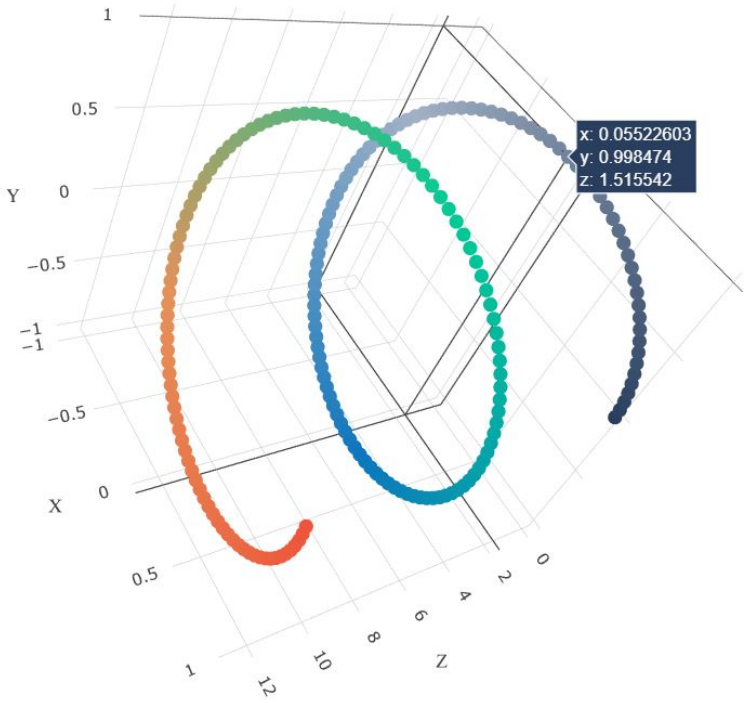
Assignment 8: DT

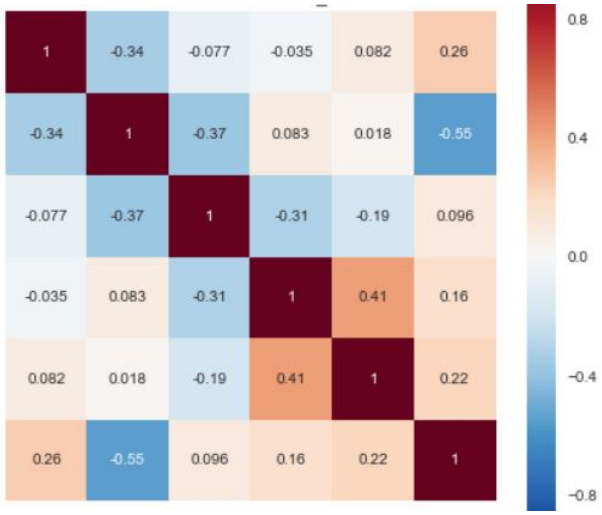
1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_eassay (TFIDF)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (best `depth` in range [1, 5, 10, 50], and the best `min_samples_s**
    - Find the best hyper parameter which will give the maximum AUC value
    - find the best hyper paramter using k-fold cross validation(use gridsearch cv or random
      data(you can write your own for loops refer sample solution)
3. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data



in the figure wit
as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains h
in the same drive *3d_scatter_plot.ipynb*

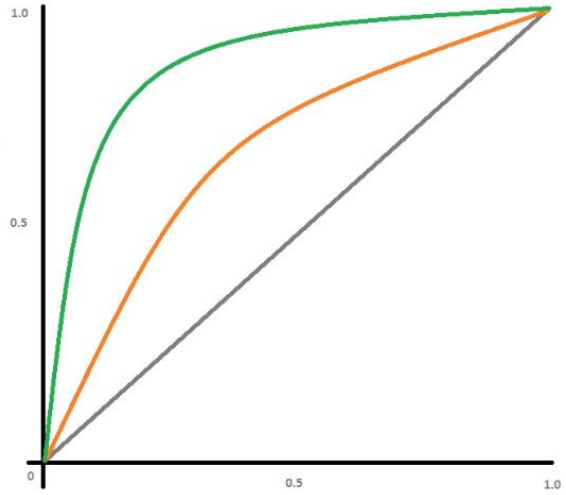
or

- You need to plot the performance of model both on train data and cross validation data



in the figure seaborn heat maps with rows a **max_depth**, and values inside the cell representing **AUC Score**

- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, an



the ROC curve on both train and test.

- Along with plotting ROC curve, you need to print the confusion matrix with predicted an

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

- Once after you plot the confusion matrix with the test data, get all the `false positive da
  - Plot the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-pytho these `false positive data points`
  - Plot the box plot with the `price` of these `false positive data points`
  - Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `f

4. **Task 2:** For this task consider set-1 features. Select all the features which are having non-zer feature importance using 'feature_importances_` (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html), discard the a

apply any of the model of you choice i.e. (Dession tree, Logistic Regression, Linear SVM), you
corresponding to the model you selected and procedure in step 2 and step 3
Note: when you want to find the feature importance make sure you don't use max_depth para

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

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

1. Decision Tree

```
!pip install chart_studio
```

```
Collecting chart_studio
  Downloading https://files.pythonhosted.org/packages/ca/ce/330794a6b6ca4b9182c38fc69
     |████████████████████████████████| 71kB 2.0MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (fr
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from ch
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: plotly in /usr/local/lib/python3.6/dist-packages (from
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-pac
Installing collected packages: chart-studio
Successfully installed chart-studio-1.1.0
```

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

import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
```

```
import pickle
from tqdm import tqdm
import os

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

1.1 Loading Data

```
ZqrZ0gKEV7cCSw9ikIQ?e=download&authuser=0&nonce=qvr4obitd30bs&user=01895839698977569751&ha
```

```
--2020-05-12 08:55:50--  https://doc-0s-c8-docs.googleusercontent.com/docs/securesc/m
Resolving doc-0s-c8-docs.googleusercontent.com (doc-0s-c8-docs.googleusercontent.com)
Connecting to doc-0s-c8-docs.googleusercontent.com (doc-0s-c8-docs.googleusercontent.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/csv]
Saving to: ‘preprocessed_data.csv’

preprocessed_data.c     [     <=>            ] 118.69M  98.3MB/s    in 1.2s

2020-05-12 08:55:52 (98.3 MB/s) - ‘preprocessed_data.csv’ saved [124454659]
```

```
data  = pd.read_csv('preprocessed_data.csv', nrows=50000)
data.head(2)
```

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

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

```
# separating y from dataframe
y = data['project_is_approved'].values
```

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

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously
0	ca	mrs	grades_prek_2	

```
data.columns.values
```

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

```
# separating data into train and test.
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)
```

1.3 Make Data Model Ready: encoding eassay, and project_title

TFIDF vectorizer

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)
vectorizer_tfidf = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer_tfidf.fit(X_train['essay'].values)
X_train_eassy_tfidf = vectorizer_tfidf.fit_transform(X_train['essay'].values)
X_test_eassy_tfidf = vectorizer_tfidf.fit_transform(X_test['essay'].values)
print("After vectorizations")
print(X_train_eassy_tfidf.shape, y_train.shape)
print(X_test_eassy_tfidf.shape, y_test.shape)
print("="*100)
```

```
(33500, 8) (33500,)
(16500, 8) (16500,)
====================================================================================================
After vectorizations
(33500, 5000) (33500,)
(16500, 5000) (16500,)
====================================================================================================
```

TFIDF weighted W2V

```
3vm4jsk2igas9tb89osmm/1589274225000/00484516897554883881/01895839698977569751/1lDca_ge-GYO
```

```
--2020-05-12 09:05:33--  https://doc-0g-c8-docs.googleusercontent.com/docs/securesc/m
Resolving doc-0g-c8-docs.googleusercontent.com (doc-0g-c8-docs.googleusercontent.com)
Connecting to doc-0g-c8-docs.googleusercontent.com (doc-0g-c8-docs.googleusercontent.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/octet-stream]
Saving to: ‘glove_vectors’

glove_vectors           [       <=>          ] 121.60M  46.8MB/s    in 2.6s

2020-05-12 09:05:37 (46.8 MB/s) - ‘glove_vectors’ saved [127506004]
```

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

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

```
# average Word2Vec
# compute average word2vec for each review.
def tf_idf_done(word_list):
  # average Word2Vec
  # compute average word2vec for each review.
  tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
  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(): # 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((sent
              tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # get
              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
```

```
train_tfidf_w2v_essays=tf_idf_done(X_train['essay'].values)
test_tfidf_w2v_essays=tf_idf_done(X_test['essay'].values)
```

```
100%|██████████| 33500/33500 [01:01<00:00, 544.93it/s]
100%|██████████| 16500/16500 [00:30<00:00, 546.06it/s]
```

```
train_tfidf_w2v_essays = np.reshape(train_tfidf_w2v_essays,(len(train_tfidf_w2v_essays),le
test_tfidf_w2v_essays  = np.reshape(test_tfidf_w2v_essays,(len(test_tfidf_w2v_essays),len(
```

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)
print("After vectorizations")
print(train_tfidf_w2v_essays.shape,y_train.shape)
print(test_tfidf_w2v_essays.shape,y_test.shape)
print("="*100)
```

```
(33500, 8) (33500,)
(16500, 8) (16500,)
====================================================================================================
After vectorizations
(33500, 300) (33500,)
(16500, 300) (16500,)
====================================================================================================
```

Make Data Model Ready: encoding numerical, categorical fe

encoding categorical features: School State

```
vectorizer_school_state = CountVectorizer()
vectorizer_school_state.fit(X_train['school_state'].values) # fit has to happen only on tr

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

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer_school_state.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 51) (33500,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id',
====================================================================================================
```

encoding categorical features: teacher_prefix

```
vectorizer_teacher_prefix = CountVectorizer()
vectorizer_teacher_prefix.fit(X_train['teacher_prefix'].values) # fit has to happen only o

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

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer_teacher_prefix.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 5) (33500,)
(16500, 5) (16500,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

encoding categorical features: project_grade_category

```
vectorizer_project_grade_category = CountVectorizer()
vectorizer_project_grade_category.fit(X_train['project_grade_category'].values) # fit has

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer_project_grade_category.transform(X_train['project_grade_cat
X_test_grade_ohe = vectorizer_project_grade_category.transform(X_test['project_grade_categ

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer_project_grade_category.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 4) (33500,)
(16500, 4) (16500,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================================
```

encoding categorical features: clean_categories

```
vectorizer_clean_categories = CountVectorizer()
vectorizer_clean_categories.fit(X_train['clean_categories'].values) # fit has to happen on

# we use the fitted CountVectorizer to convert the text to vector
X_train_category_ohe= vectorizer_clean_categories.transform(X_train['clean_categories'].va
X_test_category_ohe = vectorizer_clean_categories.transform(X_test['clean_categories'].val

print("After vectorizations")
print(X_train_category_ohe.shape, y_train.shape)
print(X_test_category_ohe.shape, y_test.shape)
```

```
print(X_test_category_ohe.shape, y_test.shape)
print(vectorizer_clean_categories.get_feature_names())
print("="*100)
```

```
After vectorizations
(33500, 7) (33500,)
(16500, 7) (16500,)
['appliedlearning', 'health_sports', 'history_civics', 'literacy_language', 'math_sci
====================================================================================
```

encoding categorical features: clean_subcategories

```
vectorizer_clean_subcategories = CountVectorizer()
vectorizer_clean_subcategories.fit(X_train['clean_subcategories'].values) # fit has to hap

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategory_ohe = vectorizer_clean_subcategories.transform(X_train['clean_subcateg
X_test_subcategory_ohe = vectorizer_clean_subcategories.transform(X_test['clean_subcategor

print("After vectorizations")
print(X_train_subcategory_ohe.shape, y_train.shape)
print(X_test_subcategory_ohe.shape, y_test.shape)
print(vectorizer_clean_subcategories.get_feature_names())
print("="*100)
```

```
 vectorizations
00, 28) (33500,)
00, 28) (16500,)
liedsciences', 'charactereducation', 'civics_government', 'college_careerprep', 'commu
==========================================================================================
```

Encoding numerical features: Price

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['price'].values.reshape(1,-1))

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



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

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================
```

Encoding numerical features : teacher_number_of_previously_posted_projects

```
normalizer = Normalizer()

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1

X_train_teacher_posted_projects_norm = normalizer.transform(X_train['teacher_number_of_pre
X_test_teacher_posted_projects_norm = normalizer.transform(X_test['teacher_number_of_previ



print("After vectorizations")
print(X_train_teacher_posted_projects_norm.shape, y_train.shape)
print(X_test_teacher_posted_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

Concatinating all the features

data set1 with tfidf

```
from scipy.sparse import hstack
X_tr_set1 = hstack((X_train_eassy_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_g
X_te_set1 = hstack((X_test_eassy_tfidf, X_test_state_ohe, X_test_teacher_ohe, X_test_grade

print("Final Data matrix")
print(X_tr_set1.shape, y_train.shape)
print(X_te_set1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 5097) (33500,)
(16500, 5097) (16500,)
====================================================================================================
```

data set2 with tfidf_weighted_w2v

```
from scipy.sparse import hstack
X_tr_set2= hstack((train_tfidf_w2v_essays, X_train_state_ohe, X_train_teacher_ohe, X_train
X_te_set2 = hstack((test_tfidf_w2v_essays, X_test_state_ohe, X_test_teacher_ohe, X_test_gr

print("Final Data matrix")
print(X_tr_set1.shape, y_train.shape)
print(X_te_set1.shape, y_test.shape)
```

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

```
Final Data matrix
(33500, 5097) (33500,)
(16500, 5097) (16500,)
====================================================================================================
```

1.5 Appling Decision Tree on different kind of featurization as mentic

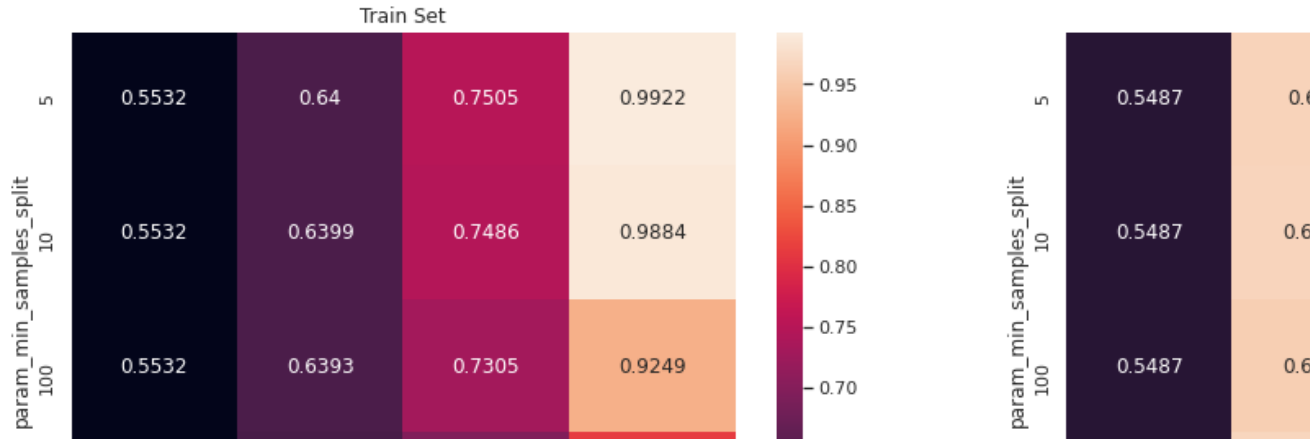
Apply Decision Tree on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

Appling Decision Tree on different on data set1(with tf_idf)

```
from sklearn.metrics import roc_auc_score
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], 'min_samples_split': [5, 10, 100, 500]}
clf1 = GridSearchCV(dt1, parameters, cv=3, scoring='roc_auc',return_train_score=True)
se1 = clf1.fit(X_tr_set1, y_train)
```

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

```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(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()
```



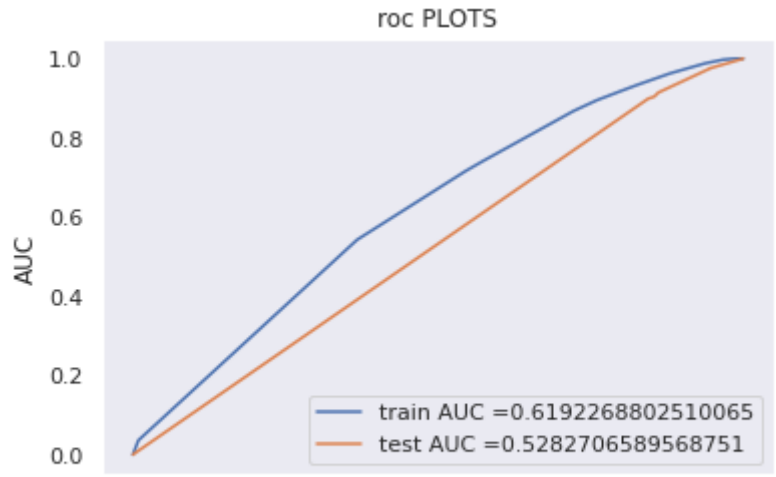

```
# best_parameters
print(clf1.best_params_)
```

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

```
# best_estimitor
clf1.best_estimator_
```

```
DecisionTreeClassifier(ccp_alpha=0.0, 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='deprecated',
                       random_state=None, splitter='best')
```

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn
from sklearn.metrics import roc_curve, auc
best_tune_parameters=[{'max_depth':[10], 'min_samples_split':[500] } ]

clf11= DecisionTreeClassifier(max_depth=10,random_state=0 ,min_samples_split=500,criterion
clf11.fit(X_tr_set1, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs

y_train_pred= clf11.predict_proba(X_tr_set1)[:,1]
y_test_pred = clf11.predict_proba(X_te_set1)[:,1]


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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("roc PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix

```
# 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(
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1= predictions
    return predictions


print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.34325782809583205 for threshold 0.881
Train confusion matrix
[[ 3394  1971]
 [12869 15266]]
Test confusion matrix
[[  411  2231]
 [ 1401 12457]]
```

false positive data point analysis

WORD CLOUD OF ESSAY

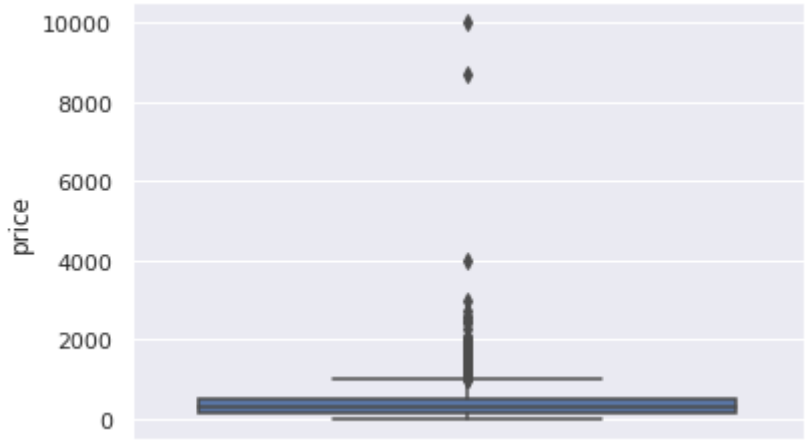
```
predictions1=predict_with_best_t(y_test_pred, best_t)
FPI= []
for i in range(len(y_test)) :
  if (y_test[i] == 0) & (predictions1[i]==1):

    FPI.append(i)
FP_essay =[]
for i in FPI :
  FP_essay.append(X_test['essay'].values[i])


from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for val in FP_essay:
  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 = st

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

box plot of price

```
predictions1=predict_with_best_t(y_test_pred, best_t)
FPI= []
for i in range(len(y_test)) :
  if (y_test[i] == 0) & (predictions1[i]==1):

    FPI.append(i)
FP_price =[]
for i in FPI :
  FP_price.append(X_test['price'].values[i])
```

```
import seaborn as sns
FP_price1= pd.DataFrame({"price":FP_price})
sns.boxplot(y='price', data=FP_price1)
print(FP_price1.shape)
```

```
(2231, 1)
```



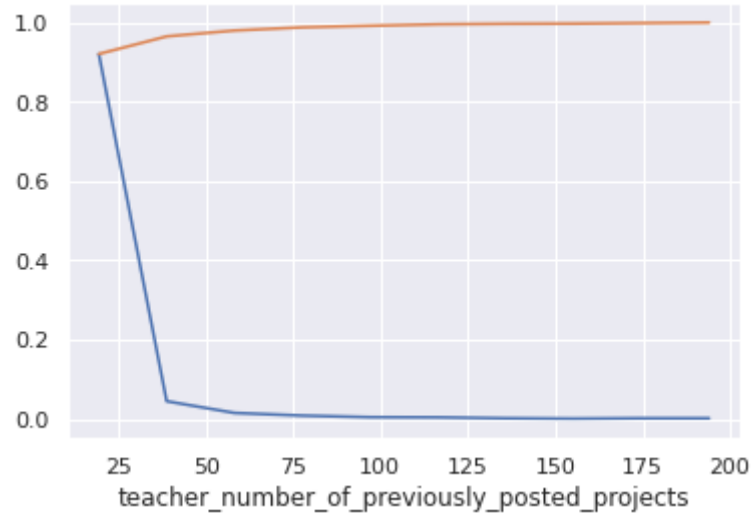
pdf of FP_teacher_number_of_previously_posted_projects

```
predictions1=predict_with_best_t(y_test_pred, best_t)
FPI= []
for i in range(len(y_test)) :
  if (y_test[i] == 0) & (predictions1[i]==1):

    FPI.append(i)
FP_teacher_number_of_previously_posted_projects =[]
for i in FPI :
```

```
    FP_teacher_number_of_previously_posted_projects.append(X_test['teacher_number_of_previou
```

```
counts, bin_edges = np.histogram(FP_teacher_number_of_previously_posted_projects, bins=10,
                                 density = True)
pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)

#compute CDF
plt.xlabel('teacher_number_of_previously_posted_projects')
cdf = np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf)
plt.plot(bin_edges[1:], cdf)
```

```
[9.20663380e-01 4.43747199e-02 1.47915733e-02 8.06813088e-03
 4.03406544e-03 3.58583595e-03 1.34468848e-03 4.48229494e-04
 1.34468848e-03 1.34468848e-03]
[  0.   19.4  38.8  58.2  77.6  97.  116.4 135.8 155.2 174.6 194. ]
[<matplotlib.lines.Line2D at 0x7fe764bbdcf8>]
```

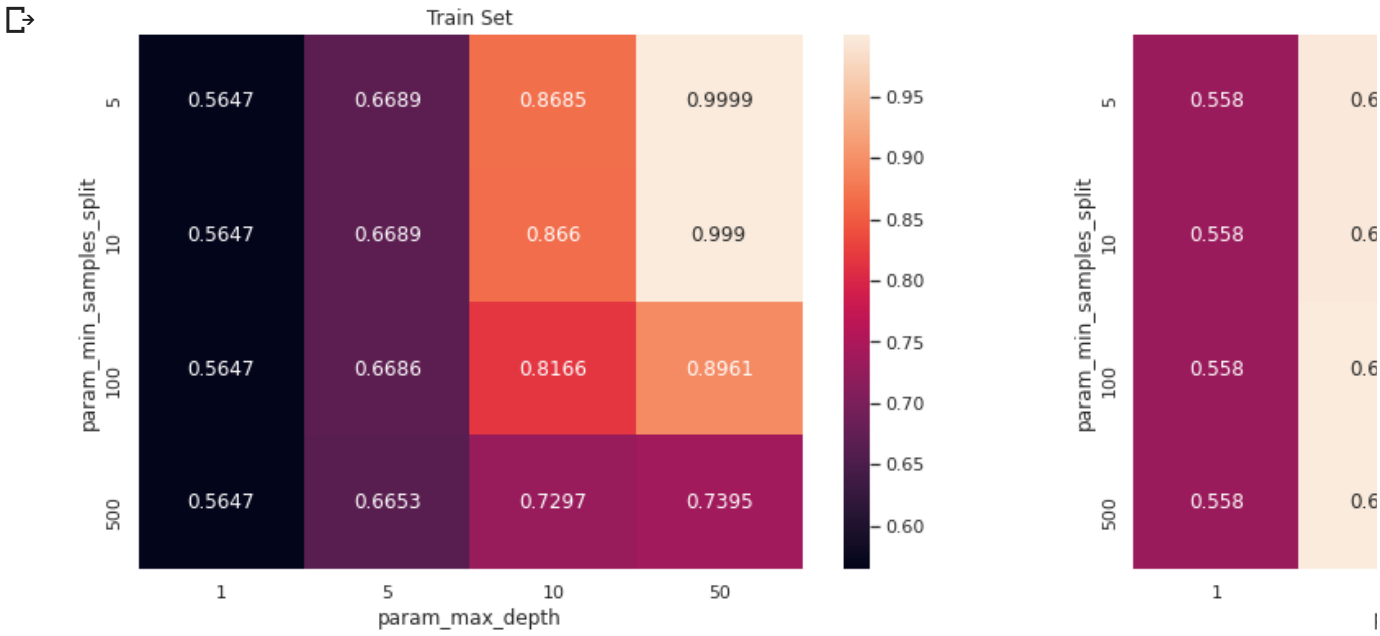


Apply Decision Tree on data set2(with tfidf_weighted_w2v)

```
from sklearn.metrics import roc_auc_score
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], 'min_samples_split': [5, 10, 100, 500]}
clf2 = GridSearchCV(dt2, parameters, cv=3, scoring='roc_auc',return_train_score=True)
set2= clf2.fit(X_tr_set2, y_train)
```

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

```
#results = results.sort_values(['param_min_samples_split'])
max_depth=  results['param_max_depth']
min_samples_split= results['param_min_samples_split']


import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(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()
```

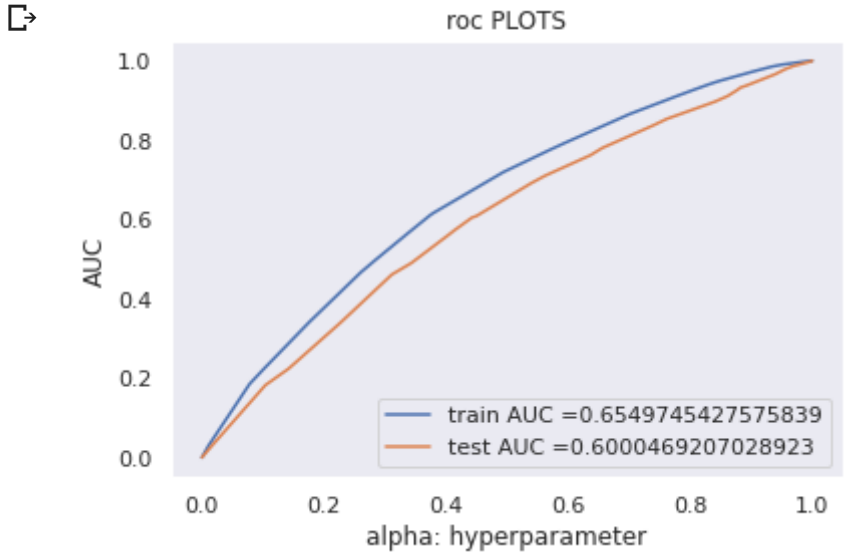



```
#best parameter
print(clf2.best_params_)
```

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

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn
from sklearn.metrics import roc_curve, auc
best_tune_parameters=[{'max_depth':[5], 'min_samples_split':[500] } ]

clf22 = DecisionTreeClassifier(max_depth=5,random_state=0 ,min_samples_split=500,criterion
clf22.fit(X_tr_set2, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs

y_train_pred= clf22.predict_proba(X_tr_set2)[:,1]
y_test_pred = clf22.predict_proba(X_te_set2)[:,1]


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

```
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("roc PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix

```
# 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(
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1= predictions
    return predictions


print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

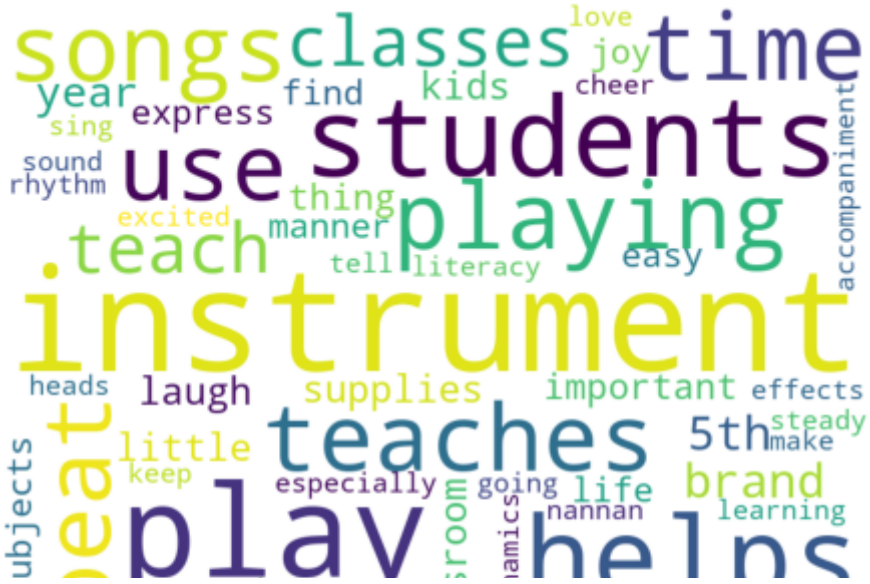
```
==================================================================================================
the maximum value of tpr*(1-fpr) 0.4314059079087299 for threshold 0.841
Train confusion matrix
[[ 3463  1902]
 [ 9331 18804]]
Test confusion matrix
[[1355 1287]
 [4938 8920]]
```

false positive data point analysis

WORD CLOUD OF ESSAY

```
predictions1=predict_with_best_t(y_test_pred, best_t)
FPI= []
for i in range(len(y_test)) :
  if (y_test[i] == 0) & (predictions1[i]==1):

    FPI.append(i)
FP_essay =[]
for i in FPI :
  FP_essay.append(X_test['essay'].values[i])
```

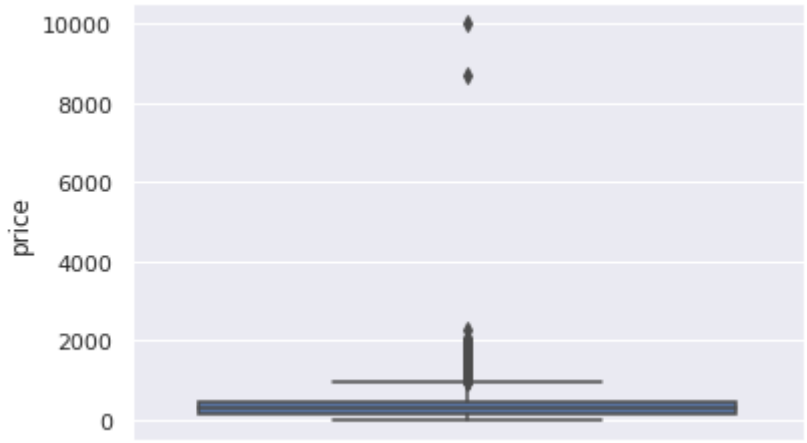
```
om wordcloud import WordCloud, STOPWORDS
mment_words = ' '
opwords = set(STOPWORDS)
r val in FP_essay:
val = str(val)
tokens = val.split()
r i in range(len(tokens)):
tokens[i] = tokens[i].lower()
r words in tokens :
comment_words = comment_words + words + ' '
rdcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = stop

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



box plot of fp_price

```
predictions1=predict_with_best_t(y_test_pred, best_t)
FPI= []
for i in range(len(y_test)) :
  if (y_test[i] == 0) & (predictions1[i]==1):

    FPI.append(i)
FP_price =[]
for i in FPI :
  FP_price.append(X_test['price'].values[i])


import seaborn as sns
FP_price1= pd.DataFrame({"price":FP_price})
sns.boxplot(y='price', data=FP_price1)
print(FP_price1.shape)
```

```
(1287, 1)
```



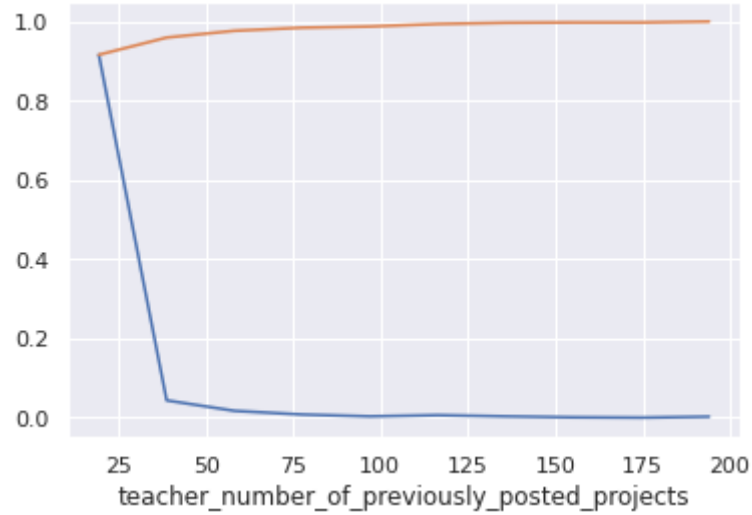
pdf of FP_teacher_number_of_previously_posted_projects

```
predictions1=predict_with_best_t(y_test_pred, best_t)
```

```
FPI= []
for i in range(len(y_test)) :
  if (y_test[i] == 0) & (predictions1[i]==1):

    FPI.append(i)
FP_teacher_number_of_previously_posted_projects =[]
for i in FPI :
  FP_teacher_number_of_previously_posted_projects.append(X_test['teacher_number_of_previou
```

```
counts, bin_edges = np.histogram(FP_teacher_number_of_previously_posted_projects, bins=10,
                                 density = True)
pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)

#compute CDF
plt.xlabel('teacher_number_of_previously_posted_projects')
cdf = np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf)
plt.plot(bin_edges[1:], cdf)
```

```
[9.16083916e-01 4.35120435e-02 1.70940171e-02 7.77000777e-03
 3.10800311e-03 6.21600622e-03 3.10800311e-03 7.77000777e-04
 0.00000000e+00 2.33100233e-03]
[  0.   19.4  38.8  58.2  77.6  97.  116.4 135.8 155.2 174.6 194. ]
[<matplotlib.lines.Line2D at 0x7fe77dd84278>]
```



1.6 Getting top features using `feature_importances_(of data set1(w

```
c=clf11.feature_importances_
f_i_i= []
for i,j in enumerate(c):
  if j !=0:
```

```
    f_i_i.append(i)
print(len(f_i_i))
```

```
23
```

```
X_tr_new = X_tr_set1[:,f_i_i]
X_te_new = X_te_set1[:,f_i_i]
print("new Data matrix")
print(X_tr_new.shape)
print(X_te_new.shape)
```

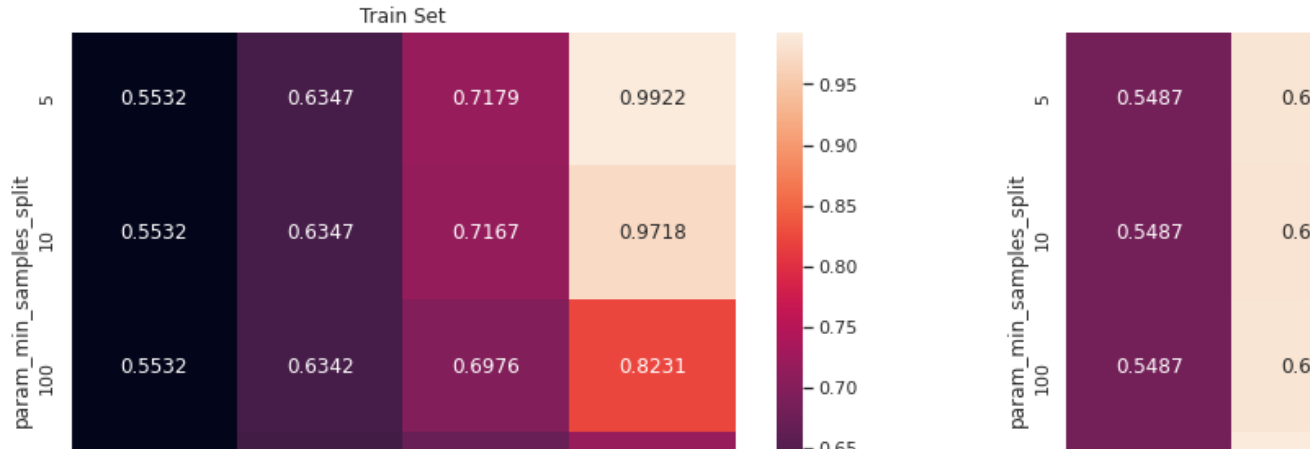
```
new Data matrix
(33500, 23)
(16500, 23)
```

Applying Decision trees on non-zero feature importance

```
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
dt_new= DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth':[1, 5, 10, 50], 'min_samples_split': [5, 10, 100, 500]}
clf_new= GridSearchCV(dt_new, parameters, cv=3, scoring='roc_auc',return_train_score=True)
set_new = clf_new.fit(X_tr_new, y_train)
```

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

```
import seaborn as sns; sns.set()
max_scores1 = pd.DataFrame(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()
```




```
clf_new.best_params_
```

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

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn
from sklearn.metrics import roc_curve, auc
best_tune_parameters=[{'max_depth':[5], 'min_samples_split':[500] } ]

clf_new1= DecisionTreeClassifier(max_depth=5,random_state=0 ,min_samples_split=500,criteri
clf_new1.fit(X_tr_new, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the
# not the predicted outputs

y_train_pred= clf_new1.predict_proba(X_tr_new)[:,1]
y_test_pred = clf_new1.predict_proba(X_te_new)[:,1]


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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("roc PLOTS")
plt.grid()
plt.show()
```


```
# 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(
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    predictions1= predictions
    return predictions


print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

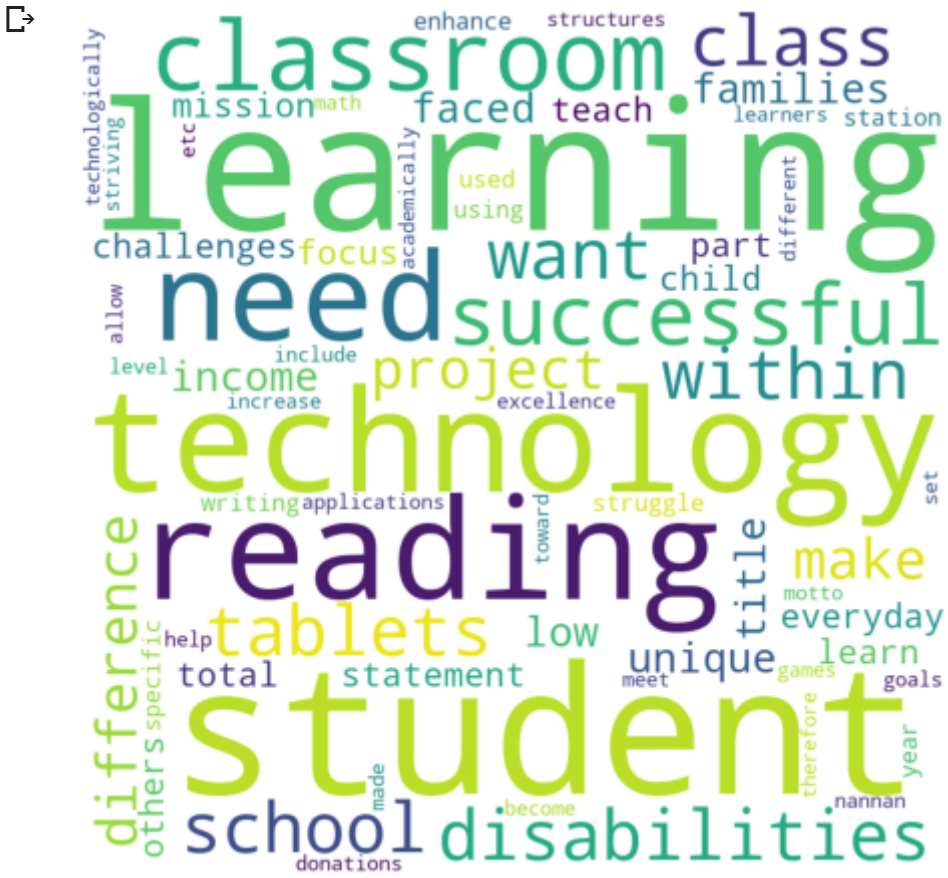
```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.34325782809583205 for threshold 0.881
Train confusion matrix
[[ 3394  1971]
 [12869 15266]]
Test confusion matrix
[[  411  2231]
 [ 1401 12457]]
```

false positive data points'

WordCloud of FP_essay

```
predictions1=predict_with_best_t(y_test_pred, best_t)
FPI= []
for i in range(len(y_test)) :
  if (y_test[i] == 0) & (predictions1[i]==1):

    FPI.append(i)
FP_essay =[]
for i in FPI :
  FP_essay.append(X_test['essay'].values[i])
```

```
from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)
for val in FP_essay:
  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 = st

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



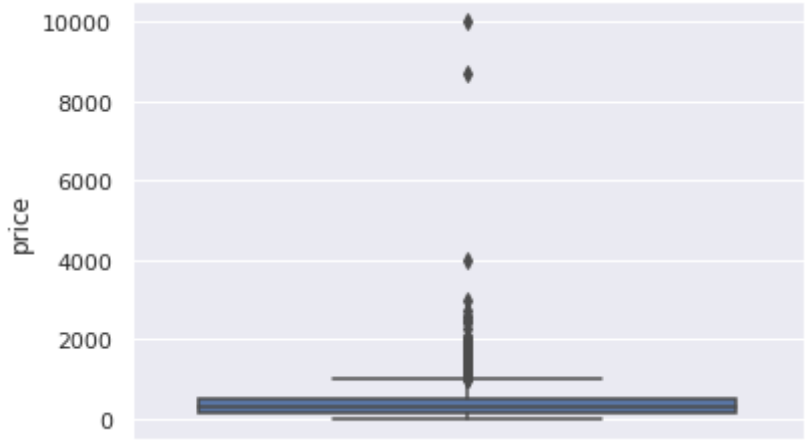
box plot of FP_price

```
predictions1=predict_with_best_t(y_test_pred, best_t)
FPI= []
for i in range(len(y_test)) :
  if (y_test[i] == 0) & (predictions1[i]==1):

    FPI.append(i)
FP_price =[]
```

```
for i in FPI :
  FP_price.append(X_test['price'].values[i])
```

```
import seaborn as sns
FP_price1= pd.DataFrame({"price":FP_price})
sns.boxplot(y='price', data=FP_price1)
print(FP_price1.shape)
```

```
(2231, 1)
```

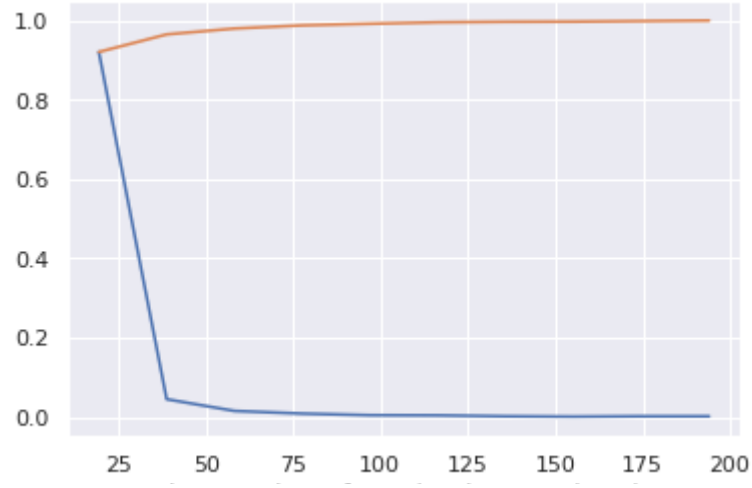


pdf of FP_teacher_number_of_previously_posted_projects

```
predictions1=predict_with_best_t(y_test_pred, best_t)
FPI= []
for i in range(len(y_test)) :
  if (y_test[i] == 0) & (predictions1[i]==1):

    FPI.append(i)
FP_teacher_number_of_previously_posted_projects =[]
for i in FPI :
  FP_teacher_number_of_previously_posted_projects.append(X_test['teacher_number_of_previous
```

```
counts, bin_edges = np.histogram(FP_teacher_number_of_previously_posted_projects, bins=10,
                                 density = True)
pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)

#compute CDF
plt.xlabel('teacher_number_of_previously_posted_projects')
cdf = np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf)
plt.plot(bin_edges[1:], cdf)
```

```
[9.20663380e-01 4.43747199e-02 1.47915733e-02 8.06813088e-03
 4.03406544e-03 3.58583595e-03 1.34468848e-03 4.48229494e-04
 1.34468848e-03 1.34468848e-03]
[  0.   19.4  38.8  58.2  77.6  97.  116.4 135.8 155.2 174.6 194. ]
[<matplotlib.lines.Line2D at 0x7fe782db6b00>]
```




```
# please write all the code with proper documentation, and proper titles for each subsecti
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your co
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2. Summary

```
 from tabulate import tabulate
weather_data = [('TFIDF', 'Decision trees ',"'max_depth': 10, 'min_samples_split': 500", 0
                ('TFIDF_weighted_w2v', 'Decision trees ', "'max_depth': 5, 'min_samples_sp
                ('TFIDF_weighted_w2v', 'De',"'max_depth': 5, 'min_samples_split': 500", 0.
               ]
df = pd.DataFrame(weather_data, columns=['Vectorizer', 'Model', 'Hyper parameter', 'AUC'])


#Ref: https://pypi.org/project/tabulate/
print(tabulate(df, headers='keys', tablefmt='github'))
```

```
|    | Vectorizer          | Model          | Hyper parameter
|----|---------------------|----------------|-----------------------------------------
|  0 | TFIDF               | Decision trees | 'max_depth': 10, 'min_samples_split': 50
|  1 | TFIDF_weighted_w2v  | Decision trees | 'max_depth': 5, 'min_samples_split': 500
|  2 | TFIDF_weighted_w2v  | De             | 'max_depth': 5, 'min_samples_split': 500
```