Assignment : DT

Please check below video before attempting this assignment

In [8]:

```
from IPython.display import YouTubeVideo
YouTubeVideo('ZhLXULFjIjQ', width="1000",height="500")
```

Out[8]:

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 [9]:

```
import pickle
#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 [10]:

```
'''
# 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-sa
ve-and-load-variables-in-python/

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


'''
```

Out[10]:

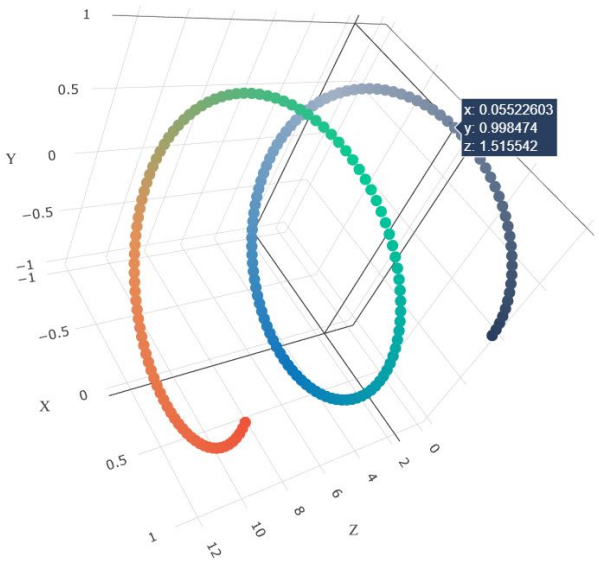

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
============================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus",
len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\n
print("word 2 vec length", len(words_courpus))\n\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
```

```
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

Task - 1

1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**

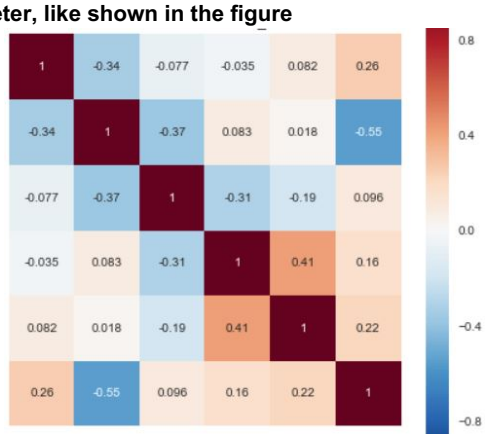
   - **Set 1: categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)**
   - **Set 2: categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)**
     ** **
   - **The hyper paramter tuning (best `depth` in range [1, 5, 10, 50], and the best `min_samples_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*

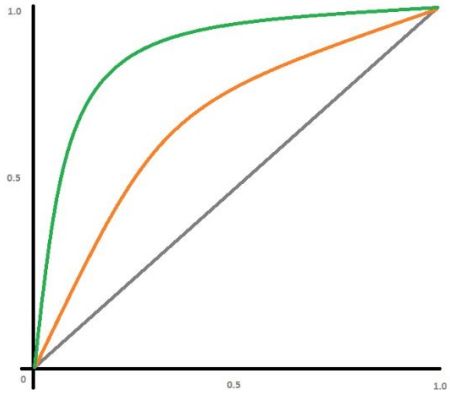
or

- **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 min_sample_split, 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 `false positive data points`
  - Plot the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) with the words of essay text of 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 `false positive data points`

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>
```


1. Decision Tree

In [12]:

```
# Imports

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

import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import Normalizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc, roc_auc_score
from tqdm import tqdm
from prettytable import PrettyTable
from pandas_ml import ConfusionMatrix
from sklearn.model_selection import GridSearchCV
import seaborn as sns

import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

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

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

Out[12]:

```
True
```

1.1 Loading Data

In [13]:

```
# Load dataset

import pandas
data = pandas.read_csv('preprocessed_data.csv', nrows=50000)
```

Task 1

In [14]:

```
# Seprate X and y

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

Out[14]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories
0	ca	mrs	grades_prek_2	53	math_science

In [15]:

```
# Split into trainig and test set and again split the training set to train and cv
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

Function to add sentiment score for essays

In [16]:

```
# Function to be applied with indevidual train, cv and test data
def getSentimentVector(essays):
    """This function gives the sentiment measures for text data"""

    sid = SentimentIntensityAnalyzer()
    output = dict()
    # SentimentIntensityAnalyzer results values for below keys
    columns = ['neg', 'neu', 'pos', 'compound']
    neg = []
    neu = []
    pos = []
    compound = []

    # Get scores for each essays
    for essay in essays:
        ss = sid.polarity_scores(essay)
        values = list(ss.values())
        neg.append(values[0])
        neu.append(values[1])
        pos.append(values[2])
        compound.append(values[3])

    # combine all the data and return
    data = []
    for value in list(zip(neg, neu, pos, compound)):
        data.append(list(value))

    output['columns'] = columns
    output['values'] = np.array(data)
    return output
```

IFIDF vectorizer for text(essay) vectorization

In [17]:

```
def encodeEssayTFIDF(trainText, testText):
    """This function returns the encoded vectors for train, cv and test data with
TfidfVectorizer"""

    output = dict()
    vectorizer = TfidfVectorizer(min_df = 10, max_features = 5000)

    # Fit the vectorizer with train data and trasnform all train, cv and test texts
    train_vec = vectorizer.fit_transform(trainText)
    output['columns'] = vectorizer.get_feature_names()
    test_vec = vectorizer.transform(testText)

    # Add all enocoded vectorized data and bows into a dict to return it
    output['train_vec'] = train_vec
    output['test_vec'] = test_vec
    output['columns'] = vectorizer.get_feature_names()

    return output
```

TFIDF_W2V vectorization of text(essay)

In [18]:

```
def getW2C(texts, tfidf_words, dictionary):
    """This function returns W2V response for given TFIDF data and texts"""

    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(texts): # 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
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting
the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)
    return np.array(tfidf_w2v_vectors)
```

In [19]:

```
def encodeEssayW2VTFIDF(trainText, testText):
    """This function returns the encoded vectors for train, cv and test data with
TfidfVectorizer"""

    output = dict()

    # Got TFIDF model and use it with W2V
    tfidf = TfidfVectorizer()
    tfidf.fit(trainText)
    dictionary = dict(zip(tfidf.get_feature_names(), list(tfidf.idf_)))
    tfidf_words = set(tfidf.get_feature_names())

    # Get w2V vectors for train, test and cv
    train_vec = getW2C(trainText, tfidf_words, dictionary)
    test_vec = getW2C(testText, tfidf_words, dictionary)

    # As w2V doesnt give the feature names as it only gives d-dim vector for a word, we are
manually adding columns
    columns = []
    for i in range(300):
        c = 'w' + str(i + 1)
        columns.append(c)

    # Retun the results in dict format
    output['columns'] = columns
    output['train_vec'] = train_vec
    output['test_vec'] = test_vec

    return output
```

Apply OneHotEncoding with Counervectorizer for categorical values

In [20]:

```
def encodeCategoricalValues(train_cat, test_cat):
    """This function takes any categorical feature values for train, cv and text
    and applies One Hot Encoding using CountVectorizer"""

    vectorizer = CountVectorizer()
    output = dict()

    # Use vectorizer to fit and transform the input categories
    Xtrain_cat = vectorizer.fit_transform(train_cat)
    output['columns'] = vectorizer.get_feature_names()
    Xtest_cat = vectorizer.transform(test_cat)

     # Add all enocoded data and categorical values into a dict to return it
    output['train_vec'] = Xtrain_cat
    output['test_vec'] = Xtest_cat

    return output
```

Normalize the numerical data

In [21]:

```
def numericNormalizer(train_val, test_val, column):
    """This function normalizes numerical values"""
    normalizer = Normalizer()
    output = dict()

    normalizer.fit(train_val.reshape(-1,1))
    x_train_val = normalizer.transform(train_val.reshape(-1,1))
    x_test_val = normalizer.transform(test_val.reshape(-1,1))

    # Add all enocoded vectorized data and bows into a dict to return it
    output['train_val'] = x_train_val
    output['test_val'] = x_test_val
    output['column'] = column

    return output
```

Transform all train, cv and test data by applying all feature transform functions

In [22]:

```
from scipy.sparse import hstack

# Transform data (text, categotical and numerical data)
def vectorizeDataset(X_train, X_test, setType):
    """This function vectorizes the feature values"""

    columns = []

    trainEssay = X_train['essay'].values
    testEssay = X_test['essay'].values
    trainState = X_train['school_state'].values
    testState = X_test['school_state'].values
    trainPrefix = X_train['teacher_prefix'].values
    testPrefix = X_test['teacher_prefix'].values
    trainGrade = X_train['project_grade_category'].values
    testGrade = X_test['project_grade_category'].values
    trainCategory = X_train['clean_categories'].values
    testCategory = X_test['clean_categories'].values
    trainSubCategories = X_train['clean_subcategories'].values
    testSubCategories = X_test['clean_subcategories'].values
    trainPrevProjects = X_train['teacher_number_of_previously_posted_projects'].values
    testPrevProjects = X_test['teacher_number_of_previously_posted_projects'].values
    trainPrice = X_train['price'].values
    testPrice = X_test['price'].values

    # vectorize essays
    output_essay = dict()

    if (setType == 1):
        output_essay = encodeEssayTFIDF(trainEssay, testEssay)
    elif (setType == 2):
        output_essay = encodeEssayW2VTFIDF(trainEssay, testEssay)


    x_train_essay = output_essay['train_vec']
    x_test_essay = output_essay['test_vec']
    columns += output_essay['columns']

    # Get sentiment scores for essays
    output_sentiment = getSentimentVector(trainEssay)
    x_train_sentiment = output_sentiment['values']
    x_test_sentiment = getSentimentVector(testEssay)['values']
    columns += output_sentiment['columns']



    # One hot encode for school state
    output_state = encodeCategoricalValues(trainState, testState)
    x_train_state = output_state['train_vec']
    x_test_state = output_state['test_vec']
    columns += output_state['columns']

    # One hot encode for teacher prefix
    output_prefix = encodeCategoricalValues(trainPrefix, testPrefix)
    x_train_prefix = output_prefix['train_vec']
```

```
    x_test_prefix = output_prefix['test_vec']
    columns += output_prefix['columns']

    # One hot encode fot project grades
    output_grade = encodeCategoricalValues(trainGrade, testGrade)
    x_train_grade = output_grade['train_vec']
    x_test_grade = output_grade['test_vec']
    columns += output_grade['columns']

    # One hot encode fot project categories
    output_Category = encodeCategoricalValues(trainCategory, testCategory)
    x_train_category = output_Category['train_vec']
    x_test_category = output_Category['test_vec']
    columns += output_Category['columns']

    # One hot encode fot project sub categories
    output_subCategory = encodeCategoricalValues(trainSubCategories, testSubCategories)
    x_train_subCategory = output_subCategory['train_vec']
    x_test_subCategory = output_subCategory['test_vec']
    columns += output_subCategory['columns']

    # Normalize previous project numbers
    output_projectNumber = numericNormalizer(trainPrice, testPrice,
'teacher_number_of_previously_posted_projects')
    x_train_number = output_projectNumber['train_val']
    x_test_number = output_projectNumber['test_val']
    columns.append(output_projectNumber['column'])

    # Normalize project price
    output_price = numericNormalizer(trainPrice, testPrice, 'price')
    x_train_price = output_price['train_val']
    x_test_price = output_price['test_val']
    columns.append(output_price['column'])



    #Combine all vectorized features and return final train, cv and test sets and columns

    X_train = hstack((x_train_essay, x_train_sentiment, x_train_state, x_train_prefix, x_train_grad
e, x_train_category, x_train_subCategory,\
                      x_train_number, x_train_price)).tocsr()
    X_test = hstack((x_test_essay, x_test_sentiment, x_test_state, x_test_prefix, x_test_grade, x_t
est_category, x_test_subCategory, \
                     x_test_number, x_test_price)).tocsr()

    print("Final Data matrix with train, cv and test")
    print(X_train.shape, Y_train.shape)
    print(X_test.shape, Y_test.shape)
    print(len(columns))

    return X_train, X_test, columns
```

In [23]:

```
X_Train_set1, X_Test_set1, columns_set1 = vectorizeDataset(X_train, X_test, 1)
X_Train_set2, X_Test_set2, columns_set2 = vectorizeDataset(X_train, X_test, 2)
```

```
Final Data matrix with train, cv and test
(33500, 5105) (33500,)
(16500, 5105) (16500,)
5105
```

```
100%|██████████| 33500/33500 [02:57<00:00, 188.99it/s]
100%|██████████| 16500/16500 [01:26<00:00, 191.52it/s]
```

```
Final Data matrix with train, cv and test
(33500, 405) (33500,)
(16500, 405) (16500,)
405
```

Apply GridSearchCV to get best hyper parameter for classification

In [24]:

```
def getBestModelDecisionTree(X_train, Y_train):

    decision = DecisionTreeClassifier(class_weight= 'balanced')
    # Hyper parameters max_depth and min_samples_split
    parameters = {'min_samples_split': [5, 10, 100, 500], 'max_depth': [1, 5, 10, 50]}

    # Apply GridSearchCV to get auc scores for train and cv data with different hyper parameter va
lues
    clf = GridSearchCV(decision, parameters, cv=3, scoring='roc_auc')
    clf.fit(X_train, Y_train)

    # Get the result and plot in 3D (parameters and auc score)
    results = pd.DataFrame.from_dict(clf.cv_results_)
    results = results.sort_values(['param_max_depth', 'param_min_samples_split'])
    train_auc= results['mean_train_score']
    cv_auc = results['mean_test_score']
    min_samples_split =  results['param_min_samples_split']
    max_depth = results['param_max_depth']

    train_auc_plot = go.Scatter3d(x = max_depth, y = min_samples_split, z = train_auc, name = 'trai
n auc')
    cv_auc_plot = go.Scatter3d(x = max_depth, y = min_samples_split, z = cv_auc, name = 'cv auc')
    data = [train_auc_plot, cv_auc_plot]

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

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

In [26]:

```
# For SET-1 data

getBestModelDecisionTree(X_Train_set1, Y_train)
```

In [27]:

```
# For SET-2 data

getBestModelDecisionTree(X_Train_set2, Y_train)
```

From the 3D plot, we can see, the maximum cv_auc by minimizing the train cv auc difference occurs at max_depth as 5 and min_samples_split as 500.

Train and model with best hyper parameter

In [28]:

```
# In ROC_AUC scores, there are number of threshold values.
# Among them best one has to be chosed to predict the class levels from the probability scores.
def get_best_threshold(threshoulds, fpr, tpr):
    """This function takes threshould values with TPR and FPR values and calculates best threshoul
d"""
    # Choose best threshould such that TPR is more and FPR is less
    threshould = threshoulds[np.argmax(tpr*(1-fpr))]
    print("best threshould",threshould)
    return threshould

# Predict the class levels with best threshould
def predict_class_levels(proba, threshould):
    """This function takes best threshould value and probability scores and predict class
levels"""
    predicted_class_levels = []
    for i in proba:
        if i>=threshould:
            predicted_class_levels.append(1)
        else:
            predicted_class_levels.append(0)
    return predicted_class_levels
```

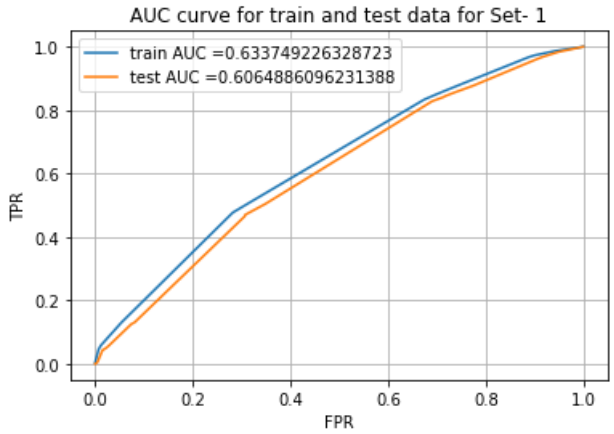
In [29]:

```
# Predict y-test values with best hyper parameter

def bestParamClassificatationAndAUC(X_train, X_test, Y_train, Y_test, max_depth, min_samples_split,
setType):
    """This function does train and test on a model with best values of hyper paramters,
```

```
    and given ROC_AUC curve and value. It returns best threshould and predicted probability values
for train and test data"""

    # Create and train Decision tree with best values of hyper parameters
    dc = DecisionTreeClassifier(class_weight= 'balanced', max_depth = max_depth, min_samples_split
= min_samples_split)
    dc.fit(X_train, Y_train)

    # Predict class level probabilities for train and test
    y_proba_train = dc.predict_proba(X_train)[:, 1]
    y_proba_test = dc.predict_proba(X_test)[:,1]

    # Get TPR, FPR and threshold values for train and test probability values
    train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_proba_train)
    test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_proba_test)

    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("AUC curve for train and test data for Set- {}".format(setType))
    plt.grid()
    plt.show()

    threshould = get_best_threshold(tr_thresholds, train_fpr, train_tpr)
    y_pred = predict_class_levels(y_proba_test, threshould)

    result = dict()
    result["y_pred"] = y_pred
    result['auc'] = auc(test_fpr, test_tpr)

    return result
```

In [30]:

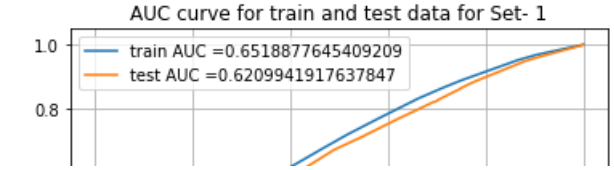
```
# Get classification result with best hyper parameter for SET1

result_set1 = bestParamClassificatationAndAUC(X_Train_set1, X_Test_set1, Y_train, Y_test, 5, 500, 1
)
```

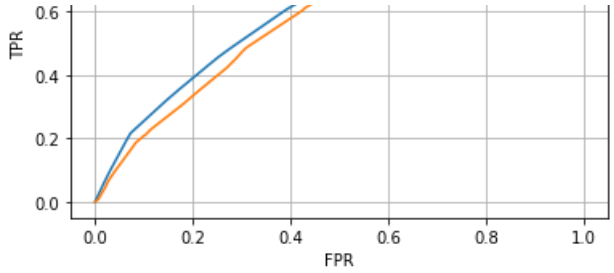



```
best threshould 0.4776869677905458
```

In [31]:

```
# Get classification result with best hyper parameter for SET2

result_set2 = bestParamClassificatationAndAUC(X_Train_set2, X_Test_set2, Y_train, Y_test, 5, 500, 1
)
```






```
best threshould 0.5122249896222522
```

Both train and test roc-auc aurve are similar and the auc score is similar. So wirh best hyper paremeters, the model works well for both train and test data.

But with TFIDF_W2V, the train and test auc score is more than the TFIDF vector.

Confusion matrxix

In [37]:

```
# For SET1

y_pred_set1 =  result_set1['y_pred']
cmSet1 = pd.DataFrame(confusion_matrix(Y_test, y_pred_set1), columns = ['Y_pred-0', 'Y_pred-1'])
cmSet1.index = ['Y_actual-0', 'Y_actual-1']
cmSet1
```

Out[37]:

	Y_pred-0	Y_pred-1
Y_actual-0	1721	921
Y_actual-1	6870	6988

In [35]:

```
# For SET1

y_pred_set2 =  result_set2['y_pred']
cmSet2 = pd.DataFrame(confusion_matrix(Y_test, y_pred_set2), columns = ['Y_pred-0', 'Y_pred-1'])
cmSet2.index = ['Y_actual-0', 'Y_actual-1']
cmSet2
```

Out[35]:

	Y_pred-0	Y_pred-1
Y_actual-0	1518	1124
Y_actual-1	5482	8376

False positive data visualization

In [38]:

```
!pip install wordcloud
```

```
Requirement already satisfied: wordcloud in c:\users\hp\anaconda3\lib\site-packages (1.8.0)
Requirement already satisfied: matplotlib in c:\users\hp\anaconda3\lib\site-packages (from
wordcloud) (2.2.2)
Requirement already satisfied: numpy>=1.6.1 in c:\users\hp\anaconda3\lib\site-packages (from
wordcloud) (1.14.3)
Requirement already satisfied: pillow in c:\users\hp\anaconda3\lib\site-packages (from wordcloud)
(5.1.0)
```

```
Requirement already satisfied: cycler>=0.10 in c:\users\hp\anaconda3\lib\site-packages (from
matplotlib->wordcloud) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
c:\users\hp\anaconda3\lib\site-packages (from matplotlib->wordcloud) (2.2.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\hp\anaconda3\lib\site-packages
(from matplotlib->wordcloud) (2.7.3)
Requirement already satisfied: pytz in c:\users\hp\anaconda3\lib\site-packages (from matplotlib->w
ordcloud) (2018.4)
Requirement already satisfied: six>=1.10 in c:\users\hp\anaconda3\lib\site-packages (from
matplotlib->wordcloud) (1.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\hp\anaconda3\lib\site-packages (from
matplotlib->wordcloud) (1.0.1)
Requirement already satisfied: setuptools in c:\users\hp\anaconda3\lib\site-packages (from
kiwisolver>=1.0.1->matplotlib->wordcloud) (39.1.0)
```

```
mysql-connector-python 8.0.21 requires protobuf>=3.0.0, which is not installed.
distributed 1.21.8 requires msgpack, which is not installed.
You are using pip version 10.0.1, however version 20.2.3 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

In [39]:

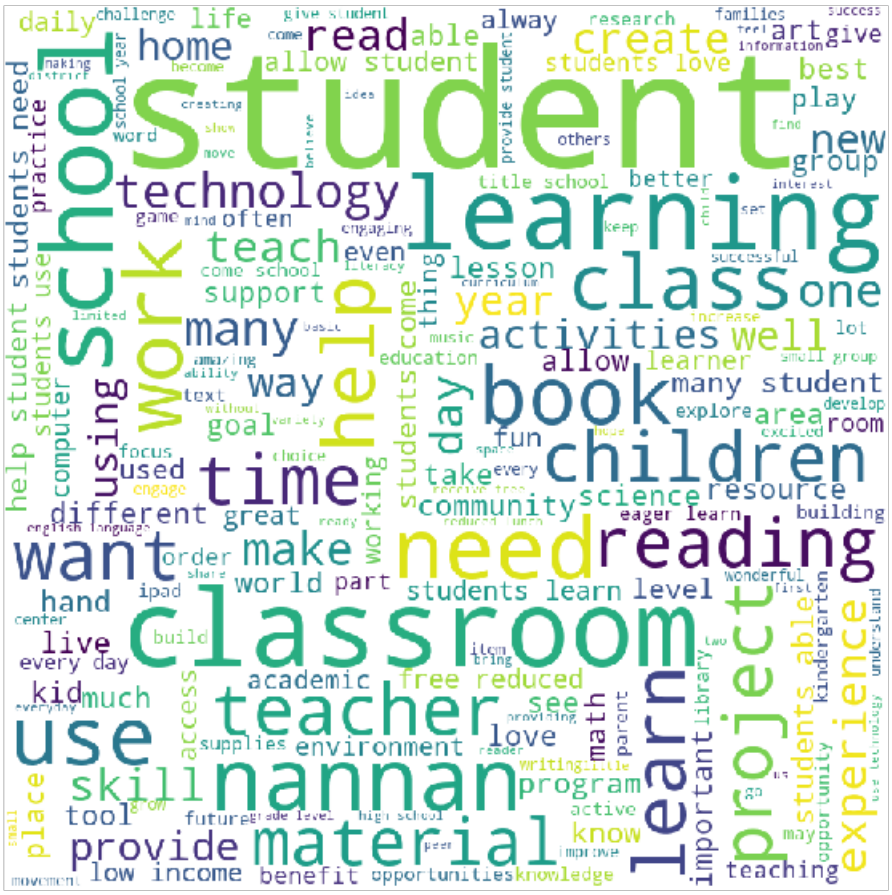
```
from wordcloud import WordCloud, STOPWORDS
```

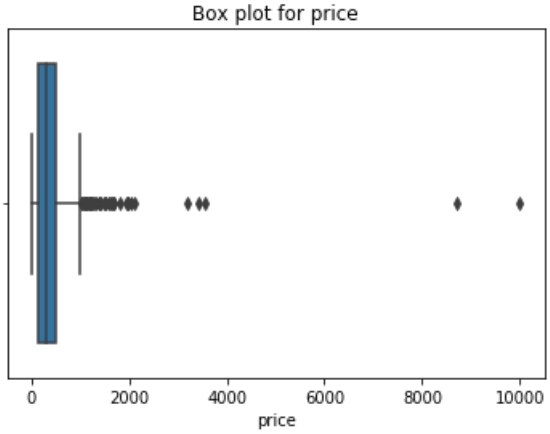
In [40]:

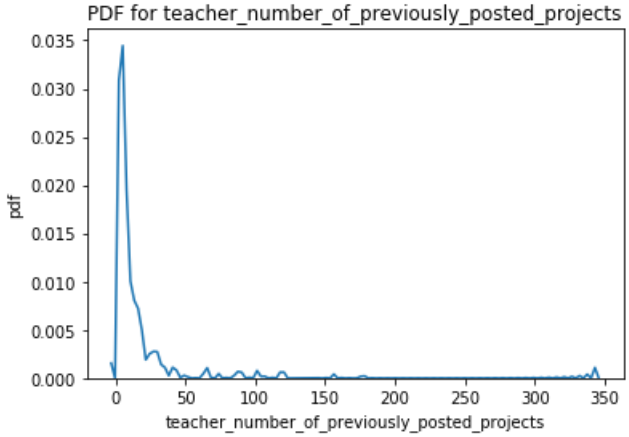
```
def visualize_FP(x_test, y_test, y_pred):
    """This function gives the false positive points"""

    # Get column values of FP data points
    y_pred = np.array(y_pred)
    x_test_fp = x_test[(y_test == 0) & (y_pred == 1)]
    x_test_essay = x_test_fp['essay'].values
    x_test_price = x_test_fp['price'].values
    x_test_posted = x_test_fp['teacher_number_of_previously_posted_projects'].values

    # Word Cloud of essays
    stopwords = set(STOPWORDS)

    words = []
    for sentence in list(x_test_essay):
        for word in sentence.split(' '):
            words.append(word)

    all_words = ''
    all_words += " ".join(words)+" "

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

    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad = 0)

    plt.show()

    # Box plot of price values
    ax1 = sns.boxplot(x=x_test_price)
    ax1.set(xlabel='price')
    plt.title("Box plot for price")
    plt.show()

    # Plot PDF for teacher_number_of_previously_posted_projects
    ax = sns.kdeplot(data=x_test_posted)
    ax.set(xlabel='teacher_number_of_previously_posted_projects', ylabel='pdf')
    plt.title("PDF for teacher_number_of_previously_posted_projects")
    plt.show()
```

In [41]:

```
visualize_FP(X_test, Y_test, y_pred_set1)
```







1-> Words like student, teacher, classroom, technology etc are frequently occuring, which add values to the text.

2-> Most of the projects have the price in between 0-100, where maximum price can be 1000/-

3-> Teachers having less projects submission counts are mostly sending new project for approvals.

Task 2

In [42]:

```
dc_clf = DecisionTreeClassifier(class_weight= 'balanced', min_samples_split = 500)
dc_clf.fit(X_Train_set1, Y_train)

# get the feature importance array
feature_importances = dc_clf.feature_importances_

# List out the index for which the value is not 0
idx = [i for i in range(len(feature_importances)) if feature_importances[i] != 0]

# Select only non-zero features from the train, cv and test data
X_Train_MF = X_Train_set1[:,idx]
X_Test_MF = X_Test_set1[:, idx]

print(X_Train_MF.shape)
print(X_Test_MF.shape)
```

```
(33500, 422)
(16500, 422)
```

In [43]:

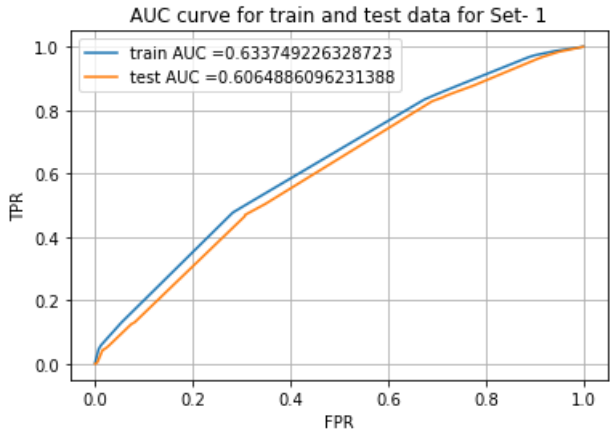
```
# Test with different hyper parameters and the new dataset

getBestModelDecisionTree(X_Train_MF, Y_train)
```

Here also the max_depth will be 5 and min_samples_split will be 500. But here the cv auc score is more with non-zero feature imporatance.

In [44]:

```
# Apply model with best hyper parameter and  features with non-zero feature importance
```

```
result_MF = bestParamClassificatationAndAUC(X_Train_MF, X_Test_MF, Y_train, Y_test, 5, 500, 1)
```




```
best threshould 0.4776869677905458
```

Now with important features, the test auc has been increased with few amount.

In [45]:

```
table = PrettyTable()
table.field_names = ["vectorizer", "Model", "Hyper parameter", "AUC"]
table.add_row(["TFIDF", "Decision Tree", 'Depth = 5, min_samples_split = 500', round(result_set1['
auc'],3)])
table.add_row(["TFIDF_W2V", "Decision Tree", 'Depth = 5, min_samples_split = 500',
round(result_set2['auc'],3)])
table.add_row(["TFIDF_nonZero_feature_importance", "Decision Tree", 'Depth = 5, min_samples_split
= 500', round(result_MF['auc'],3)])
print(table)
```

```
+-----------------------------------+---------------+-----------------------------------+-------+
|            vectorizer             |     Model     |          Hyper parameter          |  AUC  |
+-----------------------------------+---------------+-----------------------------------+-------+
|              TFIDF                | Decision Tree | Depth = 5, min_samples_split = 500 | 0.606 |
|            TFIDF_W2V              | Decision Tree | Depth = 5, min_samples_split = 500 | 0.621 |
| TFIDF_nonZero_feature_importance | Decision Tree | Depth = 5, min_samples_split = 500 | 0.606 |
+-----------------------------------+---------------+-----------------------------------+-------+
```