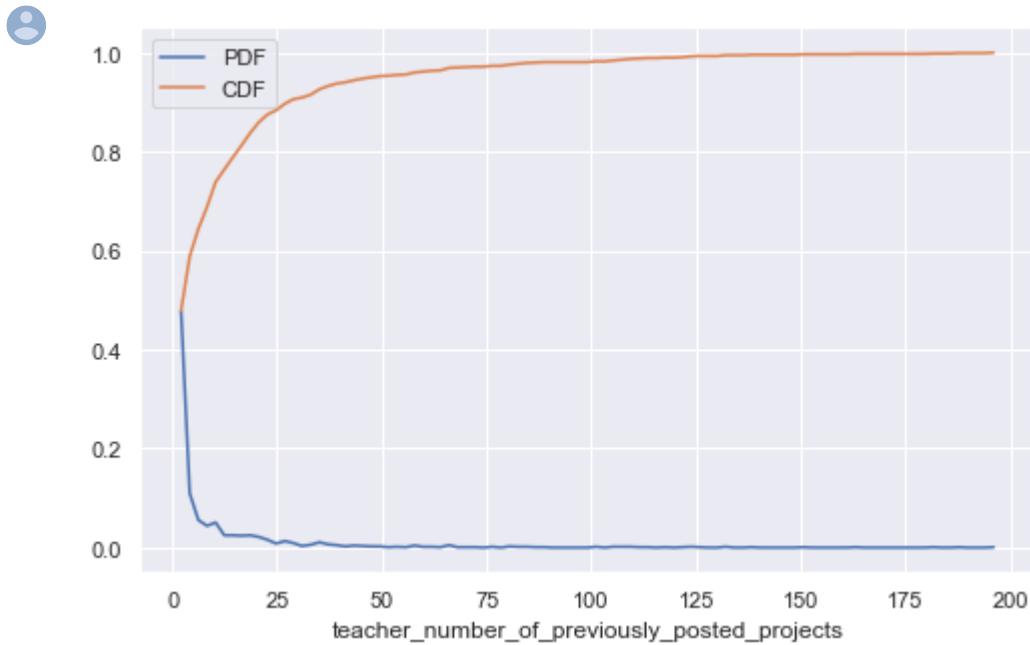


```
#PDF (FP ,teacher_number_of_previously_posted_projects)
plt.figure(figsize=(8,5))
```

```

counts, bin_edges = np.histogram(X_test_falsePos1['teacher_number_of_previously_posted_projects'])
pdf = counts/sum(counts)
cdf = np.cumsum(pdf)
pdfP, = plt.plot(bin_edges[1:], pdf)
cdfP, = plt.plot(bin_edges[1:], cdf)
plt.legend([pdfP, cdfP], ["PDF", "CDF"])
plt.xlabel('teacher_number_of_previously_posted_projects')
plt.show()

```



## ▼ Conclusions

```

# Please compare all your models using Prettytable library
#how to use pretty table http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
tb = PrettyTable()
tb.field_names= ("Vectorizer", "Max_depth", "Min_sample_split","Test -AUC ")
tb.add_row(["BOW", 10, 500, 67])
tb.add_row(["Tf - Idf", 10 , 500 ,66.5 ])
tb.add_row(["AVG-W2V", 5, 500,63.8 ])
tb.add_row(["A VG - Tf - Idf", 5 , 500 ,56.5])
tb.add_row(["Top 5000 Features", 10, 500 ,66.5 ])
print(tb.get_string(titles = "Decision trees- Observations"))

```



Vectorizer	Max_depth	Min_sample_split	Test -AUC
BOW	10	500	67
Tf - Idf	10	500	66.5
AVG-W2V	5	500	63.8
AVG - Tf - Idf	5	500	56.5
Top 5000 Features	10	500	66.5