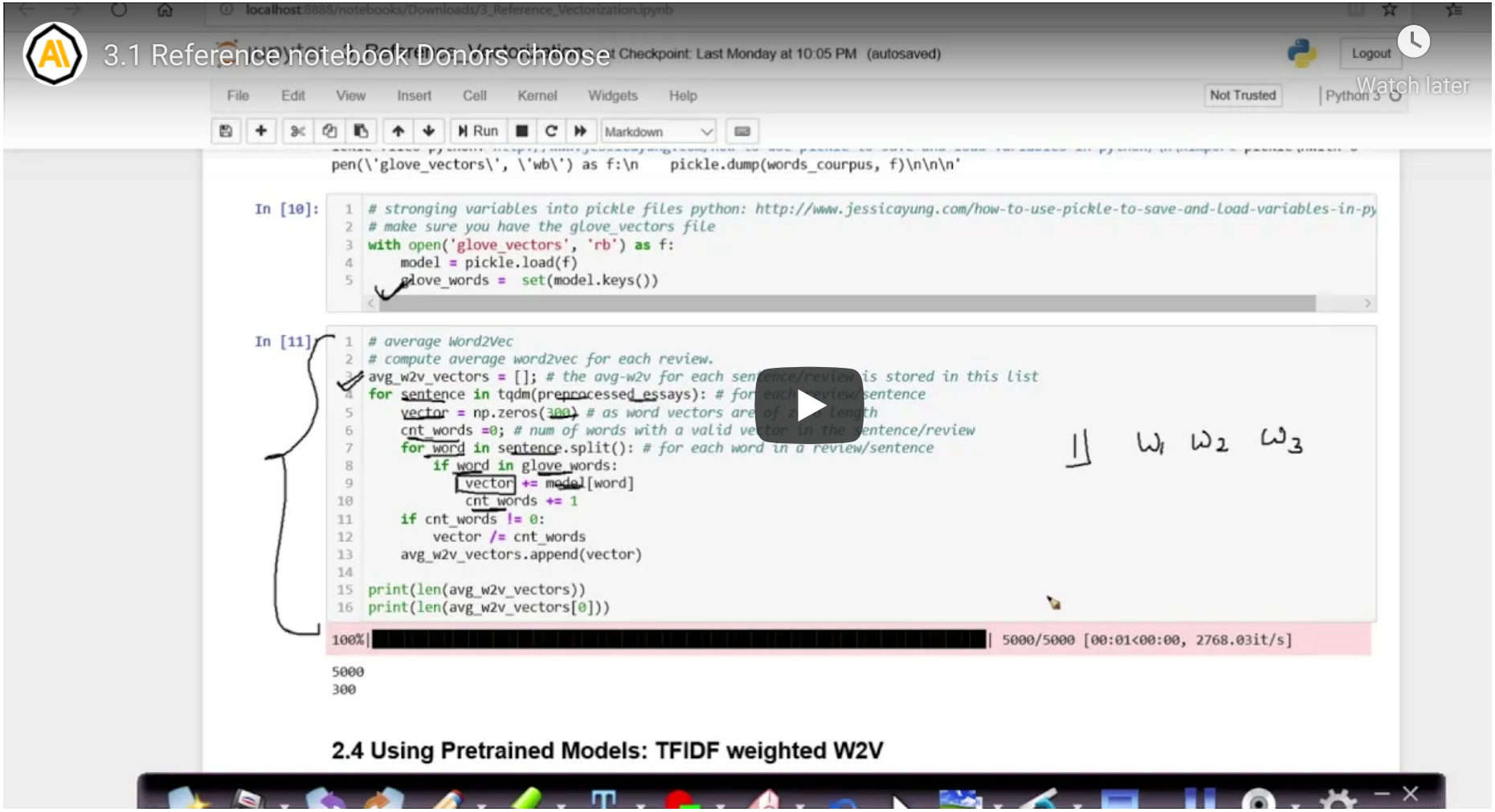
Assignment : DT

Please check below video before attempting this assignment

In [1]:
```
from IPython.display import YouTubeVideo
YouTubeVideo('ZhLXULFjIjQ', width="1000",height="500")
```

Out[1]:



TF-IDFW2V

Tfidf w2v (w1,w2..) = (tfidf(w1) * w2v(w1) + tfidf(w2) * w2v(w2) + …) / (tfidf(w1) + tfidf(w2) + …)

(Optional) Please check course video on AVgw2V and TF-IDFW2V for more details.

Glove vectors

In this assignment you will be working with glove vectors , please check [this](https://en.wikipedia.org/wiki/GloVe_(machine_learning)) and [this](https://en.wikipedia.org/wiki/GloVe_(machine_learning)) for more details.

Download glove vectors from this link

In [0]:
```
#please use below code to load glove vectors
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

or else , you can use below code

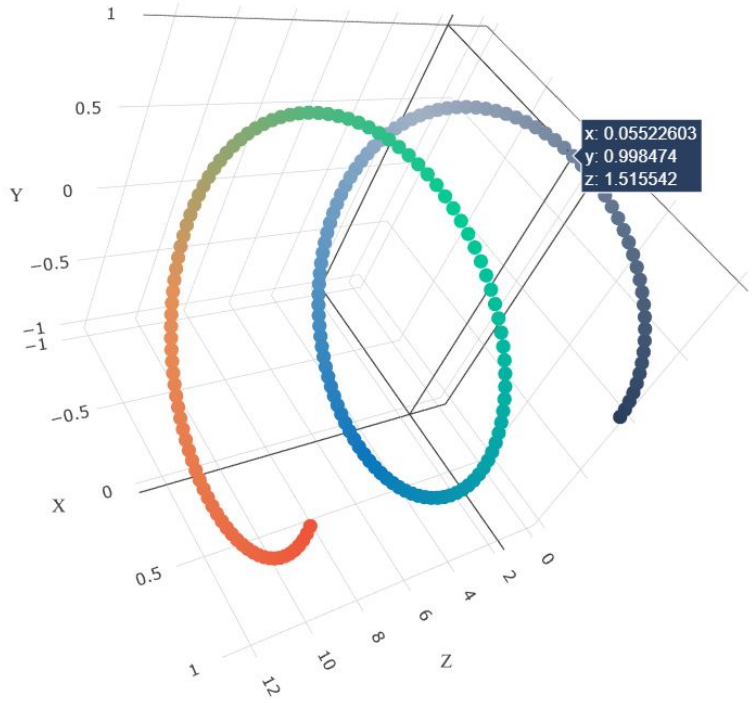
In [0]:
```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]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

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

Loading Glove Model
1917495it [06:32, 4879.69it/s]
```

```
Done. 1917495  words loaded!

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

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

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

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

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


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

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


'''
```

Task - 1

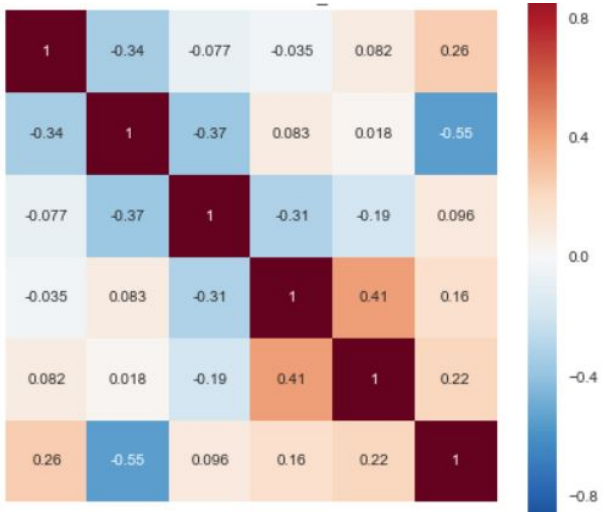
1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - **Set 1: categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)**

- **Set 2: categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)**
  ** **
- **The hyper paramter tuning (best $depth$ in range [1, 5, 10, 50], and the best $\min_s amp \leq s_split$ in range [5, 10, 100, 500])**
  - **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 randomsearch cv)/simple cross validation data(you can write your own for loops refer sample solution)**
    ** **
  - **Representation of results**
    - **You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like**



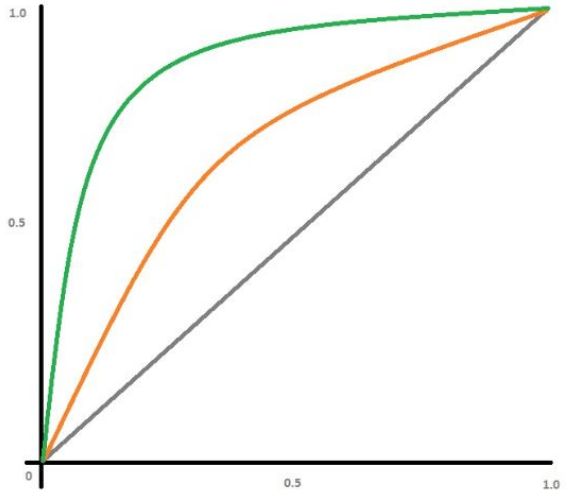
shown in the figure with X-axis as min_sample_split, Y-axis as max_depth, and Z-axis as AUC Score , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

or

- **You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure** 

seaborn heat maps with rows as n_estimators, columns as 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, and find the AUC on test data and plot the ROC curve on both train and test.**



- **Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points**

	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** $falsepositivedatapoints$
  - **Plot the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) with the words of essay text of these** $falsepositivedatapoints$
  - **Plot the box plot with the** $price$ **of these** $falsepositivedatapoints$
  - **Plot the pdf with the** $teacher_number_of_previously_posted_projects$ **of these** $falsepositivedatapoints$

Task - 2

For this task consider set-1 features.

- **Select all the features which are having non-zero feature importance.You can get the feature importance using 'feature*importances*` (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html), discard the all other remaining features and then apply any of the model of you choice i.e. (Dession tree, Logistic Regression, Linear SVM).**
- **You need to do hyperparameter tuning 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 parameter keep it None. **
  You need to summarize the results at the end of the notebook, summarize it in the table format
  `<img src='http://i.imgur.com/YVpIGGE.jpg' width=400px>`
  ** **

Hint for calculating Sentiment scores

In [1]:
```
import nltk
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\sai\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[1]:
```
True
```

In [2]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students with the biggest ent
for learning my students learn in many different ways using all of our senses and multiple intelligences i use a wide
of techniques to help all my students succeed students in my class come from a variety of different backgrounds whicl
for wonderful sharing of experiences and cultures including native americans our school is a caring community of succ
```

```
learners which can be seen through collaborative student project based learning in and out of the classroom kindergar
in my class love to work with hands on materials and have many different opportunities to practice a skill before it
mastered having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curricul
montana is the perfect place to learn about agriculture and nutrition my students love to role play in our pretend ki
in the early childhood classroom i have had several kids ask me can we try cooking with real food i will take their i
and create common core cooking lessons where we learn important math and writing concepts while cooking delicious hea
food for snack time my students will have a grounded appreciation for the work that went into making the food and kno
of where the ingredients came from as well as how it is healthy for their bodies this project would expand our learni
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce make our own
and mix up healthy plants from our classroom garden in the spring we will also create our own cookbooks to be printed
shared with families students will gain math and literature skills as well as a life long enjoyment for healthy cooki
nannan'
ss = sid.polarity_scores(for_sentiment)
print(ss)
for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
{'neg': 0.01, 'neu': 0.745, 'pos': 0.245, 'compound': 0.9975}
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

1. Decision Tree

1.1 Loading Data

In [3]:

```
# NECESSARY LIBRARIES:
%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
```

In [4]:

```
# read the dataset and fetch 50k datapoints
data = pd.read_csv('preprocessed_data.csv')

# we use only 50k datapoints
mydata = data.iloc[0:50000,:]
print(mydata.shape)

y = mydata["project_is_approved"].values # returns a numpy nd array--> Target variables
X = mydata.drop("project_is_approved",axis = 1) # Creates a dataframe X-->Input variables
mydata.head(3)
```

```
(50000, 9)
```

Out[4]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_
0	ca	mrs	grades_prek_2	53	1	math_science	a hea
1	ut	ms	grades_3_5	4	1	specialneeds	
2	ca	mrs	grades_prek_2	10	1	literacy_language	

In [5]:
```
# We have 50k rows and 9 features
mydata.shape
```

Out[5]:
```
(50000, 9)
```

TASK (1)

In [6]:
```
# Split data into train and test set:(stratified sampling)

from sklearn.model_selection import train_test_split # necessary library

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,stratify = y,random_state=6)
```

SET (1):

(A) DATA PREPROCESSING FOR CATEGORICAL FEATURES:

In [7]:
```
# CATEGORICAL FEATURE--> SCHOOL STATE:
vectorizer1 = CountVectorizer(binary = True)
vectorizer1.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_school = vectorizer1.transform(X_train['school_state'].values)
X_test_state_school = vectorizer1.transform(X_test['school_state'].values)

print("(A) Shape After vectorization of school state :")
print(X_train_state_school.shape, y_train.shape)
print(X_test_state_school.shape, y_test.shape)
print("="*100)



# CATEGORICAL FEATURES TEACHER_PREFIX:
vectorizer2 = CountVectorizer(binary = True)
vectorizer2.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data
```

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_prefix_teacher = vectorizer2.transform(X_train['teacher_prefix'].values)
X_test_prefix_teacher = vectorizer2.transform(X_test['teacher_prefix'].values)

print("(B) Shape After vectorization of teacher_prefix :")
print(X_train_prefix_teacher.shape, y_train.shape)
print(X_test_prefix_teacher.shape, y_test.shape)
print("="*100)



# CATEGORICAL FEATURES project_grade_category:
vectorizer3 = CountVectorizer(binary = True)
vectorizer3.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_cat = vectorizer3.transform(X_train['project_grade_category'].values)
X_test_grade_cat = vectorizer3.transform(X_test['project_grade_category'].values)

print("(C) Shape After vectorization of project_grade_Category :")
print(X_train_grade_cat.shape, y_train.shape)
print(X_test_grade_cat.shape, y_test.shape)
print("="*100)



# CATEGORICAL FEATURES CLEAN_CATEGORIES:

vectorizer4 = CountVectorizer(ngram_range = (1,3),binary = True)
vectorizer4.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_categories = vectorizer4.transform(X_train['clean_categories'].values)
X_test_categories = vectorizer4.transform(X_test['clean_categories'].values)

print("(D) Shape After vectorization of project Categories :")
print(X_train_categories.shape, y_train.shape)
print(X_test_categories.shape, y_test.shape)
print("="*100)
```

```
# CATEGORICAL FEATURES CLEAN_SUBCATEGORIES:

vectorizer5 = CountVectorizer(ngram_range=(1,3),binary = True)
vectorizer5.fit(X_train['clean_subcategories'].values)# fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategories = vectorizer5.transform(X_train['clean_subcategories'].values)
X_test_subcategories = vectorizer5.transform(X_test['clean_subcategories'].values)

print("(E) Shape After vectorization of project Subcategories :")
print(X_train_subcategories.shape, y_train.shape)
print(X_test_subcategories.shape, y_test.shape)
```

```
(A) Shape After vectorization of school state :
(35000, 51) (35000,)
(15000, 51) (15000,)
==========================================================================================
(B) Shape After vectorization of teacher_prefix :
(35000, 5) (35000,)
(15000, 5) (15000,)
==========================================================================================
(C) Shape After vectorization of project_grade_Category :
(35000, 4) (35000,)
(15000, 4) (15000,)
==========================================================================================
(D) Shape After vectorization of project Categories :
(35000, 50) (35000,)
(15000, 50) (15000,)
==========================================================================================
(E) Shape After vectorization of project Subcategories :
(35000, 350) (35000,)
(15000, 350) (15000,)
```

(B) DATAPREPROCESSING FOR NUMERICAL FEATURES:

ENCODING PRICE :

In [8]:
```
# Normalizing: map all the values to range of (0,1)

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

```
# fit and transform the train and test data:
# reshape of the data to single row allows the normalizer() to fit & transform
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("(F) Shape After Nomalization of feature Price :")
print(X_train_price_norm.transpose().shape, y_train.shape)
print(X_test_price_norm.transpose().shape, y_test.shape)

# assign to another variable for readability purpose :
X_train_price_norm = X_train_price_norm.transpose()
X_test_price_norm  = X_test_price_norm.transpose()
```

```
(F) Shape After Nomalization of feature Price :
(35000, 1) (35000,)
(15000, 1) (15000,)
```

ENCODING teacher_number_of_previously_posted_projects :

In [9]:
```
# Normalizing:map all the values to range of (0,1)

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

# fit and transform the train and test data:
# reshape of the data to single row allows the normalizer() to fit & transform
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_train_previous_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1
X_test_previous_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-

print("(G) Shape After Normalization of Number of previously posted projects :")
print(X_train_previous_norm.transpose().shape, y_train.shape)
print(X_test_previous_norm.transpose().shape, y_test.shape)

# assign to another variable for readability purpose :
X_train_previous_norm = X_train_previous_norm.transpose()
X_test_previous_norm = X_test_previous_norm.transpose()
```

```
(G) Shape After Normalization of Number of previously posted projects :
(35000, 1) (35000,)
(15000, 1) (15000,)
```

(C) DATAPREPROCESSING FOR TEXT FEATURE(ESSAY):

In [10]:
```
#TEXT FEATURE --> ESSAY ENCODING INTO NUMERIC VECTOR(tfidf):

from sklearn.feature_extraction.text import TfidfVectorizer

print("(i) Shape before vectorization of essay(feature) :")
print(X_train.shape,y_train.shape)
print(X_test.shape,y_test.shape)
print("\n")

# Use the tfidf vectorizer to encode the text data
vectorizer = TfidfVectorizer(min_df= 30,ngram_range=(1,2),max_features=20000)
vectorizer.fit(X_train['essay'].values) # fit on train data

# we use the fitted Vectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("(ii) Shape After vectorization of essay(feature) :")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
(i) Shape before vectorization of essay(feature) :
(35000, 8) (35000,)
(15000, 8) (15000,)


(ii) Shape After vectorization of essay(feature) :
(35000, 20000) (35000,)
(15000, 20000) (15000,)
```

(D) SENTIMENT SCORES(ESSAY FEATURE) :

In [11]:
```
# Define a function and call the function:
def sentiment_scores(lst):
    neg,neu,pos,compound = [],[],[],[]
    for sent in lst:
        sentiment_dict = sia.polarity_scores(sent)
        neg.append(sentiment_dict['neg'])
        neu.append(sentiment_dict['neu'])
        pos.append(sentiment_dict['pos'])
```

```
        compound.append(sentiment_dict['compound'])
    negative = pd.Series(neg)
    neutral = pd.Series(neu)
    positive = pd.Series(pos)
    compound = pd.Series(compound)
    features = {'Negative':negative,"Neutral":neutral,"Positive":positive,"Compound":compound}
    result = pd.DataFrame(features)
    return result

sia = SentimentIntensityAnalyzer()
lst_xtrain = X_train['essay'].values
lst_xtest = X_test['essay'].values

df_Xtrain = sentiment_scores(lst_xtrain)
df_Xtest = sentiment_scores(lst_xtest)
df_Xtrain.head()
```

Out[11]:

	Negative	Neutral	Positive	Compound
0	0.083	0.612	0.305	0.9906
1	0.046	0.854	0.099	0.8402
2	0.088	0.711	0.201	0.9747
3	0.086	0.694	0.220	0.9788
4	0.057	0.677	0.267	0.9770

In [12]:
```
# convert pandas dataframe to numpy arrays for fast computation:
sentimentscores_Xtrain = df_Xtrain.to_numpy()
sentimentscores_Xtest  = df_Xtest.to_numpy()

print(sentimentscores_Xtrain.shape)
print(sentimentscores_Xtest.shape)
```

```
(35000, 4)
(15000, 4)
```

(E) CONCATENATING FEATURES FOR SET 1 :

In [61]:
```
# concatenate all the features :
```

```
from scipy.sparse import hstack

# hstack() helps in concatenating "n" number of array like shapes into one dataframe.
# we store the concatenated outcome in a csr matrix format.

train_X = hstack((X_train_essay_tfidf,X_train_state_school,
                  X_train_prefix_teacher,X_train_grade_cat,
                 X_train_categories,X_train_subcategories,
                 X_train_price_norm,X_train_previous_norm,
                 sentimentscores_Xtrain)).tocsr()

test_X = hstack((X_test_essay_tfidf,X_test_state_school,
                  X_test_prefix_teacher,X_test_grade_cat,
                 X_test_categories,X_test_subcategories,
                 X_test_price_norm,X_test_previous_norm,
                sentimentscores_Xtest)).tocsr()

print("(H) Final Data matrix :")

print(train_X.shape, y_train.shape)#we totally have 35000 rows & 20466 columns in train data

print(test_X.shape, y_test.shape)#we totally have 15000 rows & 20466 columns in test data
```

```
(H) Final Data matrix :
(35000, 20466) (35000,)
(15000, 20466) (15000,)
```

(F) APPLY DECISION TREE WITH HYPERPARAMETER TUNING:

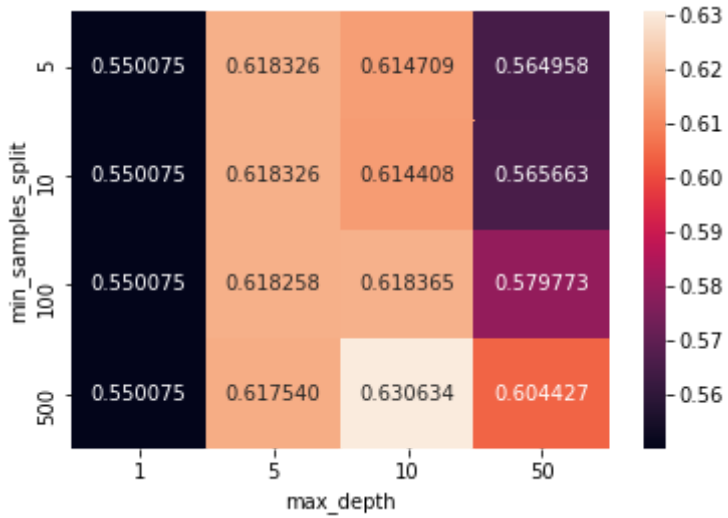
In [62]:

```
# build decision tree classifier

# import necessary libraries:
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from time import time

start = time()

# fit the model to the training data using randomsearchCV
model = DecisionTreeClassifier(random_state = 10,class_weight="balanced")
param = {'max_depth':[1,5,10,50],'min_samples_split' :[5,10,100,500]}
```

```
# use "ROC_AUC" as a scoring and CV = 10
clf = GridSearchCV(model,param,cv=4,scoring='roc_auc',
                           return_train_score = True)

clf.fit(train_X,y_train)

print("The best paramters from gridsearchCv :",clf.best_params_)
# make a dataframe out of cv_results:
cv_results = pd.DataFrame.from_dict(clf.cv_results_)


# obtain mean train,Cv scores and their corresponding hyperparameters:
train_auc = cv_results['mean_train_score']
cv_auc = cv_results['mean_test_score']
depth =  cv_results['param_max_depth']
min_sample_split = cv_results['param_min_samples_split']


# lets observe the point of minimal gap between CV and train curves:
difference = train_auc - cv_auc
print("(A) Difference between train_auc & cv_auc :\n",difference)
print("\n")

print("(B) The minimal difference is :",min(difference))
print("\n")

print("(C) The minimal difference is observed at the depth :",depth[difference.idxmin()])
print("\n")

print("(D) The minimal difference is observed at the sample_split of :"
      ,min_sample_split[difference.idxmin()])
print("\n")


# plot the Hyperparameters vs AUC plot (heatmap) for train data:

df = pd.DataFrame({'AUC_score':list(cv_auc),
                   'max_depth':list(depth),
                   'min_samples_split':list(min_sample_split)})

result = df.pivot("min_samples_split","max_depth","AUC_score")
sns.heatmap(result,annot=True,fmt="f")
```

```
plt.show()


end = time()
training_time = end - start
print("training & validation time: %0.2fs" % training_time)
```

```
The best paramters from gridsearchCv : {'max_depth': 10, 'min_samples_split': 500}
(A) Difference between train_auc & cv_auc :
 0     0.005993
1     0.005993
2     0.005993
3     0.005993
4     0.035860
5     0.035826
6     0.034264
7     0.032712
8     0.158902
9     0.156958
10    0.130574
11    0.090052
12    0.418306
13    0.414388
14    0.350547
15    0.238001
dtype: float64


(B) The minimal difference is : 0.00599325577878107


(C) The minimal difference is observed at the depth : 1


(D) The minimal difference is observed at the sample_split of : 5
```



training & validation time: 1536.24s

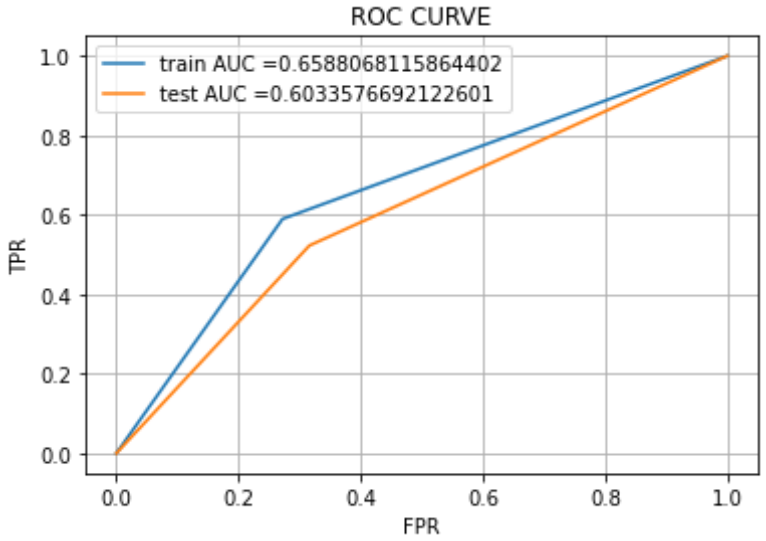
OBSERVATION:

From the above results, we can see that minimal gap between cv_auc and train_auc is observed at max_depth = 1 and min_samples_split = 5, but as the auc value is very low that is just over 50%, we can choose for max_depth = 10 and min_samples_split = 500 which provides us some reasonable high value of auc for both cv and train data with minimal gap of 0.090, hence,that is why the gridsearch implementation has chosen max_depth = 10 & min_samples_split = 500 as the best_params and now we go ahead for testing phase then interpret the results for test data.

(G) TEST THE PERFORMANCE OF MODEL ON TEST DATA:

In [63]:
```
from sklearn.metrics import roc_curve, auc # -->necessary libraries

# Fit the classifier with the optimal alpha:
clf = DecisionTreeClassifier(max_depth = 10,min_samples_split = 500,
                             class_weight="balanced",random_state = 10)
clf.fit(train_X,y_train)

# predict for train and test data :
y_pred_train = clf.predict(train_X)
y_pred_test = clf.predict(test_X)

# compute TPR,FPR values to construt ROC curve:
train_fpr,train_tpr,tr_thresholds = roc_curve(y_train,y_pred_train)
test_fpr,test_tpr,te_thresholds = roc_curve(y_test,y_pred_test)
```

```
#plot the ROC with Train AUC and Test AUC:
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("ROC CURVE")
plt.grid()
plt.show()
```

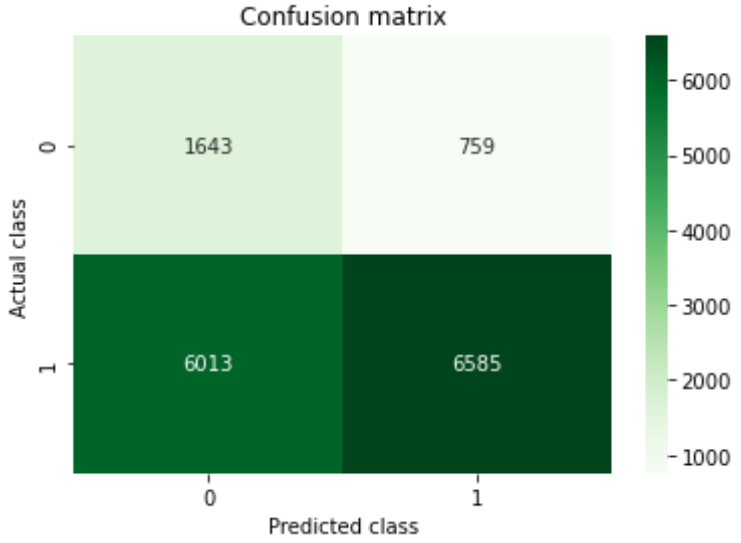


(H) PLOT CONFUSION MATRIX:

In [64]:
```
# CONFUSION MATRIX :
from sklearn.metrics import confusion_matrix
confusion_mat = confusion_matrix(y_test,y_pred_test)
print(confusion_mat)

# Represent confusion matrix as a heatmap:
cm = np.array([[1643,759],[6013,6585]])
sns.heatmap(cm,annot=True,fmt="d",cmap='Greens')
plt.xlabel("Predicted class")
plt.ylabel("Actual class")
```

```
plt.title("Confusion matrix")
plt.show()
```

```
[[1643  759]
 [6013 6585]]
```



(H1) FETCH FALSE POSITIVE DATA POINTS:

In [65]:
```
# first we initialize a list to store FP points.
false_positive_pts = []
for idx,j in enumerate(y_test):
    if ((j == 0) and (y_pred_test[idx] == 1)):
        false_positive_pts.append(idx)
    else:
        continue

print(len(false_positive_pts))

# extract "essay" values of false positive points and ensure its length is 477.
fp_essay = []
for index in false_positive_pts :
    fp_essay.append(X_test.iloc[index,6])

#print(len(fp_essay))
```

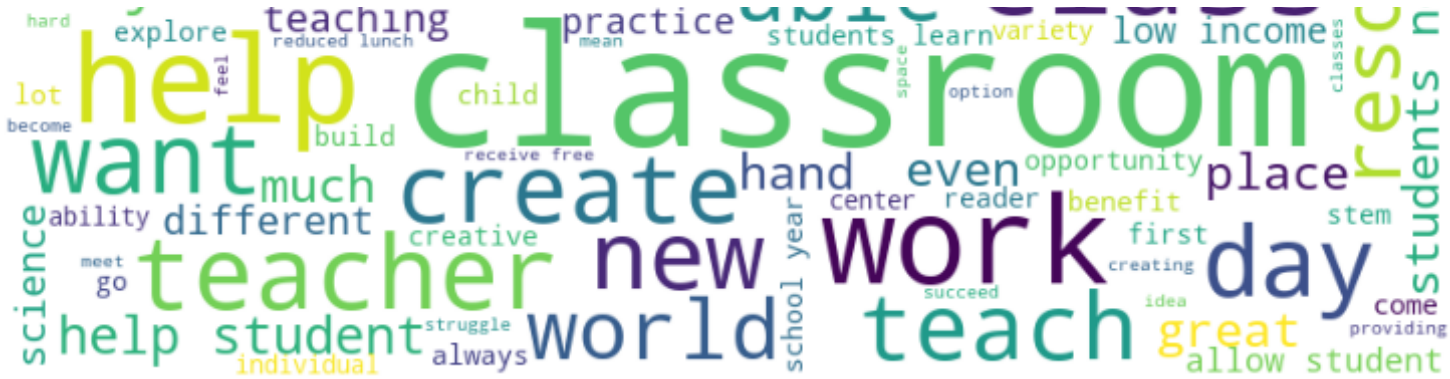
```
fp_price = []
for index in false_positive_pts :
    fp_price.append(X_test.iloc[index,7])

#print(len(fp_price))


fp_teacher_no_of_previously_posted_project = []
for index in false_positive_pts :
    fp_teacher_no_of_previously_posted_project.append(X_test.iloc[index,3])

#print(len(fp_teacher_no_of_previously_posted_project))
```

```
759
```

(H2) GENERATE WORDCLOUD FOR FEATURE ESSAY(FALSE POSITIVE PTS):

In [66]:
```
#REFERENCE :https://www.geeksforgeeks.org/generating-word-cloud-python/
# GENERATE WORDCLOUD:
from wordcloud import WordCloud, STOPWORDS

comment_words = ''
stopwords = set(STOPWORDS)

# iterate through the essay
for sent in fp_essay:

    # split the value
    tokens = sent.split()

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

    comment_words += " ".join(tokens)+" "

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

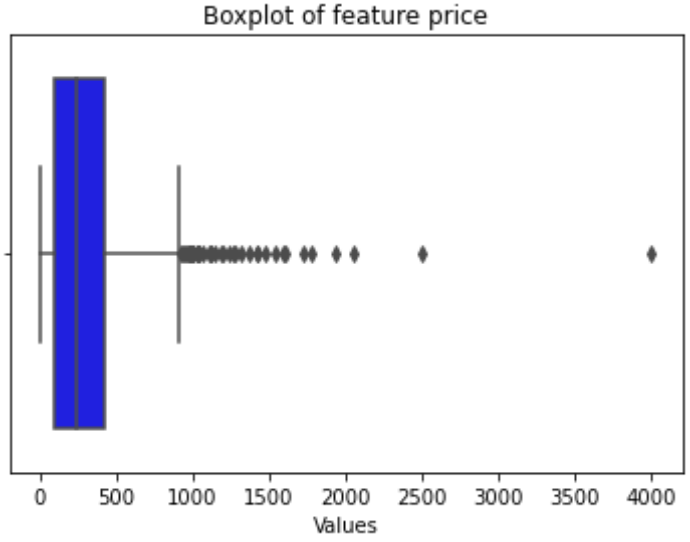
```
# plot the WordCloud image
plt.figure(figsize = (10,10),facecolor = None)
plt.imshow(word_cloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



(H3) BOXPLOT OF FEATURE "PRICE"(false positive pts):

In [67]:
```
# plot using seaborn.
sns.boxplot(x = fp_price,color = 'b')
plt.title("Boxplot of feature price")
plt.xlabel("Values")
```

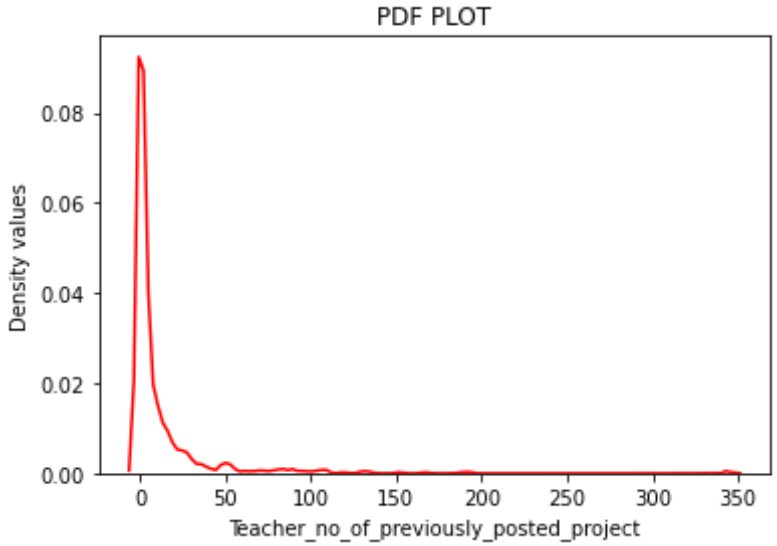
Out[67]: `Text(0.5, 0, 'Values')`



(H4) PDF OF FEATURE 'Teacher_no_of_previously_posted_projects' (False positive pts) :

In [68]:
```
# plot using seaborn
sns.distplot(fp_teacher_no_of_previously_posted_project,hist=False,kde = True,color ='r')
plt.title("PDF PLOT")
plt.ylabel("Density values")
plt.xlabel("Teacher_no_of_previously_posted_project")
```

Out[68]:
```
Text(0.5, 0, 'Teacher_no_of_previously_posted_project')
```



TASK (2)

(A) FETCH THE IMPORTANT FEATURES THAT HELPS IN DETERMINING TARGET:

In [69]:
```
# build decision tree classifier

# import necessary libraries:
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV
from time import time

start = time()

# Fit the classifier with the optimal alpha:
```

```
clf = DecisionTreeClassifier(max_depth = None,min_samples_split = 500,
                             class_weight="balanced",random_state = 10)
clf.fit(train_X,y_train)

feature_impt = clf.feature_importances_

end = time()
t = end - start
print(t)
```

```
97.59285259246826
```

In [70]:
```
# fetch the feature having non_zero feature importance:

nonzero_indices = np.argwhere(feature_impt)
print("The count of important features :",nonzero_indices.shape)
print(type(nonzero_indices))

imp_features = list(nonzero_indices.reshape(1,-1)[0]) # make a list of features.

train_X_with_impt_features = train_X[:,imp_features].toarray()

test_X_with_impt_features = test_X[:,imp_features].toarray()

print("The shape of train data :",train_X_with_impt_features.shape)
print("The shape of test data :",test_X_with_impt_features.shape)
```

```
The count of important features : (768, 1)
<class 'numpy.ndarray'>
The shape of train data : (35000, 768)
The shape of test data : (15000, 768)
```

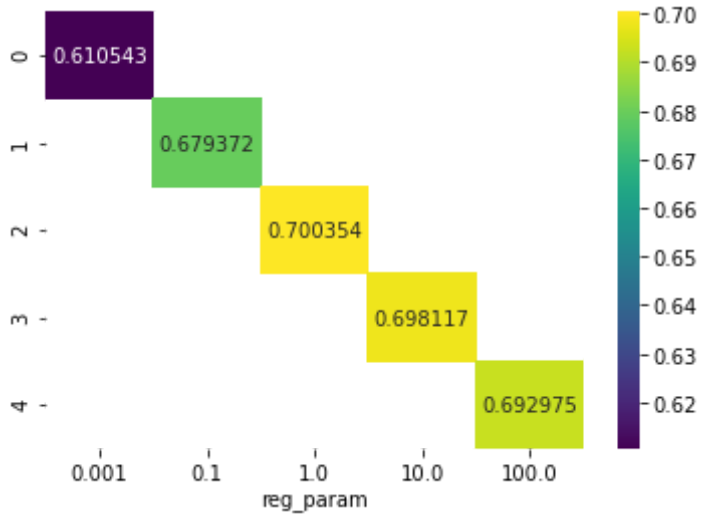
(B) APPLY LOGISTIC REGRESSION WITH HYPERPARAMETER TUNING:

In [88]:
```
# NOW LETS APPLY LOGISTIC REGRESSION ON THIS NEW TRAIN_X AND EVALUATE USING TEST_X.

# import necessary libraries:
#from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from time import time
```

```
start = time()

# fit the model to the training data using gridsearchCV
model = LogisticRegression(random_state = 10,class_weight="balanced")

param = {'C': [0.001,0.1,1,10,100]}

# use "ROC_AUC" as a scoring and CV = 5
clf = GridSearchCV(model,param,cv=5,scoring='roc_auc',
                           return_train_score = True)

for k in tqdm(range(0,1)):
    clf.fit(train_X_with_impt_features,y_train)

print("The best paramters from gridsearchCv :",clf.best_params_)
print("\n")

# make a dataframe out of cv_results:
cv_results = pd.DataFrame.from_dict(clf.cv_results_)


# obtain mean train,Cv scores and their corresponding hyperparameters:
train_auc = cv_results['mean_train_score']
cv_auc = cv_results['mean_test_score']
reg_param =  cv_results['param_C']


# lets observe the point of minimal gap between CV and train curves:
difference = train_auc - cv_auc
print("(A) Difference between train_auc & cv_auc :\n",difference)
print("\n")

print("(B) The minimal difference is :",min(difference))
print("\n")

print("(C) The minimal difference is observed at the parameter :",reg_param[difference.idxmin()])
print("\n")

# plot the Hyperparameters vs AUC plot (heatmap) for train data:

df = pd.DataFrame({'AUC_score':list(cv_auc),
                   'reg_param':list(reg_param)})
```

```
result = df.pivot(columns = "reg_param",values = "AUC_score")
sns.heatmap(result,annot=True,fmt="f",cmap='viridis')
plt.show()


end = time()
training_time = end - start
print("training & validation time: %0.2fs" % training_time)
```

```
100%|██████████| 1/1 [01:19<00:00, 79.57s/it]
The best paramters from gridsearchCv : {'C': 1}


(A) Difference between train_auc & cv_auc :
 0    0.006273
1    0.014817
2    0.033161
3    0.053872
4    0.059931
dtype: float64


(B) The minimal difference is : 0.006273184435020895


(C) The minimal difference is observed at the parameter : 0.001
```



training & validation time: 80.07s

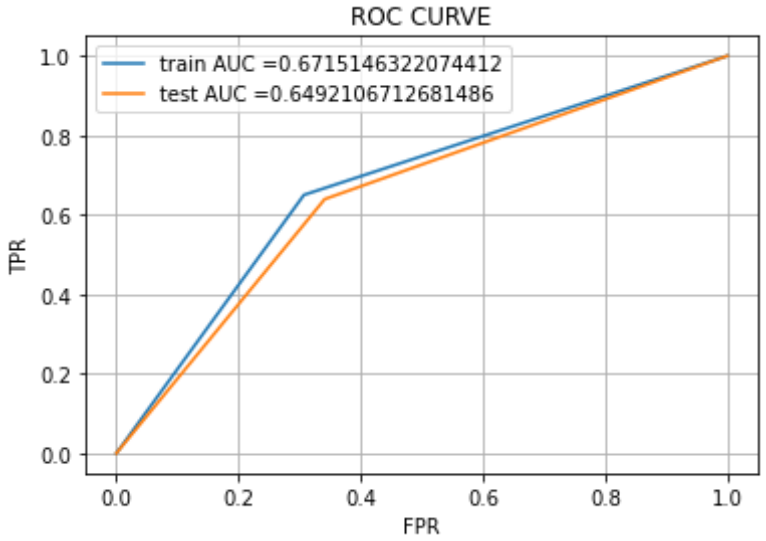
OBSERVATION :

From the above results, we can see that minimal gap between cv_auc and train_auc is observed at Reg_param = 0.001, but as the auc value is low that is just 0.61 , we can choose for reg_param = "1" which provides us some reasonable high value of auc for both cv and train data with minimal gap of 0.03, hence,that is why the gridsearch implementation has chosen reg_param = 1 as the best_params and now we go ahead for testing phase then interpret the results for test data.

(C) TEST THE PERFORMANCE OF MODEL:

In [90]:
```
from sklearn.metrics import roc_curve, auc # -->necessary libraries

# Fit the classifier with the optimal alpha:
clf = LogisticRegression(C = 1,class_weight="balanced",random_state = 10)

clf.fit(train_X_with_impt_features,y_train)

# predict for train and test data :
y_pred_train = clf.predict(train_X_with_impt_features)
y_pred_test = clf.predict(test_X_with_impt_features)

# compute TPR,FPR values to construt ROC curve:
train_fpr,train_tpr,tr_thresholds = roc_curve(y_train,y_pred_train)
test_fpr,test_tpr,te_thresholds = roc_curve(y_test,y_pred_test)
```

```
#plot the ROC with Train AUC and Test AUC:
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("ROC CURVE")
plt.grid()
plt.show()
```

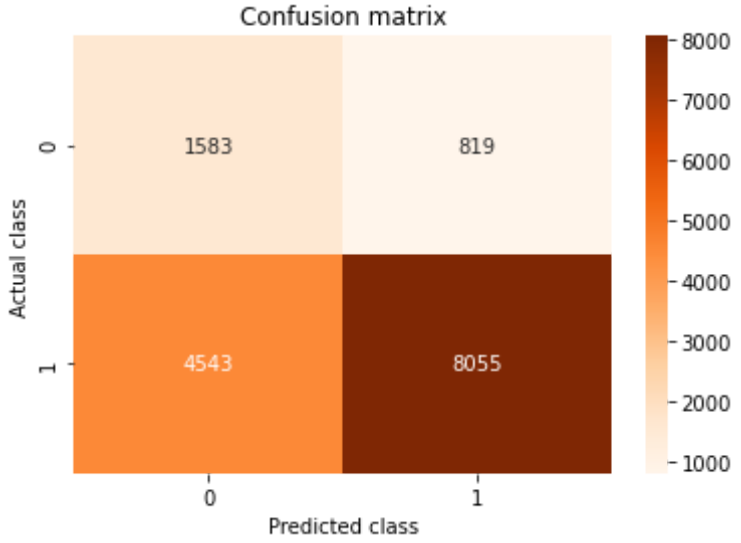


(D) PLOT CONFUSION MATRIX:

In [96]:
```
# CONFUSION MATRIX :
from sklearn.metrics import confusion_matrix
confusion_mat = confusion_matrix(y_test,y_pred_test)
print(confusion_mat)

# Represent confusion matrix as a heatmap:
cm = np.array([[1583,819],[4543,8055]])
sns.heatmap(cm,annot=True,fmt="d",cmap='Oranges')
plt.xlabel("Predicted class")
plt.ylabel("Actual class")
```

```
plt.title("Confusion matrix")
plt.show()
```

```
[[1583  819]
 [4543 8055]]
```



(D1) FETCH FALSE POSITIVE DATAPOINTS:

In [97]:
```
# first we initialize a list to store FP points.
false_positive_pts = []
for idx,j in enumerate(y_test):
    if ((j == 0) and (y_pred_test[idx] == 1)):
        false_positive_pts.append(idx)
    else:
        continue

print(len(false_positive_pts))

# extract "essay" values of false positive points and ensure its length is 819.
fp_essay = []
for index in false_positive_pts :
    fp_essay.append(X_test.iloc[index,6])

#print(len(fp_essay))
```

```
fp_price = []
for index in false_positive_pts :
    fp_price.append(X_test.iloc[index,7])

#print(len(fp_price))


fp_teacher_no_of_previously_posted_project = []
for index in false_positive_pts :
    fp_teacher_no_of_previously_posted_project.append(X_test.iloc[index,3])

#print(len(fp_teacher_no_of_previously_posted_project))
```

```
819
```

(D3) GENERATE WORD CLOUD FOR "ESSAY" (False positive pts):

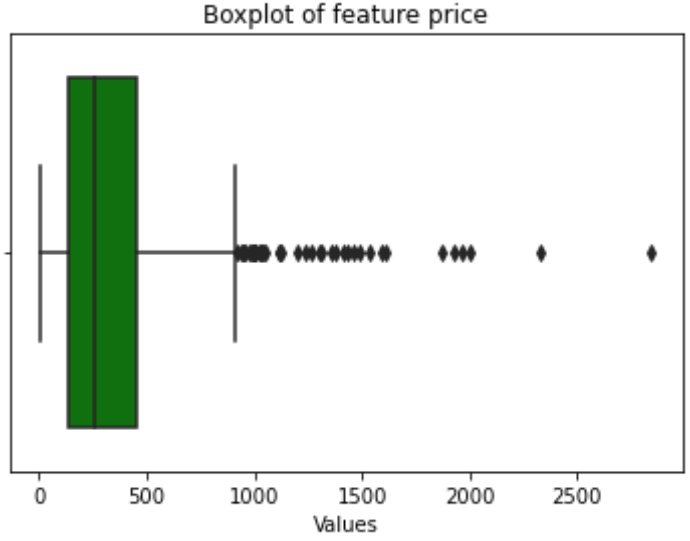
In [98]:
```
#REFERENCE :https://www.geeksforgeeks.org/generating-word-cloud-python/
# GENERATE WORDCLOUD:
from wordcloud import WordCloud, STOPWORDS

comment_words = ''
stopwords = set(STOPWORDS)

# iterate through the essay
for sent in fp_essay:

    # split the value
    tokens = sent.split()

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

    comment_words += " ".join(tokens)+" "

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

```
# plot the WordCloud image
plt.figure(figsize = (10,10),facecolor = None)
plt.imshow(word_cloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

(D4) BOXPLOT OF FEATURE "PRICE" (false positive pts):

In [99]:
```
# plot using seaborn.
sns.boxplot(x = fp_price,color = 'g')
plt.title("Boxplot of feature price")
plt.xlabel("Values")
```

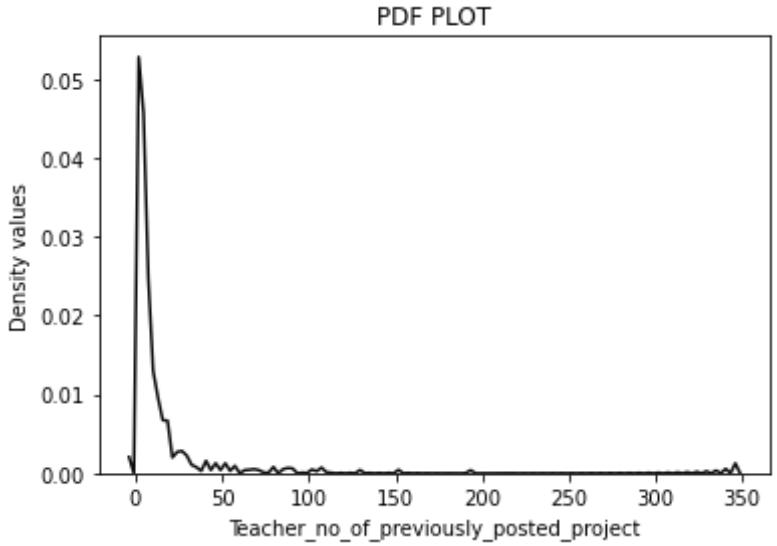
Out[99]: Text(0.5, 0, 'Values')



(D5) PDF OF FEATURE 'Teacher_no_of_previously_posted_projects' (False positive pts) :

In [105…
```
# plot using seaborn
sns.distplot(fp_teacher_no_of_previously_posted_project,hist=False,kde = True,color ='#0f0f0f')
plt.title("PDF PLOT")
plt.ylabel("Density values")
plt.xlabel("Teacher_no_of_previously_posted_project")
```

Out[105…
```
Text(0.5, 0, 'Teacher_no_of_previously_posted_project')
```



SET 2 :

(CONSIDERING ONLY 5K POINTS TO HANDLE TIME COMPLEXITY OF TFIDF-W2V PROCESS)

In [106…
```
# read the dataset and fetch 50k datapoints
data = pd.read_csv('preprocessed_data.csv')

# we use only 50k datapoints
mydata = data.iloc[5000:10000,:]
mydata.reset_index(inplace = True)
print(mydata.shape)

y = mydata["project_is_approved"].values # returns a numpy nd array--> Target variables
```

```
X = mydata.drop(["index","project_is_approved"],axis = 1) # Creates a dataframe X-->Input variables
X.head(3)
```

```
(5000, 10)
```

Out[106…

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	
0	co	mrs	grades_3_5	2	literacy_language	literacy	th g clas st
1	ca	ms	grades_prek_2	21	math_science	appliedsciences environmentalscience	gre stu cor lear
2	ca	mr	grades_6_8	10	math_science	appliedsciences mathematics	as m s stu alm

In [107…

```
# Split data into train and test set:(stratified sampling)

from sklearn.model_selection import train_test_split # necessary library

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,stratify = y,random_state=6)
```

(A) DATAPREPROCESSING FOR CATEGORICAL FEATURES:

In [108…

```
# CATEGORICAL FEATURE--> SCHOOL STATE:
vectorizer1 = CountVectorizer(binary = True)
vectorizer1.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_school = vectorizer1.transform(X_train['school_state'].values)
X_test_state_school = vectorizer1.transform(X_test['school_state'].values)
```

```
print("(A) Shape After vectorization of school state :")
print(X_train_state_school.shape, y_train.shape)
print(X_test_state_school.shape, y_test.shape)
print("="*100)



# CATEGORICAL FEATURES TEACHER_PREFIX:
vectorizer2 = CountVectorizer(binary = True)
vectorizer2.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_prefix_teacher = vectorizer2.transform(X_train['teacher_prefix'].values)
X_test_prefix_teacher = vectorizer2.transform(X_test['teacher_prefix'].values)

print("(B) Shape After vectorization of teacher_prefix :")
print(X_train_prefix_teacher.shape, y_train.shape)
print(X_test_prefix_teacher.shape, y_test.shape)
print("="*100)



# CATEGORICAL FEATURES project_grade_category:
vectorizer3 = CountVectorizer(binary = True)
vectorizer3.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_cat = vectorizer3.transform(X_train['project_grade_category'].values)
X_test_grade_cat = vectorizer3.transform(X_test['project_grade_category'].values)

print("(C) Shape After vectorization of project_grade_Category :")
print(X_train_grade_cat.shape, y_train.shape)
print(X_test_grade_cat.shape, y_test.shape)
print("="*100)



# CATEGORICAL FEATURES CLEAN_CATEGORIES:
```

```
vectorizer4 = CountVectorizer(ngram_range = (1,3),binary = True)
vectorizer4.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_categories = vectorizer4.transform(X_train['clean_categories'].values)
X_test_categories = vectorizer4.transform(X_test['clean_categories'].values)

print("(D) Shape After vectorization of project Categories :")
print(X_train_categories.shape, y_train.shape)
print(X_test_categories.shape, y_test.shape)
print("="*100)


# CATEGORICAL FEATURES CLEAN_SUBCATEGORIES:

vectorizer5 = CountVectorizer(ngram_range=(1,3),binary = True)
vectorizer5.fit(X_train['clean_subcategories'].values)# fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategories = vectorizer5.transform(X_train['clean_subcategories'].values)
X_test_subcategories = vectorizer5.transform(X_test['clean_subcategories'].values)

print("(E) Shape After vectorization of project Subcategories :")
print(X_train_subcategories.shape, y_train.shape)
print(X_test_subcategories.shape, y_test.shape)
```

```
(A) Shape After vectorization of school state :
(3500, 51) (3500,)
(1500, 51) (1500,)
====================================================================================================
(B) Shape After vectorization of teacher_prefix :
(3500, 5) (3500,)
(1500, 5) (1500,)
====================================================================================================
(C) Shape After vectorization of project_grade_Category :
(3500, 4) (3500,)
(1500, 4) (1500,)
====================================================================================================
(D) Shape After vectorization of project Categories :
(3500, 40) (3500,)
(1500, 40) (1500,)
====================================================================================================
```

```
(E) Shape After vectorization of project Subcategories :
(3500, 214) (3500,)
(1500, 214) (1500,)
```

(B) ENCODING NUMERICAL FEATURE "price":

In [109…

```
# Normalizing: map all the values to range of (0,1)

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

# fit and transform the train and test data:
# reshape of the data to single row allows the normalizer() to fit & transform
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("(F) Shape After Nomalization of feature Price :")
print(X_train_price_norm.transpose().shape, y_train.shape)
print(X_test_price_norm.transpose().shape, y_test.shape)

# assign to another variable for readability purpose :
X_train_price_norm = X_train_price_norm.transpose()
X_test_price_norm  = X_test_price_norm.transpose()
```

```
(F) Shape After Nomalization of feature Price :
(3500, 1) (3500,)
(1500, 1) (1500,)
```

(C) ENCODING NUMERICAL FEATURE (Teacher_number_of_previously_posted_projects):

In [110…

```
# Normalizing:map all the values to range of (0,1)

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

# fit and transform the train and test data:
# reshape of the data to single row allows the normalizer() to fit & transform
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_train_previous_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1
X_test_previous_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-
```

```
print("(G) Shape After Normalization of Number of previously posted projects :")
print(X_train_previous_norm.transpose().shape, y_train.shape)
print(X_test_previous_norm.transpose().shape, y_test.shape)

# assign to another variable for readability purpose :
X_train_previous_norm = X_train_previous_norm.transpose()
X_test_previous_norm = X_test_previous_norm.transpose()
```

```
(G) Shape After Normalization of Number of previously posted projects :
(3500, 1) (3500,)
(1500, 1) (1500,)
```

(D) VECTORIZATION OF TEXT FEATURE 'ESSAY' (TFIDF-W2V):

In [111…
```
#please use below code to load glove vectors
with open('glove_vectors', 'rb') as f:
    glovemodel = pickle.load(f)
    glove_words = set(glovemodel.keys())

len(glovemodel["enjoy"])
```

Out[111…
```
300
```

In [112…
```
# first lets create a tfidf model
model = TfidfVectorizer()
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(model.get_feature_names(),list(model.idf_)))
```

(FOR TRAIN DATA):

In [113…
```
# compute TFIDF word2vec for each essay in train data
import time
st = time.time()

tfidf_feat = model.get_feature_names() # tfidf words/col-names


tfidf_sent_vec_train = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0; #--> To store number of iterations
```

```
list_of_sent = X_train['essay'].values

for sent in tqdm(list_of_sent):
    sent_vec = np.zeros(50)
    weight_sum =0;                                  # To store sum of tfidf values.
    for word in sent:                               # for each word in a review/sentence
        if word in glove_words and word in tfidf_feat:
            vec = glovemodel[word]
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))

            sent_vec += (vec * tf_idf)   # (vector) x (tfidf value)

            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum

    tfidf_sent_vec_train.append(sent_vec) # first sentence transformed to a vector and appended
    row += 1

print(row) # To know the number of iterations.
print(len(tfidf_sent_vec_train)) # list of each review vector having 50 as dimension
print(len(tfidf_sent_vec_train[0])) # each review has been transformed to 50 dim vecctor
et = time.time()
print(et-st) # # time taken by the program.
```

```
100%|██████████████████████████████████████████████████████████████████████████| 3500/3500 [18:50<00:00,
3.09it/s]
3500
3500
50
1130.968782901764
```

(FOR TEST DATA):

In [114…

```
# compute TFIDF word2vec for each essay in test data
import time
st = time.time()

# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
tfidf_matrix = model.transform(X_test['essay'].values)# returns tfidf weighted values
```

```
df = pd.DataFrame(tfidf_matrix.toarray(),columns = tfidf_feat)

tfidf_sent_vec_test = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0; #--> To store number of iterations

list_of_sent = X_test['essay'].values

for idx,sent in tqdm(enumerate(list_of_sent)):
    sent_vec = np.zeros(50)
    weight_sum =0; # To store sum of tfidf values.
    for w,word in enumerate(sent): # for each word in a review/sentence
        if word in glove_words and word in tfidf_feat:
            vec = glovemodel[word]
            tf_idf = df.iloc[idx,w]      # grab the value of tfidf from df.
            sent_vec += (vec * tf_idf)   # (vector) x (tfidf value).
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum

    tfidf_sent_vec_test.append(sent_vec) # first sentence transformed to a vector and appended
    row += 1

print(row)
print(len(tfidf_sent_vec_test)) # list of each review vector having 50 as dimension
print(len(tfidf_sent_vec_test[0])) # each review has been transformed to 50 dim vecctor
et = time.time()
print(et-st) # time taken by the program.
```

```
1500it [07:03,  3.54it/s]
1500
1500
50
423.82789850234985
```

(E) SENTIMENT SCORES:

In [115…

```
# Define a function and call the function:
def sentiment_scores(lst):
    neg,neu,pos,compound = [],[],[],[]
    for sent in lst:
        sentiment_dict = sia.polarity_scores(sent)
```

```
        neg.append(sentiment_dict['neg'])
        neu.append(sentiment_dict['neu'])
        pos.append(sentiment_dict['pos'])
        compound.append(sentiment_dict['compound'])
    negative = pd.Series(neg)
    neutral = pd.Series(neu)
    positive = pd.Series(pos)
    compound = pd.Series(compound)
    features = {'Negative':negative,"Neutral":neutral,"Positive":positive,"Compound":compound}
    result = pd.DataFrame(features)
    return result

sia = SentimentIntensityAnalyzer()

# Pass Xtrain and Xtest into the above function to obtain scores:

lst_xtrain = X_train['essay'].values
lst_xtest = X_test['essay'].values

df_Xtrain = sentiment_scores(lst_xtrain)
df_Xtest = sentiment_scores(lst_xtest)
df_Xtrain.head()
```

Out[115…

	Negative	Neutral	Positive	Compound
0	0.022	0.810	0.168	0.9442
1	0.029	0.853	0.118	0.9431
2	0.051	0.569	0.381	0.9975
3	0.117	0.664	0.220	0.9638
4	0.022	0.735	0.243	0.9854

In [116…

```
# convert pandas dataframe to numpy arrays for fast computation:
sentimentscores_Xtrain = df_Xtrain.to_numpy()
sentimentscores_Xtest  = df_Xtest.to_numpy()

# Get the shape of the array:
print(sentimentscores_Xtrain.shape)
print(sentimentscores_Xtest.shape)
```

```
(3500, 4)
(1500, 4)
```

(F) CONCATENATING FEATURES:

In [117…

```
# concatenate all the features :

from scipy.sparse import hstack

# hstack() helps in concatenating "n" number of array like shapes into one dataframe.
# we store the concatenated outcome in a csr matrix format.
tfidf_w2v_essay_train = np.asarray(tfidf_sent_vec_train)
train_X = hstack((tfidf_w2v_essay_train,X_train_state_school,
                  X_train_prefix_teacher,X_train_grade_cat,
                 X_train_categories,X_train_subcategories,
                 X_train_price_norm,X_train_previous_norm,
                 sentimentscores_Xtrain)).tocsr()

tfidf_w2v_essay_test = np.asarray(tfidf_sent_vec_test)
test_X = hstack((tfidf_w2v_essay_test,X_test_state_school,
                  X_test_prefix_teacher,X_test_grade_cat,
                 X_test_categories,X_test_subcategories,
                 X_test_price_norm,X_test_previous_norm,
                sentimentscores_Xtest)).tocsr()

print("(H) Final Data matrix :")

print(train_X.shape, y_train.shape)#we totally have 3500 rows & 370 columns in train data

print(test_X.shape, y_test.shape)#we totally have 1500 rows & 370 columns in test data
```

```
(H) Final Data matrix :
(3500, 370) (3500,)
(1500, 370) (1500,)
```

(G) APPLYING DECISION TREE WITH HYPERPARAMETER TUNING:

In [118…

```
# build decision tree classifier

# import necessary libraries:
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

```
from time import time

start = time()

# fit the model to the training data using randomsearchCV
model = DecisionTreeClassifier(random_state = 10,class_weight="balanced")

param = {'max_depth':[1,5,10,50],'min_samples_split' : [5,10,100,500]}

# use "ROC_AUC" as a scoring and CV = 4
clf = GridSearchCV(model,param,cv=4,scoring='roc_auc',return_train_score = True)

for k in tqdm(range(0,1)):
    clf.fit(train_X,y_train)


print("The best paramters from gridsearchCv :",clf.best_params_)
print("\n")


# make a dataframe out of cv_results:
cv_results = pd.DataFrame.from_dict(clf.cv_results_)


# obtain mean train,Cv scores and their corresponding hyperparameters:
train_auc = cv_results['mean_train_score']
cv_auc = cv_results['mean_test_score']
depth =  cv_results['param_max_depth']
min_sample_split = cv_results['param_min_samples_split']


# lets observe the point of minimal gap between CV and train curves:
difference = train_auc - cv_auc
print("(A) Difference between train_auc & cv_auc :\n",difference)
print("\n")

print("(B) The minimal difference is :",min(difference))
print("\n")

print("(C) The minimal difference is observed at the depth :",depth[difference.idxmin()])
print("\n")
```

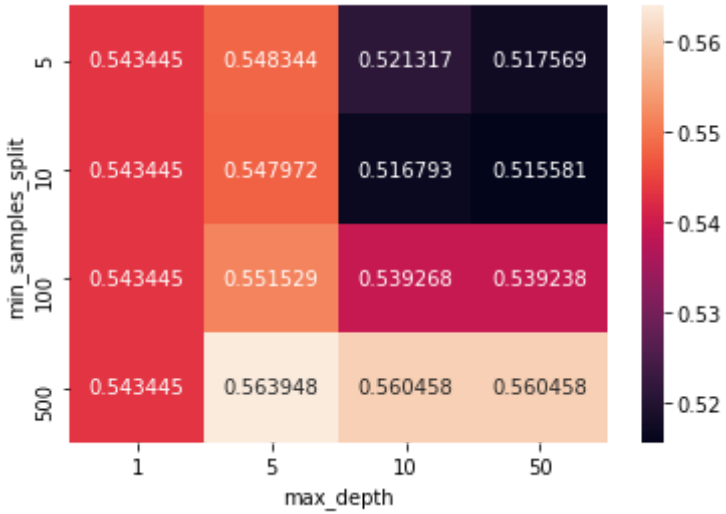
```
print("(D) The minimal difference is observed at the sample_split of :"
      ,min_sample_split[difference.idxmin()])
print("\n")


# plot the Hyperparameters vs AUC plot (heatmap) for train data:

df = pd.DataFrame({'AUC_score':list(cv_auc),
                   'max_depth':list(depth),
                   'min_samples_split':list(min_sample_split)})

result = df.pivot("min_samples_split","max_depth","AUC_score")
sns.heatmap(result,annot=True,fmt="f")
plt.show()


end = time()
training_time = end - start
print("training & validation time: %0.2fs" % training_time)
```

```
100%|██████████| 1/1 [00:02<00:00,
2.44s/it]
The best paramters from gridsearchCv : {'max_depth': 5, 'min_samples_split': 500}


(A) Difference between train_auc & cv_auc :
 0     0.015910
1     0.015910
2     0.015910
3     0.015910
4     0.152230
5     0.151328
6     0.133742
7     0.090442
8     0.359317
9     0.352427
10    0.234810
11    0.121076
12    0.481016
13    0.471216
14    0.279557
15    0.121076
dtype: float64
```

(B) The minimal difference is : 0.015910237962286966

(C) The minimal difference is observed at the depth : 1

(D) The minimal difference is observed at the sample_split of : 5



training & validation time: 2.81s

OBSERVATION :

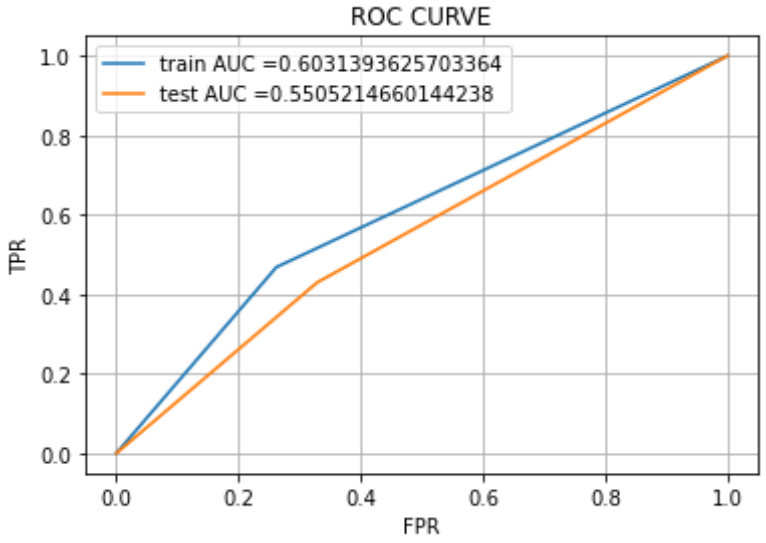
From the above results, we can see that minimal gap between cv_auc and train_auc is observed at max_depth = 1 and min_samples_split = 5, but as the auc value is low. Thus, we can choose max_depth = 5 and min_samples_split = 500 which provides us some reasonable high value of auc for both cv and train data with minimal gap of 0.090, hence,that is why the gridsearch implementation has chosen max_depth = 5 & min_samples_split = 500 as the best_params and now we go ahead for testing phase then interpret the results for test data.

(H) TEST PERFORMANCE OF THE MODEL:

In [123…

```
from sklearn.metrics import roc_curve, auc # -->necessary libraries

# Fit the classifier with the optimal alpha:
clf = DecisionTreeClassifier(max_depth = 5,min_samples_split = 500,
```

```
                                class_weight="balanced",random_state = 10)
clf.fit(train_X,y_train)

# predict for train and test data :
y_pred_train = clf.predict(train_X)
y_pred_test = clf.predict(test_X)

# compute TPR,FPR values to construt ROC curve:
train_fpr,train_tpr,tr_thresholds = roc_curve(y_train,y_pred_train)
test_fpr,test_tpr,te_thresholds = roc_curve(y_test,y_pred_test)

#plot the ROC with Train AUC and Test AUC:
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("ROC CURVE")
plt.grid()
plt.show()
```

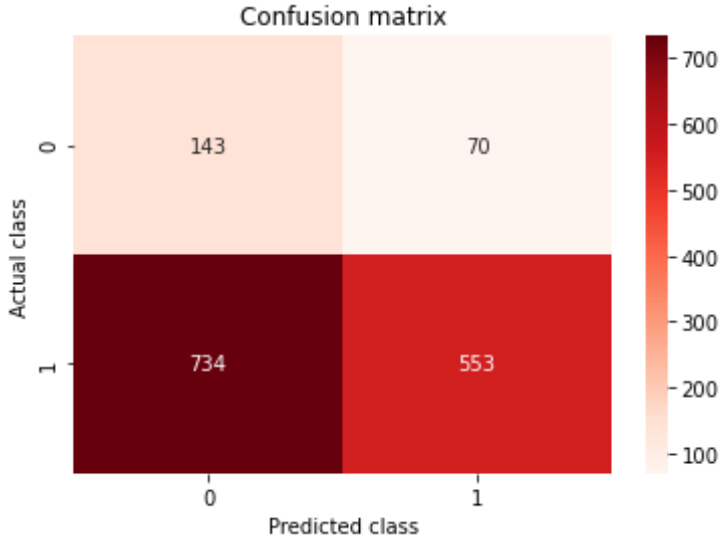


(I) PLOT CONFUSION MATRIX:

In [124…

```
# CONFUSION MATRIX :
from sklearn.metrics import confusion_matrix
confusion_mat = confusion_matrix(y_test,y_pred_test)
print(confusion_mat)

# Represent confusion matrix as a heatmap:
cm = np.array([[143,70],[734,553]])
sns.heatmap(cm,annot=True,fmt="d",cmap='Reds')
plt.xlabel("Predicted class")
plt.ylabel("Actual class")
plt.title("Confusion matrix")
plt.show()
```

```
[[143  70]
 [734 553]]
```



(I1) FETCH ALL POSITIVE POINTS:

In [125…

```
# first we initialize a list to store FP points.
false_positive_pts = []
for idx,j in enumerate(y_test):
    if ((j == 0) and (y_pred_test[idx] == 1)):
        false_positive_pts.append(idx)
    else:
        continue
```

```
print(len(false_positive_pts))

# extract "essay" values of false positive points and ensure its length is 477.
fp_essay = []
for index in false_positive_pts :
    fp_essay.append(X_test.iloc[index,6])

#print(len(fp_essay))


fp_price = []
for index in false_positive_pts :
    fp_price.append(X_test.iloc[index,7])

#print(len(fp_price))


fp_teacher_no_of_previously_posted_project = []
for index in false_positive_pts :
    fp_teacher_no_of_previously_posted_project.append(X_test.iloc[index,3])

#print(len(fp_teacher_no_of_previously_posted_project))
```

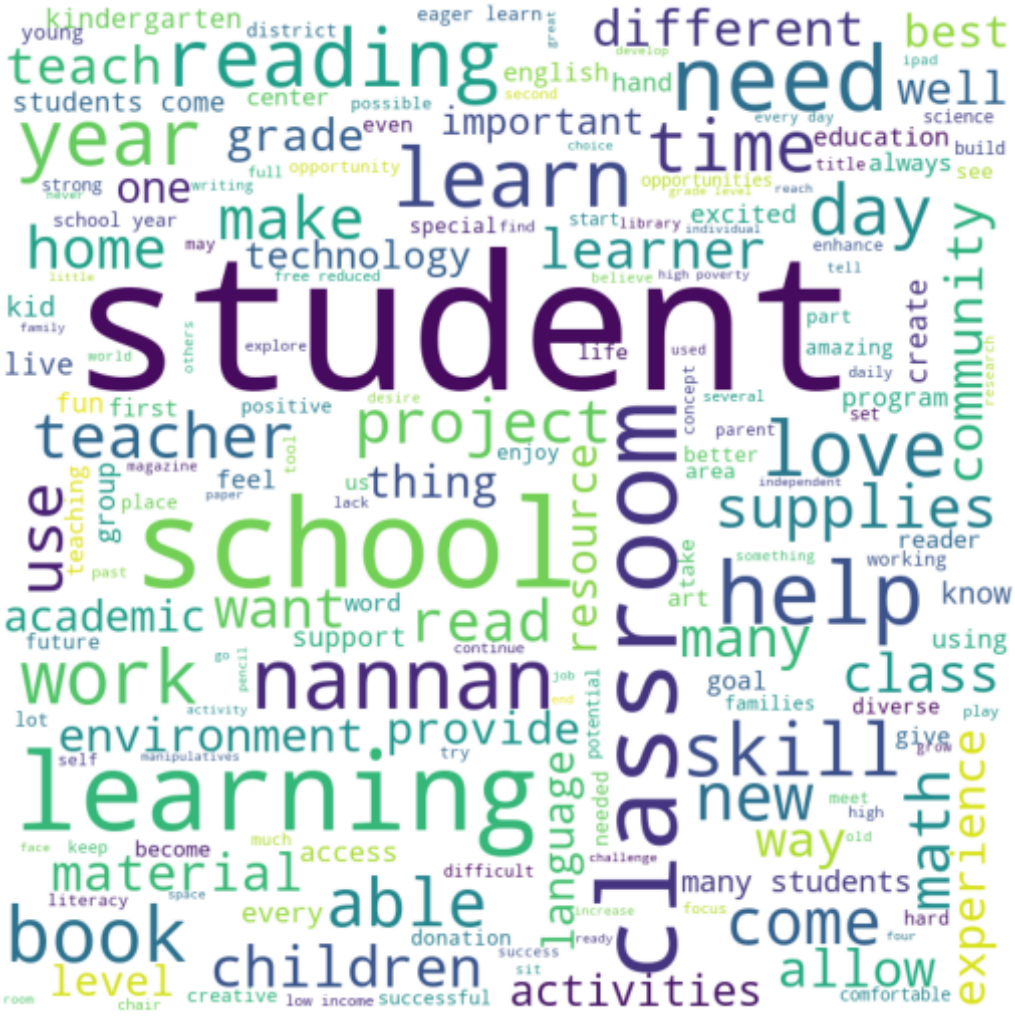
```
70
```

(I2) GENERATE WORD CLOUD FOR ESSAY(False positive pts):

In [126…

```
#REFERENCE :https://www.geeksforgeeks.org/generating-word-cloud-python/
# GENERATE WORDCLOUD:
from wordcloud import WordCloud, STOPWORDS

comment_words = ''
stopwords = set(STOPWORDS)

# iterate through the essay
for sent in fp_essay:

    # split the value
    tokens = sent.split()

    # Converts each token into lowercase
```

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

    comment_words += " ".join(tokens)+" "

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

# plot the WordCloud image
plt.figure(figsize = (7,7),facecolor = None)
plt.imshow(word_cloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



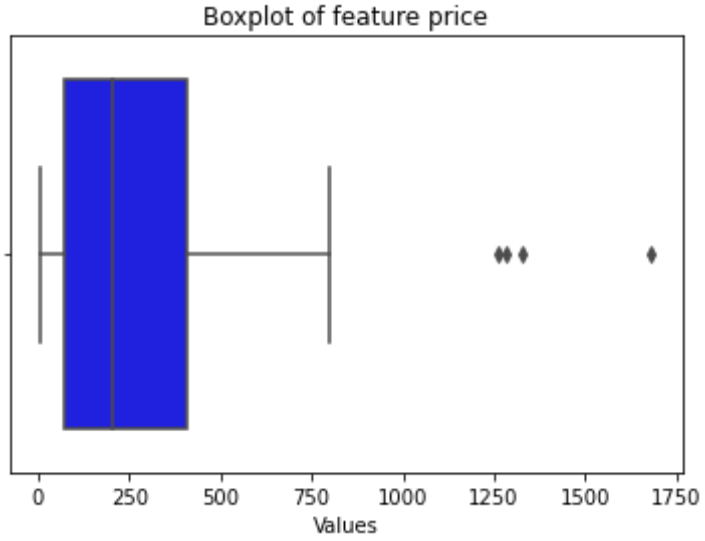
(I3) BOXPLOT OF FEATURE "PRICE"(false positive pts):

In [127…

```
# plot using seaborn.
sns.boxplot(x = fp_price,color = 'b')
plt.title("Boxplot of feature price")
plt.xlabel("Values")
```

Out[127…

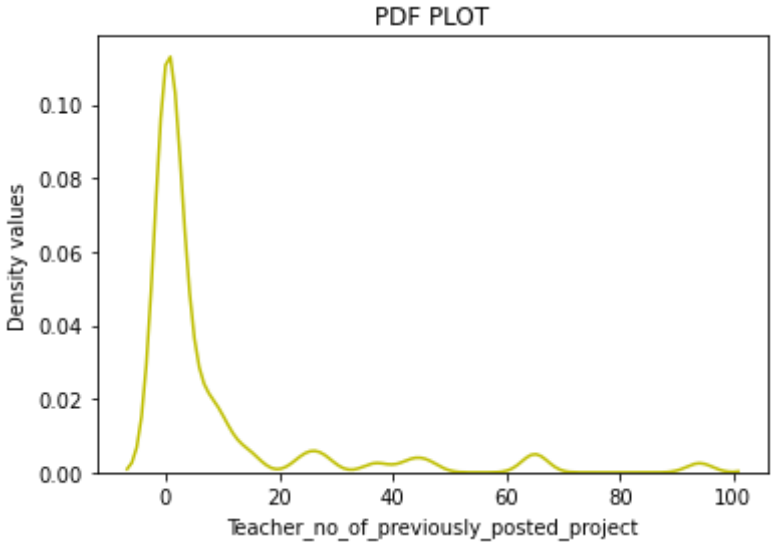
```
Text(0.5, 0, 'Values')
```



(I4) PDF OF FEATURE 'Teacher_no_of_previously_posted_projects' (False positive pts) :

In [128…
```
#plot using seaborn
sns.distplot(fp_teacher_no_of_previously_posted_project,hist=False,kde = True,color ='y')
plt.title("PDF PLOT")
plt.ylabel("Density values")
plt.xlabel("Teacher_no_of_previously_posted_project")
```

Out[128…
```
Text(0.5, 0, 'Teacher_no_of_previously_posted_project')
```



SUMMARY:

In [1]:
```
from prettytable import PrettyTable

# final results of the tasks:
# create a table with desired attributes:
summary = [["TFIDF","Decision Tree","10(max_depth)","0.603"],
           ["TFIDF-W2V","Decision Tree","5(max_depth)","0.55"]
           ,["TFIDF","Logistic regression","1(lambda)","0.64"]]

table = PrettyTable(["Vectorizer","Model","Hyperparameters","AUC"])

# add rows to the table:
for j in summary:
    table.add_row(j)

print(table)
```

```
+------------+---------------------+-----------------+-------+
| Vectorizer |        Model        | Hyperparameters |  AUC  |
+------------+---------------------+-----------------+-------+
|   TFIDF    |    Decision Tree    |  10(max_depth)  | 0.603 |
```

TFIDF-W2V	Decision Tree	5(max_depth)	0.55
TFIDF	Logistic regression	1(lambda)	0.64