```
%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
import os
```

```
from google.colab import files
files=files.upload()
```

Choose Files preprocessed_data.csv

- **preprocessed_data.csv**(application/vnd.ms-excel) - 124454659 bytes, last modified: 11/11/2019 - 100% done

Saving preprocessed_data.csv to preprocessed_data.csv

```
preprocessed_data = pd.read_csv('preprocessed_data.csv')
preprocessed_data.head(3)
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previousl
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	
2	ca	mrs	grades_prek_2	

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

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previousl
0	ca	mrs	grades_prek_2	

Splitting the Data into Train and Test : Stratified Sampling

```
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)
```

Encoding Using Tfidf: "essay"

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4)) #Apply Tfidf Vectorizer
vectorizer.fit(X_train['essay'].values)

X_train_essay_Tfidf = vectorizer.transform(X_train['essay'].values)
X_test_essay_Tfidf= vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
```

```
print("After vectorizations")
print(X_train_essay_Tfidf.shape, y_train.shape)
print(X_test_essay_Tfidf.shape, y_test.shape)
```

```
After vectorizations
(73196, 258959) (73196,)
(36052, 258959) (36052,)
```

Encoding using tfidf: "clean_subcategories"

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

X_train_clean_subcategories = vectorizer.transform(X_train['clean_subcategories'].values)
X_test_clean_subcategories = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_clean_subcategories.shape, y_train.shape)
print(X_test_clean_subcategories.shape, y_test.shape)
```

```
After vectorizations
(73196, 245) (73196,)
(36052, 245) (36052,)
```

Encoding using tfidf: "clean_categories"

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

X_train_clean_categories = vectorizer.transform(X_train['clean_categories'].values)
X_test_clean_categories = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_categories.shape, y_train.shape)
print(X_test_clean_categories.shape, y_test.shape)
```

```
After vectorizations
(73196, 48) (73196,)
(36052, 48) (36052,)
```

Encoding using tfidf: "project_grade_category"

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

X_train_project_grade_category = vectorizer.transform(X_train['project_grade_category'].va
X_test_project_grade_category = vectorizer.transform(X_test['project_grade_category'].valu

print("After vectorizations")
print(X_train_project_grade_category.shape, y_train.shape)
print(X_test_project_grade_category.shape, y_test.shape)
```

```
print(X_test_project_grade_category.shape, y_test.shape)
```

```
After vectorizations
(73196, 4) (73196,)
(36052, 4) (36052,)
```

Encoding using tfidf: "teacher_prefix"

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

X_train_teacher_prefix = vectorizer.transform(X_train['teacher_prefix'].values)
X_test_teacher_prefix = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_prefix.shape, y_train.shape)
print(X_test_teacher_prefix.shape, y_test.shape)
```

```
After vectorizations
(73196, 5) (73196,)
(36052, 5) (36052,)
```

Encoding using tfidf: "school_state"

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

X_train_school_state = vectorizer.transform(X_train['school_state'].values)
X_test_school_state = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_school_state.shape, y_train.shape)
print(X_test_school_state.shape, y_test.shape)
```

```
After vectorizations
(73196, 51) (73196,)
(36052, 51) (36052,)
```

Encoding Numerical features using tfidf: "teacher_number_of_previously_posted_projects"

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

X_train_project_teachers_norm = normalizer.transform(X_train['teacher_number_of_previously
X_test_project_teachers_norm = normalizer.transform(X_test['teacher_number_of_previously_p
```

```
print("After vectorizations")
print(X_train_project_teachers_norm.shape, y_train.shape)
print(X_test_project_teachers_norm.shape, y_test.shape)
```

```
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Encoding Numerical features using tfidf: "price"

```
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)
```

```
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
```

Concatenating all the Features:(Tfidf)

```
from scipy.sparse import hstack

X_train1 = hstack((X_train_essay_Tfidf, X_train_clean_categories, X_train_clean_subcategor
X_test1 = hstack((X_test_essay_Tfidf, X_test_clean_categories, X_test_clean_subcategories,

print("Final Data matrix")
print(X_train1.shape, y_train.shape)
print(X_test1.shape, y_test.shape)
```

```
Final Data matrix
(73196, 259314) (73196,)
(36052, 259314) (36052,)
```

Implementing Decision Tree Using Tfidf

```
import math as mt
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
```

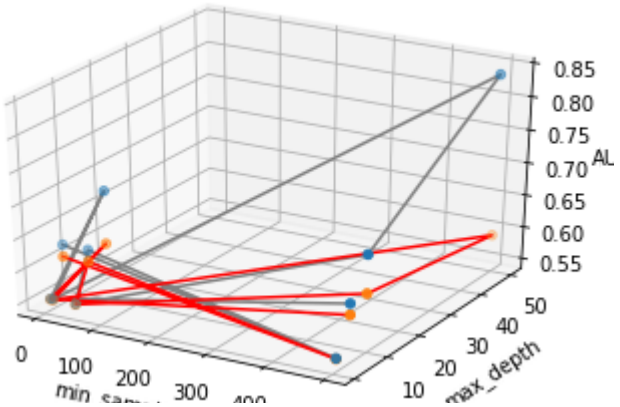
```
#The “balanced” mode uses the values of class to automatically adjust weights inversely pr
#to class frequencies in the input data as n_samples / (n_classes * np.bincount(y))

Decision_Tree = DecisionTreeClassifier(class_weight='balanced')
parameters = {'max_depth':[1,5,10,50],'min_samples_split':[5,10,50,500]}
clf = RandomizedSearchCV(Decision_Tree, parameters, cv=5, scoring='roc_auc',return_train_s
clf.fit(X_train1, y_train)

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


#Ref:https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedS
train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
max_depth= results['param_max_depth']
min_samples_split=results['param_min_samples_split']


print("Train AUC= ",train_auc)
print(50*'-')
print("Std Train Score= ",train_auc_std)
print(50*'-')
print("CV AUC= ",cv_auc)
print(50*'-')
print("CV AUC STD=",cv_auc_std)
print(50*'-')
print("Maximum Depth of the Tree=",max_depth)
print(50*'-')
print("Minimum Samples Split= ",min_samples_split)
```

```
Std Train Score=  0    0.002492
1    0.001432
2    0.002508
3    0.001432
4    0.001688
5    0.003531
6    0.001432
7    0.003512
8    0.001432
9    0.002825
Name: std_train_score, dtype: float64
--------------------------------------------------
CV AUC=  0    0.608541

1    0.554361
2    0.608513
3    0.554361
4    0.624527
5    0.591165
6    0.554361
7    0.621141
8    0.554361
9    0.608347
Name: mean_test_score, dtype: float64
--------------------------------------------------
CV AUC STD= 0    0.006971
1    0.007205
```

```
2    0.007126
3    0.007205
4    0.006944
5    0.007261
6    0.007205
7    0.008863
8    0.007205
9    0.006893
Name: std_test_score, dtype: float64
--------------------------------------------------
Maximum Depth of the Tree= 0     5
1     1
2     5
3     1
4    10
5    50
6     1
7    10
8     1
9     5
Name: param_max_depth, dtype: object
--------------------------------------------------
Minimum Samples Split=  0      5
1    500
2     50
3     50
4    500
5    500
6     10
7     50
8      5
9    500
Name: param_min_samples_split, dtype: object
```

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

```
from mpl_toolkits import mplot3d
%matplotlib inline
import matplotlib.pyplot as plt

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter3D(min_samples_split, max_depth, train_auc, cmap="Black")
ax.plot3D(min_samples_split, max_depth, train_auc, 'gray')
ax.set_xlabel('min_samples_split')
ax.set_ylabel('max_depth')
ax.set_zlabel('AUC');
ax.scatter3D(min_samples_split, max_depth, cv_auc, cmap="Green")
ax.plot3D(min_samples_split, max_depth, cv_auc, 'Red')
plt.show()
```



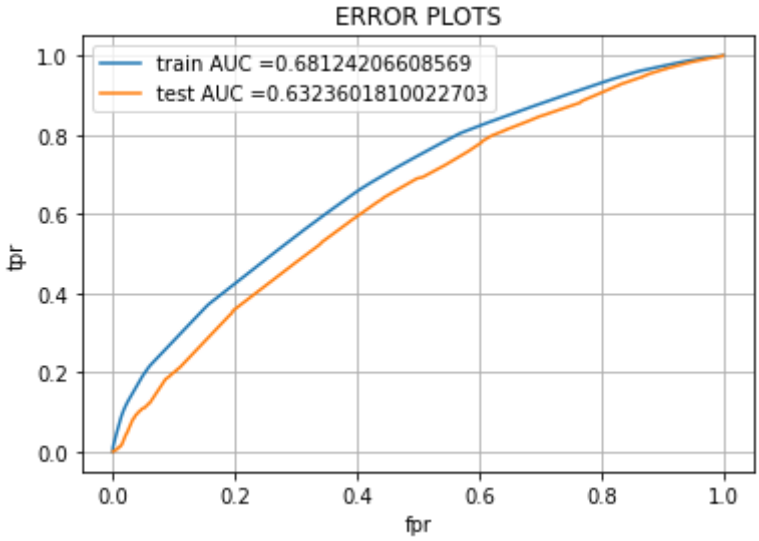

```
clf.best_estimator_
```

```
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')
```

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 73196, then your tr_loop will be 73196 - 73196%1000 = 730
    # in this for loop we will iterate until the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

```
from sklearn.metrics import roc_curve, auc

Model= DecisionTreeClassifier(max_depth=10,min_samples_split=500,class_weight='balanced')
Model.fit(X_train1, y_train)

y_train_pred = batch_predict(Model, X_train1)
y_test_pred = batch_predict(Model, X_test1)

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("fpr")
plt.ylabel("tpr")
plt.title("ERROR PLOTS")
```

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




```
import numpy as np

def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(
    return t

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


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("-"*50)
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) 0.39434212554994996 for threshold 0.525
Train confusion matrix
[[ 6685  4398]
 [21505 40608]]
--------------------------------------------------
Test confusion matrix
[[ 3102  2357]
 [11365 19228]]
```

Now, Implementing Decision Tree on TFidf w2v

```
from google.colab import files
files=files.upload()
```

```
Choose Files glove_vectors
• glove_vectors(n/a) - 127506004 bytes, last modified: 9/29/2021 - 100% done
Saving glove_vectors to glove_vectors
```

```
#please use below code to load glove vectors
import pickle
with open('glove_vectors', 'rb') as f:
  model = pickle.load(f)
  glove_words =  set(model.keys())
```

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'])
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

```
#Computing tfidf_w2v:

from tqdm import tqdm
tfidf_w2v_vectors = []
for sentence in tqdm(X_train['essay'].values):
  vector = np.zeros(300)
  tf_idf_weight=0;
  for word in sentence.split():
    if (word in glove_words) and (word in tfidf_words):
      vec = model[word]

      tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
      vector += (vec * tf_idf)
      tf_idf_weight += tf_idf
  if tf_idf_weight != 0:
    vector /= tf_idf_weight
  tfidf_w2v_vectors.append(vector)

print(len(tfidf_w2v_vectors))
print(len(tfidf_w2v_vectors[0]))
```

```
100%|██████████| 73196/73196 [02:37<00:00, 465.37it/s]73196
300
```

```
tfidf_w2v_test = [];
for sentence in tqdm(X_test['essay'].values):
  vector = np.zeros(300)
  tf_idf_weight =0;
```

```
    for word in sentence.split():
      if (word in glove_words) and (word in tfidf_words):
        vec = model[word]

        tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split()))
        vector += (vec * tf_idf)
        tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
      vector /= tf_idf_weight
    tfidf_w2v_test.append(vector)

print(len(tfidf_w2v_test))
print(len(tfidf_w2v_test[0]))
```

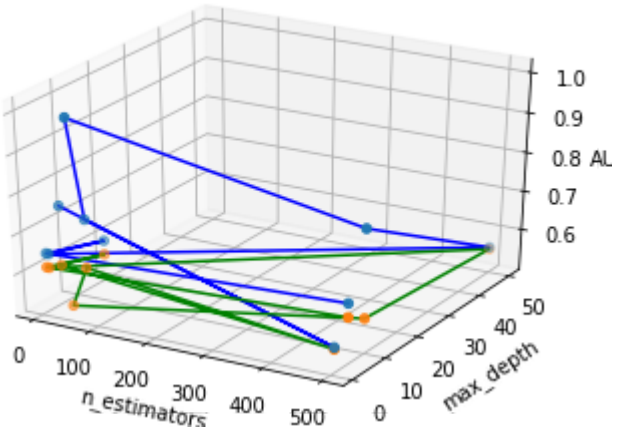
```
100%|██████████| 36052/36052 [01:17<00:00, 467.97it/s]36052
300
```

```
#Convert into Sparse Matrix:
X_tr1_w2v= hstack((tfidf_w2v_vectors, X_train_clean_categories, X_train_clean_subcategorie
X_te1_w2v= hstack((tfidf_w2v_test, X_test_clean_categories, X_test_clean_subcategories, X_

print("Final Data matrix")
print(X_tr1_w2v.shape, y_train.shape)
print(X_te1_w2v.shape, y_test.shape)
```

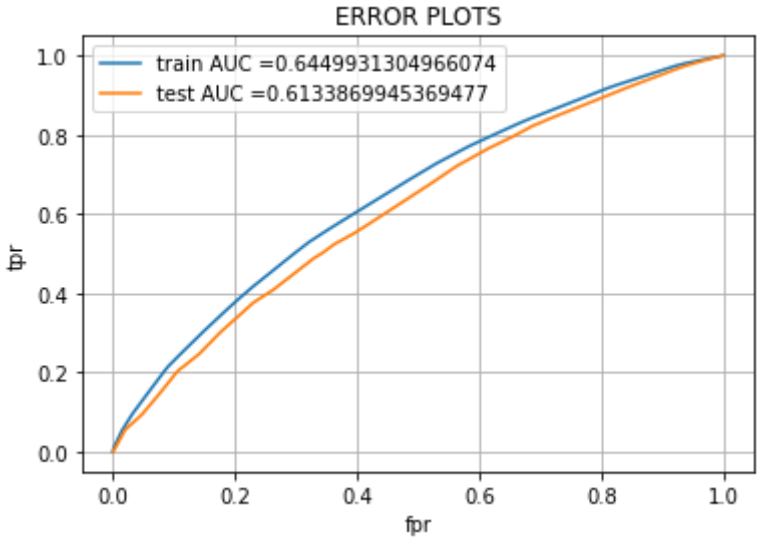
```
Final Data matrix
(73196, 655) (73196,)
(36052, 655) (36052,)
```

```
Model1=DecisionTreeClassifier(max_depth=5,min_samples_split=50,class_weight='balanced')


#Computing results
clf.fit(X_tr1_w2v, y_train)

results1= pd.DataFrame.from_dict(clf.cv_results_)
train_auc1= results1['mean_train_score']
train_auc_std1= results1['std_train_score']
cv_auc1= results1['mean_test_score']
cv_auc_std1= results1['std_test_score']
max_depth1= results1['param_max_depth']
min_samples_spli1t=results1['param_min_samples_split']


#Plotting ROCAUC Curve:
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter3D(min_samples_split, max_depth, train_auc1, cmap="Red")
ax.plot3D(min_samples_split, max_depth, train_auc1, 'blue')
ax.set_xlabel('n_estimators')
ax.set_ylabel('max_depth')
ax.set_zlabel('AUC');
ax.scatter3D(min_samples_split, max_depth, cv_auc1, cmap="Orange")
```

```
ax.plot3D(min_samples_split, max_depth, cv_auc1, 'Green')
plt.show()
```




```
#Plotting Error Plot:
Model1.fit(X_tr1_w2v,y_train)
y_train_pred1 = batch_predict(Model1, X_tr1_w2v)
y_test_pred1= batch_predict(Model1, X_te1_w2v)

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("fpr")
plt.ylabel("tpr")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```




```
#Build Confusion Matrix:
best_t1= find_best_threshold(tr_thresholds1, train_fpr1, train_tpr1)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred1, best_t1)))
print("Test confusion matrix")
```

```
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred1, best_t1)))
```

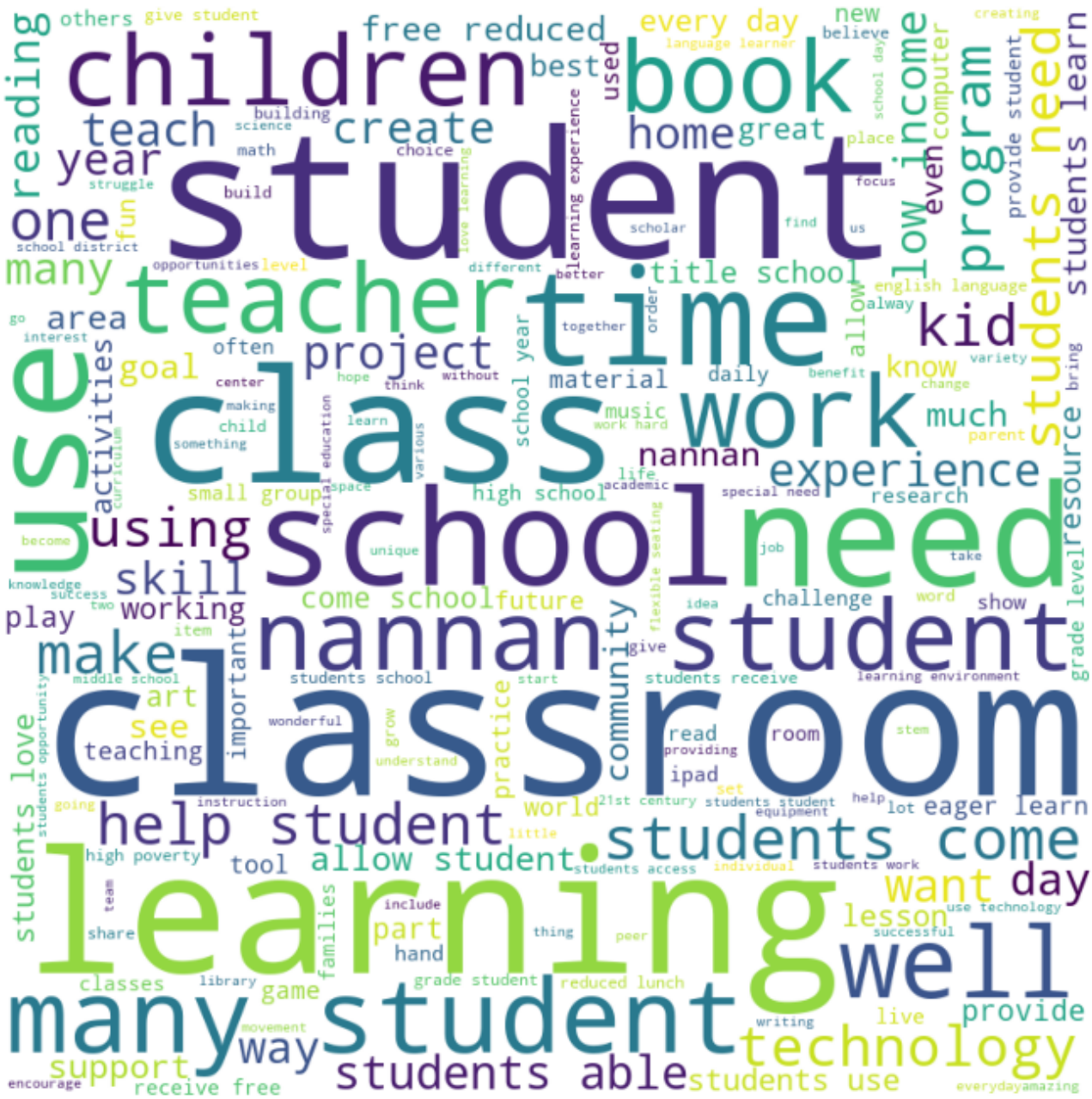
```
the maximum value of tpr*(1-fpr) 0.3637596624846207 for threshold 0.491
Train confusion matrix
[[ 6947  4136]
 [26067 36046]]
Test confusion matrix
[[ 3174  2285]
 [13028 17565]]
```

Generate Word Cloud:

```
def get_false_index(y_pred,y):
  l=[]
  for i in range(len(y)):
    if(y[i]==0 and y_pred[i]==1):
      l.append(i)
  return l


y_pred=predict_with_best_t(y_train_pred,best_t)


false_index=get_false_index(y_pred,y_train)


X=X_train.iloc[false_index,6]


from wordcloud import WordCloud, STOPWORDS
comment_words = ' '
stopwords = set(STOPWORDS)

for val in tqdm(X):
  val = str(val)
  tokens = val.split()

  for i in range(len(tokens)):
    tokens[i] = tokens[i].lower()

  for words in tokens:
    if((words.isdigit())==False):
      comment_words = comment_words + words + ' '


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

plt.figure(figsize = (9, 9), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
```

```
plt.show()
```

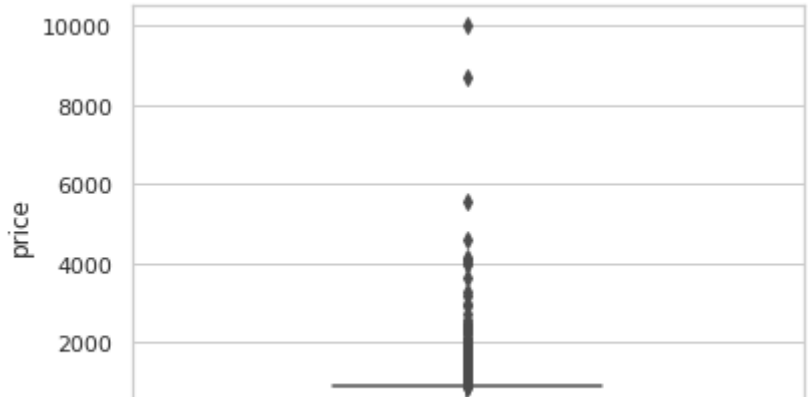
```
100%|██████████| 4398/4398 [05:11<00:00, 14.11it/s]
```



Plot the box plot with the 'price' of 'false positive datapoints'

```
X_price_fp=X_train.iloc[false_index,7]
```

```
import seaborn as sns
sns.set(style="whitegrid")
ax=sns.boxplot(y=X_price_fp)
```



Pdf of teacher_number_of_previously_posted_projects for false positive points

```
X_teachers_fp=X_train.iloc[false_index,3]
counts, bin_edges = np.histogram(X_teachers_fp, bins=10, density = True)
pdf = counts/(sum(counts))

print(pdf);
print(bin_edges);

cdf = np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf,label="pdf");
plt.plot(bin_edges[1:], cdf,label="cdf")
plt.title('CDF & PDF for teacher_number_of_previously_posted_projects for false positive p
plt.legend()
```

```
[9.27239654e-01 3.91086858e-02 1.31878126e-02 7.27603456e-03
 6.59390632e-03 2.27376080e-03 1.36425648e-03 2.27376080e-03
 2.27376080e-04 4.54752160e-04]
[  0.   24.8  49.6  74.4  99.2 124.  148.8 173.6 198.4 223.2 248. ]
<matplotlib.legend.Legend at 0x7fe16e8ed350>
```


```
#Let's Build PrettyTable:
from prettytable import PrettyTable

x=PrettyTable()
```

```
x.field_names=["Vectorizer","Model","max_depth","min_samples_spli1t","AUC"]
x.add_row(["Tfidf","Decision Tree Classifier","10","500","0.6288"])
x.add_row(["Tfidf_w2v","Decision Tree Classifier","5","50","0.6254"])
print(x)
```

```
+------------+---------------------------+-----------+--------------------+--------+
| Vectorizer |          Model            | max_depth | min_samples_spli1t |  AUC   |
+------------+---------------------------+-----------+--------------------+--------+
|   Tfidf    | Decision Tree Classifier |     10    |        500         | 0.6288 |
| Tfidf_w2v  | Decision Tree Classifier |     5     |         50         | 0.6254 |
+------------+---------------------------+-----------+--------------------+--------+
```

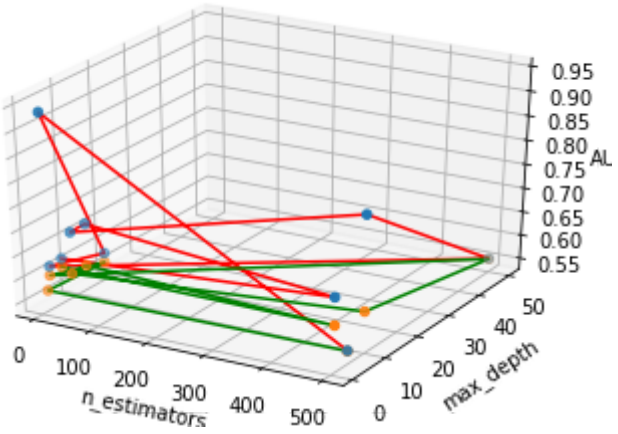
Task-2: Implementing Decision Tree for TFidf(essay) on Non-negative Features

```
from tqdm import tqdm

Model2= DecisionTreeClassifier(min_samples_split=500,class_weight='balanced')
Model2.fit(X_train1,y_train)
feature_importance=Model2.feature_importances_
print(np.count_nonzero(feature_importance))

index=[]
for i in tqdm(range(len(feature_importance))):
  if(feature_importance[i]!=0):
    index.append(i)
print(len(index))
```

```
1145
100%|██████████| 259314/259314 [00:00<00:00, 1319036.33it/s]1145
```

```
X_tr_fp = X_train1[:,index]
X_te_fp = X_test1[:,index]


clf.fit(X_tr_fp, y_train)

results2= pd.DataFrame.from_dict(clf.cv_results_)
train_auc2= results2['mean_train_score']
train_auc_std2= results2['std_train_score']
cv_auc2 = results2['mean_test_score']
cv_auc_std2= results2['std_test_score']
max_depth2= results2['param_max_depth']
min_samples_split2=results2['param_min_samples_split']


fig = plt.figure()
ax = plt.axes(projection='3d')
```

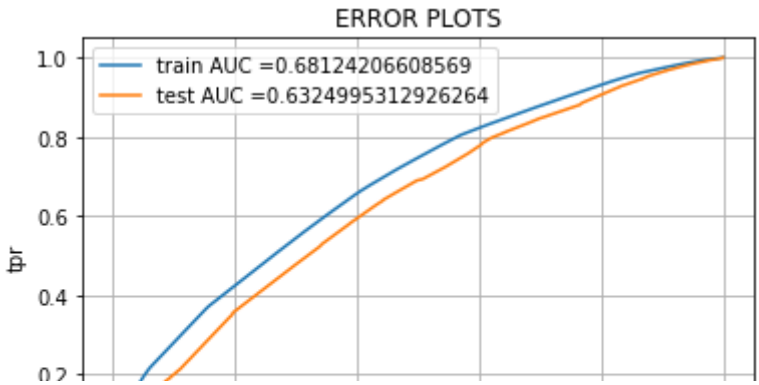
```
ax.scatter3D(min_samples_split, max_depth, train_auc2, cmap="Orange")
ax.plot3D(min_samples_split, max_depth, train_auc2, 'Red')
ax.set_xlabel('n_estimators')
ax.set_ylabel('max_depth')
ax.set_zlabel('AUC');
ax.scatter3D(min_samples_split, max_depth, cv_auc2, cmap="Blue")
ax.plot3D(min_samples_split, max_depth, cv_auc2, 'green')
plt.show()
```




```
clf.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')
```

```
Model2= DecisionTreeClassifier(max_depth=10,min_samples_split=500,class_weight='balanced')
Model2.fit(X_tr_fp, y_train)

y_train_pred_fp= batch_predict(Model2, X_tr_fp)
y_test_pred_fp= batch_predict(Model2, X_te_fp)

train_fpr3, train_tpr3, tr_thresholds3 = roc_curve(y_train, y_train_pred_fp)
test_fpr3, test_tpr3, te_thresholds3 = roc_curve(y_test, y_test_pred_fp)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr3, train_tpr3)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr3, test_tpr3)))
plt.legend()
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```




```
print("-"*50)
best_t3= find_best_threshold(tr_thresholds3, train_fpr3, train_tpr3)
print("-"*50)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred_fp, best_t3)))
print("-"*50)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred_fp, best_t3)))
```

```
--------------------------------------------------
the maximum value of tpr*(1-fpr) 0.39434212554994996 for threshold 0.525
--------------------------------------------------
Train confusion matrix
[[ 6685  4398]
 [21505 40608]]
--------------------------------------------------
Test confusion matrix
[[ 3104  2355]
 [11372 19221]]
```