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 (https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/2916/avg-word2vec-tf-idf-weighted-word2vec/3/module-3-foundations-of-natural-language-processing-and-machine-learning)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 (https://drive.google.com/file/d/1lDca_ge-GYO0iQ6_XDLWePQFMdAA2b8f/view?usp=sharing)

In [2]:
```
%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

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

In [3]:
```
#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 [4]:
```
'''
# 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
-variables-in-python/

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


'''
```

Out[4]:
```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef loadGloveModel(glove
File):\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        model[word] = embedding\n    print ("Done.",len(mode
l)," words loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.txt\')\n\n# ==========
==========\nOutput:\n    \nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loade
d!\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", l
en(words))\nwords = set(words)\nprint("the unique words in the coupus", len(words))\n\ninter_words = set(mod
el.keys()).intersection(words)\nprint("The number of words that are present in both glove vectors and our co
upus",       len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nw
ords_glove = set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model
[i]\nprint("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python: htt
p://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pickle\nwith open
(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

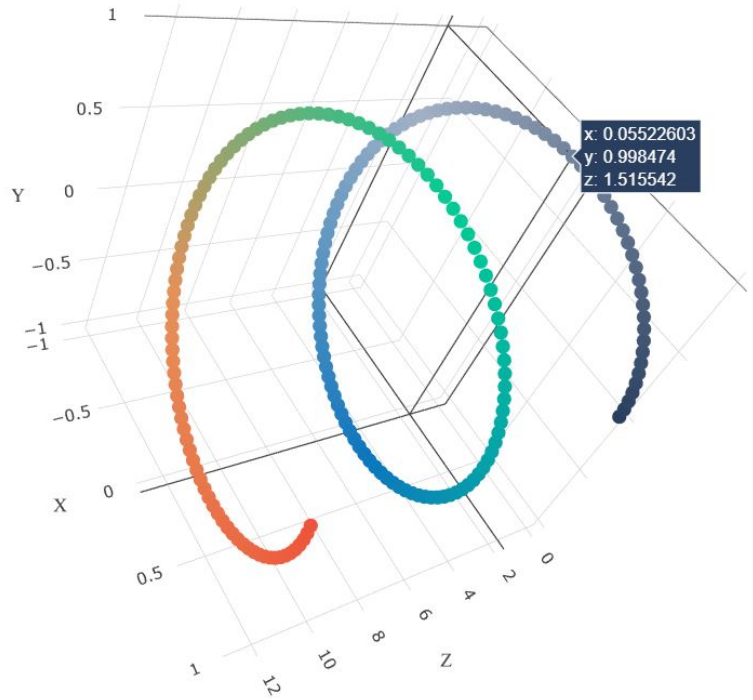
Task - 1

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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) 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)

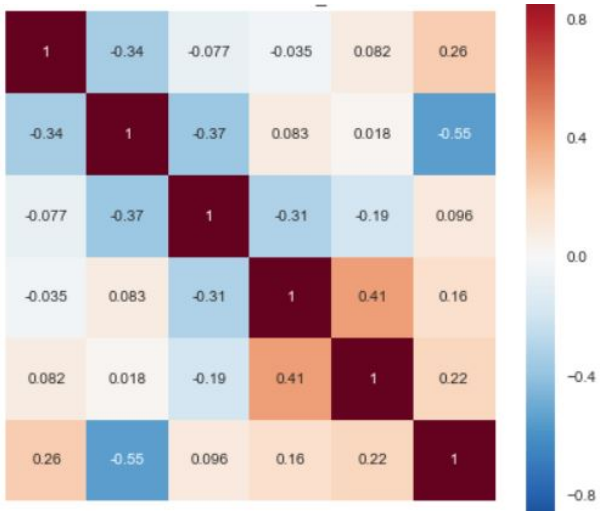
        - Representation of results
            - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



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

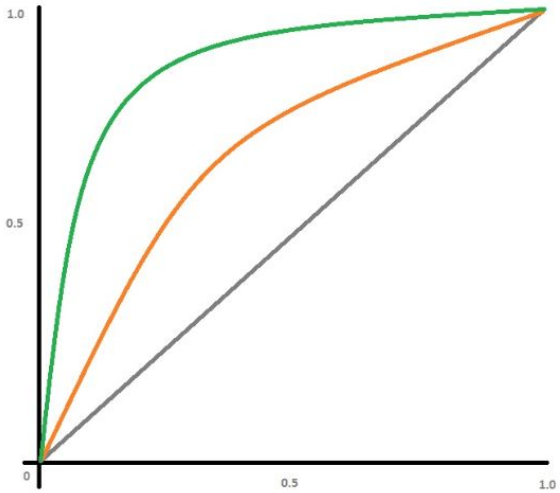
or

- 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 (https://seaborn.pydata.org/generated/seaborn.heatmap.html) 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

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

In [5]:
```
data = pd.read_csv('preprocessed_data.csv', nrows=40000) # only using these many rows because it is computati
onally very expensive to use more
```

In [6]:
```
from sklearn.model_selection import train_test_split
X = data.drop(['project_is_approved'], axis=1)
y = data['project_is_approved'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.25, stratify=y_train)
```

In [7]:
```
preprocessed_essays = data['essay'].values
```

SET 1 (TF-IDF)

In [8]:
```
print("Before vectorizations: \n")
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

vectorizer_essay_tfidf = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer_essay_tfidf.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer_essay_tfidf.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer_essay_tfidf.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer_essay_tfidf.transform(X_test['essay'].values)

print("After vectorizations: \n")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
Before vectorizations:

(24000, 8) (24000,)
(8000, 8) (8000,)
(8000, 8) (8000,)
====================================================================================================
After vectorizations:

(24000, 5000) (24000,)
(8000, 5000) (8000,)
(8000, 5000) (8000,)
```

SET 2 (TF-IDF W2V)

In [9]:
```
tfidf_model = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)

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

In [10]:
```
# average Word2Vec
# compute average word2vec for each review.
X_train_essay_tfidf_w2v = []
X_cv_essay_tfidf_w2v = []
X_test_essay_tfidf_w2v = []

# the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len
(sentence.split())))
            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
    X_train_essay_tfidf_w2v.append(vector)

for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len
(sentence.split())))
            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
    X_cv_essay_tfidf_w2v.append(vector)

for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
```

```
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len
(sentence.split())))
            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
    X_test_essay_tfidf_w2v.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████| 24000/24000 [00:54<00:00,
443.02it/s]
100%|████████████████████████████████████████████████████████████████| 8000/8000 [00:15<00:00,
501.21it/s]
100%|████████████████████████████████████████████████████████████████| 8000/8000 [00:15<00:00,
505.19it/s]
```

In [11]:
```
print(len(X_train_essay_tfidf_w2v))
print(len(X_train_essay_tfidf_w2v[0]))
```

```
24000
300
```

Encoding categorical features: teacher_prefix

In [12]:
```
vectorizer_teacher = CountVectorizer()
vectorizer_teacher.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_teacher_ohe = vectorizer_teacher.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer_teacher.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer_teacher.transform(X_test['teacher_prefix'].values)

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

```
After vectorizations
(24000, 5) (24000,)
(8000, 5) (8000,)
(8000, 5) (8000,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

Encoding categorical features: project_grade_category

In [13]:
```
vectorizer_grade_cat = CountVectorizer()
vectorizer_grade_cat.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_ohe = vectorizer_grade_cat.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer_grade_cat.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer_grade_cat.transform(X_test['project_grade_category'].values)

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

```
After vectorizations
(24000, 4) (24000,)
(8000, 4) (8000,)
(8000, 4) (8000,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================================
```

Encoding categorical features: school_state

In [14]:
```
vectorizer_state = CountVectorizer()
vectorizer_state.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_ohe = vectorizer_state.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer_state.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer_state.transform(X_test['school_state'].values)

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

```
After vectorizations
(24000, 51) (24000,)
(8000, 51) (8000,)
(8000, 51) (8000,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky',
'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh',
'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
====================================================================================================
```

Encoding categorical features: clean_categories

In [15]:
```
vectorizer_cat = CountVectorizer()
vectorizer_cat.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_cat_ohe = vectorizer_cat.transform(X_train['clean_categories'].values)
X_cv_cat_ohe = vectorizer_cat.transform(X_cv['clean_categories'].values)
X_test_cat_ohe = vectorizer_cat.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_cat_ohe.shape, y_train.shape)
print(X_cv_cat_ohe.shape, y_cv.shape)
print(X_test_cat_ohe.shape, y_test.shape)
print(vectorizer_cat.get_feature_names())
print("="*100)
```

```
After vectorizations
(24000, 7) (24000,)
(8000, 7) (8000,)
(8000, 7) (8000,)
['appliedlearning', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_arts', 's
pecialneeds']
====================================================================================================
```

Encoding categorical features: clean_subcategories

In [16]:
```
vectorizer_subcat = CountVectorizer()
vectorizer_subcat.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_subcat_ohe = vectorizer_subcat.transform(X_train['clean_subcategories'].values)
X_cv_subcat_ohe = vectorizer_subcat.transform(X_cv['clean_subcategories'].values)
X_test_subcat_ohe = vectorizer_subcat.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subcat_ohe.shape, y_train.shape)
print(X_cv_subcat_ohe.shape, y_cv.shape)
print(X_test_subcat_ohe.shape, y_test.shape)
print(vectorizer_subcat.get_feature_names())
print("="*100)
```

```
After vectorizations
(24000, 28) (24000,)
(8000, 28) (8000,)
(8000, 28) (8000,)
['appliedsciences', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice', 'ea
rlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreig
nlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'liter
ature_writing', 'mathematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingart
s', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts']
====================================================================================================
```

Encoding numerical features: price

In [17]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

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


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

```
After vectorizations
(24000, 1) (24000,)
(8000, 1) (8000,)
(8000, 1) (8000,)
====================================================================================================
```

Encoding numerical features: teacher_number_of_previously_posted_projects

In [18]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

X_train_pre_projects_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].valu
es.reshape(-1,1))
X_cv_pre_projects_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.res
hape(-1,1))
X_test_pre_projects_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values
.reshape(-1,1))


print("After vectorizations")
print(X_train_pre_projects_norm.shape, y_train.shape)
print(X_cv_pre_projects_norm.shape, y_cv.shape)
print(X_test_pre_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(24000, 1) (24000,)
(8000, 1) (8000,)
(8000, 1) (8000,)
====================================================================================================
```

Hint for calculating Sentiment scores

In [19]:
```
import nltk
nltk.download('vader_lexicon')
```

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

Out[19]:
```
True
```

In [20]:
```
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 big
gest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelligences i us
e a wide range\
of techniques to help all my students succeed students in my class come from a variety of different backgroun
ds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a caring community
of successful \
learners which can be seen through collaborative student project based learning in and out of the classroom k
indergarteners \
in my class love to work with hands on materials and have many different opportunities to practice a skill be
fore it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten
 curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role play in our pr
etend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with real food i will take
their idea \
and create common core cooking lessons where we learn important math and writing concepts while cooking delic
ious healthy \
food for snack time my students will have a grounded appreciation for the work that went into making the food
and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project would expand ou
r learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce make
 our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cookbooks to be
printed and \
shared with families students will gain math and literature skills as well as a life long enjoyment for healt
hy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

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,
```

In [21]:
```
sid.polarity_scores(for_sentiment)['neg']
```

Out[21]:
```
0.01
```

In [22]:
```
vectorizer_essay_tfidf = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer_essay_tfidf.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer_essay_tfidf.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer_essay_tfidf.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer_essay_tfidf.transform(X_test['essay'].values)

print("After vectorizations: \n")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
After vectorizations:

(24000, 5000) (24000,)
(8000, 5000) (8000,)
(8000, 5000) (8000,)
```

In [23]:
```
sid = SentimentIntensityAnalyzer()

X_train_essay_neg  = []
X_train_essay_neu  = []
X_train_essay_pos  = []
X_train_essay_comp = []

X_cv_essay_neg  = []
X_cv_essay_neu  = []
X_cv_essay_pos  = []
X_cv_essay_comp = []

X_test_essay_neg  = []
X_test_essay_neu  = []
X_test_essay_pos  = []
X_test_essay_comp = []


for text in tqdm(X_train['essay']):
    X_train_essay_neg.append(sid.polarity_scores(text)['neg'])
    X_train_essay_neu.append(sid.polarity_scores(text)['neu'])
    X_train_essay_pos.append(sid.polarity_scores(text)['pos'])
    X_train_essay_comp.append(sid.polarity_scores(text)['compound'])

for text in tqdm(X_cv['essay']):
    X_cv_essay_neg.append(sid.polarity_scores(text)['neg'])
    X_cv_essay_neu.append(sid.polarity_scores(text)['neu'])
    X_cv_essay_pos.append(sid.polarity_scores(text)['pos'])
    X_cv_essay_comp.append(sid.polarity_scores(text)['compound'])

for text in tqdm(X_test['essay']):
    X_test_essay_neg.append(sid.polarity_scores(text)['neg'])
    X_test_essay_neu.append(sid.polarity_scores(text)['neu'])
    X_test_essay_pos.append(sid.polarity_scores(text)['pos'])
    X_test_essay_comp.append(sid.polarity_scores(text)['compound'])
```

```
100%|██████████| 24000/24000 [02:59<00:00,
133.73it/s]
100%|██████████| 8000/8000 [01:01<00:00,
130.40it/s]
100%|██████████| 8000/8000 [00:59<00:00,
133.85it/s]
```

In [24]:
```
X_train_essay_neg  = np.array(X_train_essay_neg)
X_train_essay_neu  = np.array(X_train_essay_neu)
X_train_essay_pos  = np.array(X_train_essay_pos)
X_train_essay_comp = np.array(X_train_essay_comp)

X_cv_essay_neg  = np.array(X_cv_essay_neg)
X_cv_essay_neu  = np.array(X_cv_essay_neu)
X_cv_essay_pos  = np.array(X_cv_essay_pos)
X_cv_essay_comp = np.array(X_cv_essay_comp)

X_test_essay_neg  = np.array(X_test_essay_neg)
X_test_essay_neu  = np.array(X_test_essay_neu)
X_test_essay_pos  = np.array(X_test_essay_pos)
X_test_essay_comp = np.array(X_test_essay_comp)
```

In [25]:
```
X_train_essay_neg  = X_train_essay_neg.reshape(-1, 1)
X_train_essay_neu  = X_train_essay_neu.reshape(-1, 1)
X_train_essay_pos  = X_train_essay_pos.reshape(-1, 1)
X_train_essay_comp = X_train_essay_comp.reshape(-1, 1)

X_cv_essay_neg  = X_cv_essay_neg.reshape(-1, 1)
X_cv_essay_neu  = X_cv_essay_neu.reshape(-1, 1)
X_cv_essay_pos  = X_cv_essay_pos.reshape(-1, 1)
X_cv_essay_comp = X_cv_essay_comp.reshape(-1, 1)

X_test_essay_neg  = X_test_essay_neg.reshape(-1, 1)
X_test_essay_neu  = X_test_essay_neu.reshape(-1, 1)
X_test_essay_pos  = X_test_essay_pos.reshape(-1, 1)
X_test_essay_comp = X_test_essay_comp.reshape(-1, 1)
```

In [26]:
```
from scipy.sparse import hstack

X_tr_tfidf = hstack((X_train_essay_tfidf, X_train_teacher_ohe, X_train_grade_ohe, X_train_state_ohe, X_train_
cat_ohe, X_train_subcat_ohe, X_train_price_norm, X_train_pre_projects_norm, X_train_essay_neg,  X_train_essay
_neu, X_train_essay_pos, X_train_essay_comp)).tocsr()
X_cr_tfidf = hstack((X_cv_essay_tfidf, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_state_ohe, X_cv_cat_ohe, X_cv_s
ubcat_ohe, X_cv_price_norm, X_cv_pre_projects_norm, X_cv_essay_neg, X_cv_essay_neu, X_cv_essay_pos, X_cv_essa
y_comp)).tocsr()
X_te_tfidf = hstack((X_test_essay_tfidf, X_test_teacher_ohe, X_test_grade_ohe, X_test_state_ohe, X_test_cat_o
he, X_test_subcat_ohe, X_test_price_norm, X_test_pre_projects_norm, X_test_essay_neg, X_test_essay_neu, X_tes
t_essay_pos, X_test_essay_comp)).tocsr()

X_tr_tfidf_w2v = hstack((X_train_essay_tfidf_w2v, X_train_teacher_ohe, X_train_grade_ohe, X_train_state_ohe,
X_train_cat_ohe, X_train_subcat_ohe, X_train_price_norm, X_train_pre_projects_norm, X_train_essay_neg,  X_tra
in_essay_neu, X_train_essay_pos, X_train_essay_comp)).tocsr()
X_cr_tfidf_w2v = hstack((X_cv_essay_tfidf_w2v, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_state_ohe, X_cv_cat_ohe
, X_cv_subcat_ohe, X_cv_price_norm, X_cv_pre_projects_norm, X_cv_essay_neg, X_cv_essay_neu, X_cv_essay_pos, X
_cv_essay_comp)).tocsr()
X_te_tfidf_w2v = hstack((X_test_essay_tfidf_w2v, X_test_teacher_ohe, X_test_grade_ohe, X_test_state_ohe, X_te
st_cat_ohe, X_test_subcat_ohe, X_test_price_norm, X_test_pre_projects_norm, X_test_essay_neg, X_test_essay_ne
u, X_test_essay_pos, X_test_essay_comp)).tocsr()

print("Final Data matrix for tfidf")
print(X_tr_tfidf.shape, y_train.shape)
print(X_cr_tfidf.shape, y_cv.shape)
print(X_te_tfidf.shape, y_test.shape)
print("="*100)
print("Final Data matrix for tfidf w2v")
print(X_tr_tfidf_w2v.shape, y_train.shape)
print(X_cr_tfidf_w2v.shape, y_cv.shape)
print(X_te_tfidf_w2v.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix for tfidf
(24000, 5101) (24000,)
(8000, 5101) (8000,)
(8000, 5101) (8000,)
==========================================================================================
Final Data matrix for tfidf w2v
(24000, 401) (24000,)
(8000, 401) (8000,)
(8000, 401) (8000,)
==========================================================================================
```

Hyperparameter tuning

SET 1 TF-IDF

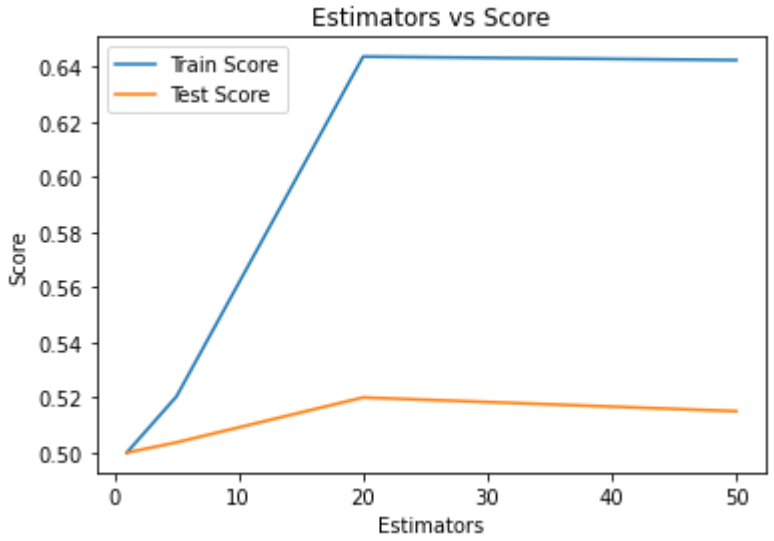
In [27]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV


depths = [1, 5, 20, 50]
min_samples_splits = [5, 10, 100, 500]

distributions = dict(max_depth=depths, min_samples_split=min_samples_splits)

clf = DecisionTreeClassifier()
clf_for_s1 = RandomizedSearchCV(clf, distributions, cv=3, scoring='roc_auc')

best_model_s1 = clf_for_s1.fit(X_tr_tfidf, y_train)

print('Best maximum depth:', best_model_s1.best_estimator_.get_params()['max_depth'])
print('Best split:', best_model_s1.best_estimator_.get_params()['min_samples_split'])
print('Best AUC score: ', clf_for_s1.best_score_)
print('='*50)
```

```
Best maximum depth: 5
Best split: 500
Best AUC score:  0.5875063719814037
==================================================
```

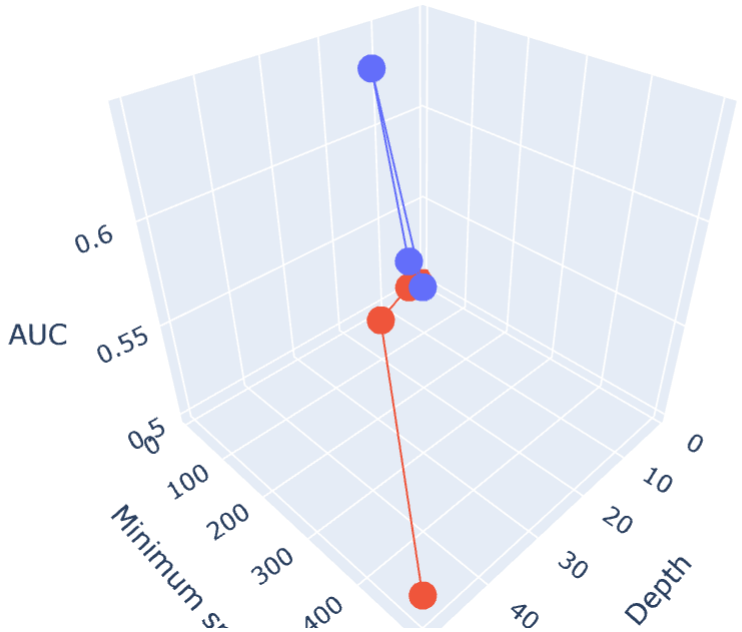
In [28]:
```
from sklearn.metrics import roc_auc_score

depths = [1, 5, 20, 50]
min_samples_splits = [5, 10, 100, 500]

train_scores_for_s1 = []
test_scores_for_s1 = []

for depth, split in zip(depths, min_samples_splits):

    clf = DecisionTreeClassifier(max_depth=depth, min_samples_split=split)
    clf.fit(X_tr_tfidf, y_train)

    train_sc = roc_auc_score(y_train, clf.predict(X_tr_tfidf))
    test_sc = roc_auc_score(y_test, clf.predict(X_te_tfidf))

    test_scores_for_s1.append(test_sc)
    train_scores_for_s1.append(train_sc)
    print('Depth = ', depth, 'Split = ', split, 'Train Score: ', train_sc, 'Test Score: ', test_sc)

plt.plot(depths , train_scores_for_s1, label='Train Score')
plt.plot(depths, test_scores_for_s1, label='Test Score')

plt.xlabel('Estimators')
plt.ylabel('Score')
plt.legend()
plt.title('Estimators vs Score')
```

```
Depth =  1 Split =  5 Train Score:  0.5 Test Score:  0.5
Depth =  5 Split =  10 Train Score:  0.5204044191157557 Test Score:  0.5037147112359817
Depth =  20 Split =  100 Train Score:  0.6437689672863709 Test Score:  0.5200292498463686
Depth =  50 Split =  500 Train Score:  0.6424933075393682 Test Score:  0.5150418068963841
```

Out[28]: `Text(0.5, 1.0, 'Estimators vs Score')`



In [29]:
```
trace1 = go.Scatter3d(x=depths,y=min_samples_splits,z=train_scores_for_s1, name = 'Train')
trace2 = go.Scatter3d(x=depths,y=min_samples_splits,z=test_scores_for_s1, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='Depth'),
        yaxis = dict(title='Minimum splits'),
        zaxis = dict(title='AUC'),))

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



SET 2 (TF-IDF W2V)

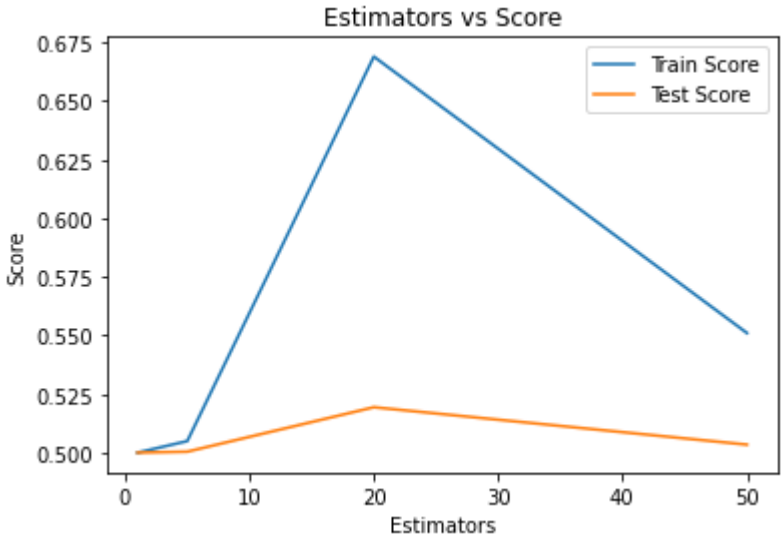
In [30]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV


depths = [1, 5, 20, 50]
min_samples_splits = [5, 10, 100, 500]

distributions = dict(max_depth=depths, min_samples_split=min_samples_splits)

clf = DecisionTreeClassifier()
clf_for_s2 = RandomizedSearchCV(clf, distributions, cv=3, scoring='roc_auc')

best_model_s2 = clf_for_s2.fit(X_tr_tfidf_w2v, y_train)

print('Best maximum depth:', best_model_s2.best_estimator_.get_params()['max_depth'])
print('Best split:', best_model_s2.best_estimator_.get_params()['min_samples_split'])
print('Best AUC score: ', clf_for_s2.best_score_)
print('='*50)
```

```
Best maximum depth: 5
Best split: 10
Best AUC score:  0.6027832838836692
==================================================
```

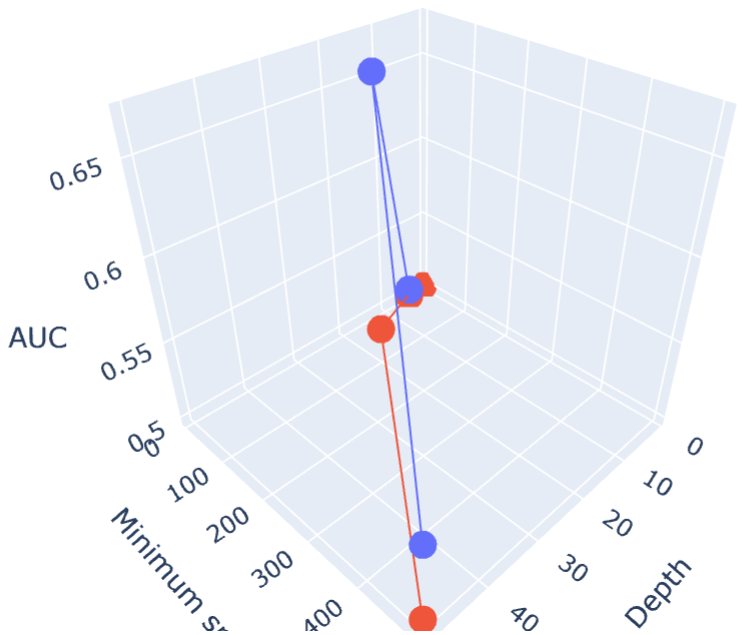
In [31]:
```
depths = [1, 5, 20, 50]
min_samples_splits = [5, 10, 100, 500]

train_scores_for_s2 = []
test_scores_for_s2 = []

for depth, split in zip(depths, min_samples_splits):

    clf = DecisionTreeClassifier(max_depth=depth, min_samples_split=split)
    clf.fit(X_tr_tfidf_w2v, y_train)

    train_sc = roc_auc_score(y_train, clf.predict(X_tr_tfidf_w2v))
    test_sc = roc_auc_score(y_test, clf.predict(X_te_tfidf_w2v))

    test_scores_for_s2.append(test_sc)
    train_scores_for_s2.append(train_sc)
    print('Depth = ', depth, 'Split = ', split, 'Train Score: ', train_sc, 'Test Score: ', test_sc)

plt.plot(depths , train_scores_for_s2, label='Train Score')
plt.plot(depths, test_scores_for_s2, label='Test Score')

plt.xlabel('Estimators')
plt.ylabel('Score')
plt.legend()
plt.title('Estimators vs Score')
```

```
Depth =  1 Split =  5 Train Score:  0.5 Test Score:  0.5
Depth =  5 Split =  10 Train Score:  0.5049273781634438 Test Score:  0.5004104460505306
Depth =  20 Split =  100 Train Score:  0.6689521167532702 Test Score:  0.5193765792576618
Depth =  50 Split =  500 Train Score:  0.5510238428924987 Test Score:  0.5034134647723336
```

Out[31]: Text(0.5, 1.0, 'Estimators vs Score')



In [32]:
```
trace1 = go.Scatter3d(x=depths,y=min_samples_splits,z=train_scores_for_s2, name = 'Train')
trace2 = go.Scatter3d(x=depths,y=min_samples_splits,z=test_scores_for_s2, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='Depth'),
        yaxis = dict(title='Minimum splits'),
        zaxis = dict(title='AUC'),))

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



1. Decision Tree

1.1 Loading Data

In [33]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv')
```

SET 1 TF-IDF

In [34]:
```
clf1 = DecisionTreeClassifier(max_depth=best_model_s1.best_estimator_.get_params()['max_depth'], min_samples_
split=best_model_s1.best_estimator_.get_params()['min_samples_split'])
clf1.fit(X_tr_tfidf, y_train)
```

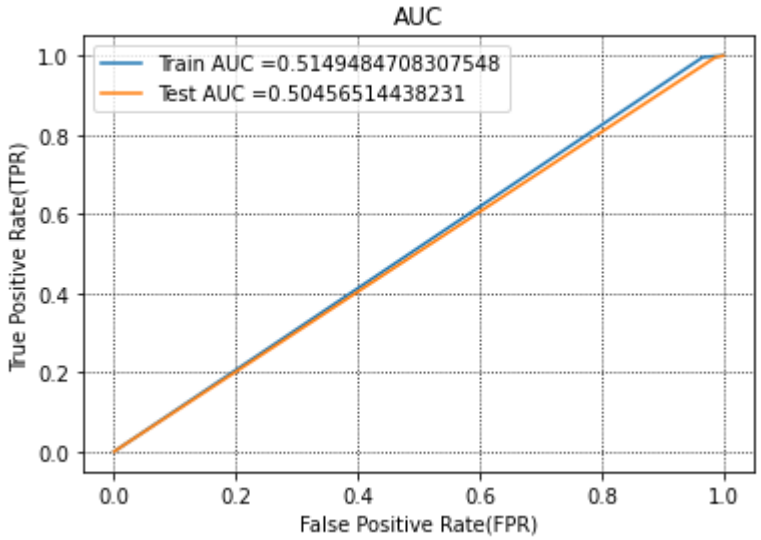
Out[34]:
```
DecisionTreeClassifier(max_depth=5, min_samples_split=500)
```

In [35]:
```
y_testing_pred1 = clf1.predict(X_te_tfidf)
best_model_accuracy = roc_auc_score(y_testing_pred1, y_test)
print('The AUC score for the best model: ', best_model_accuracy)
```

```
The AUC score for the best model:  0.5973783300265145
```

In [36]:
```
y_training_pred1 = clf1.predict(X_tr_tfidf)
```

In [37]:
```
y_train_pred = clf1.predict(X_tr_tfidf)
y_test_pred = clf1.predict(X_te_tfidf)

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

auc_set1_train = auc(train_fpr, train_tpr)
auc_set1_test = auc(test_fpr, test_tpr)

ax = plt.subplot()
ax.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
ax.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid(b=True, which='major', color='k', linestyle=':')
ax.set_facecolor("white")
plt.show()
```



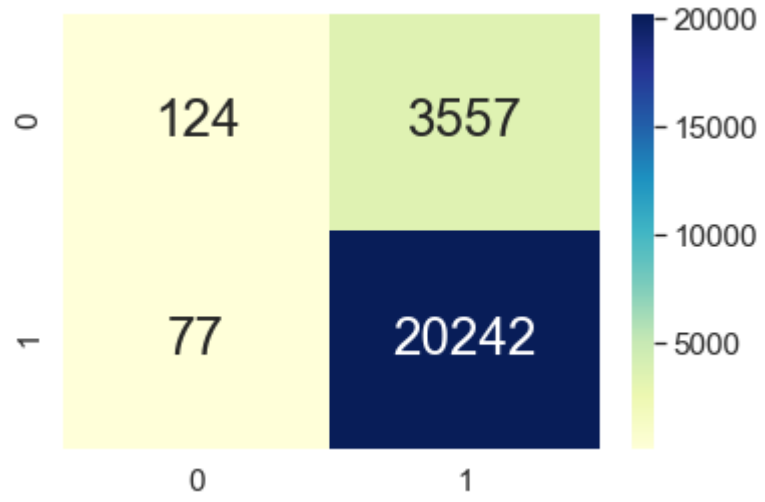
In [38]:
```
def predict(proba, threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)

    return predictions
```

Train Confusion Matrix

In [39]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))

conf_matr_df_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds,train_fpr, tr
ain_tpr)), range(2),range(2))

sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_train, annot=True,annot_kws={"size": 26}, fmt='g',cmap="YlGnBu")
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.033558841343382044 for threshold 1
[[  124  3557]
 [   77 20242]]
the maximum value of tpr*(1-fpr) 0.033558841343382044 for threshold 1
```

Out[39]: <AxesSubplot:>

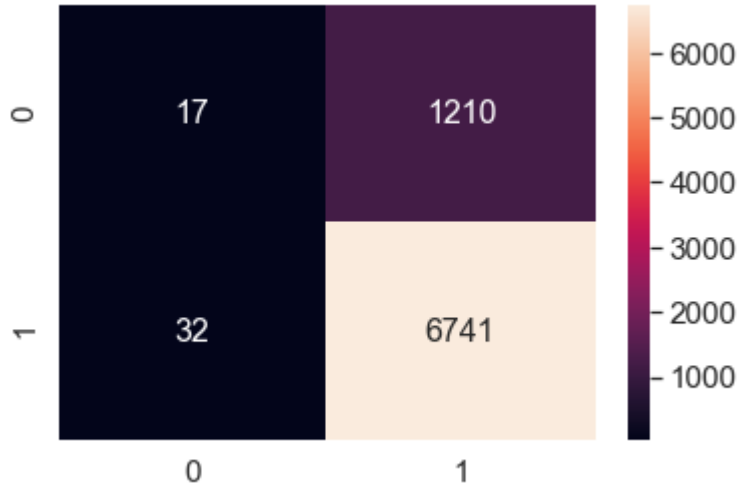


Test Confusion Matrix

In [40]:
```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))

conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_
tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
====================================================================================================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.013789471138278452 for threshold 1
[[  17 1210]
 [  32 6741]]
the maximum value of tpr*(1-fpr) 0.013789471138278452 for threshold 1
```

Out[40]: <AxesSubplot:>



Retrieving False Positives

In [41]:
```
FP_essay_train_set1=[]
FP_price_train_set1=[]
FP_previous_posted_train_set1=[]

FP_essay_train_set2=[]
FP_price_train_set2=[]
FP_previous_posted_train_set2=[]

FP_essay_train_set3=[]
FP_price_train_set3=[]
FP_previous_posted_train_set3=[]

FP_essay_test_set1=[]
FP_price_test_set1=[]
FP_previous_posted_test_set1=[]

FP_essay_test_set2=[]
FP_price_test_set2=[]
FP_previous_posted_test_set2=[]

FP_essay_test_set3=[]
FP_price_test_set3=[]
FP_previous_posted_test_set3=[]


def retrievingFalsePositives(setNumber, part):
    if(setNumber==1 and part=="train"):
        FP_train_indexes_set1=[]
        for i in range(len(y_train)):
            if((y_train[i]==0) and (y_training_pred1[i]==1) ):
                FP_train_indexes_set1.append(i)

        for i in FP_train_indexes_set1:
            FP_essay_train_set1.append(X_train['essay'].values[i])
            FP_price_train_set1.append(X_train['price_x'].values[i])
            FP_previous_posted_train_set1.append(X_train['teacher_number_of_previously_posted_projects'].valu
es[i])

    if(setNumber==2 and part=="train"):
        FP_train_indexes_set2=[]
        for i in range(len(y_train)):
            if(y_train[i]==0 and y_training_pred2[i]==1 ):
```

```
                FP_train_indexes_set2.append(i)

        for i in FP_train_indexes_set2:
            FP_essay_train_set2.append(X_train['essay'].values[i])
            FP_price_train_set2.append(X_train['price'].values[i])
            FP_previous_posted_train_set2.append(X_train['teacher_number_of_previously_posted_projects'].valu
es[i])

    if(setNumber==3 and part=="train"):
        FP_train_indexes_set3=[]
        for i in range(len(y_train)):
            if(y_train[i]==0 and y_training_pred3[i]==1 ):
                FP_train_indexes_set3.append(i)

        for i in FP_train_indexes_set3:
            FP_essay_train_set3.append(X_train['essay'].values[i])
            FP_price_train_set3.append(X_train['price'].values[i])
            FP_previous_posted_train_set3.append(x_train['teacher_number_of_previously_posted_projects'].valu
es[i])

    if(setNumber==1 and part=="test"):
        FP_test_indexes_set1=[]
        for i in range(len(y_test)):
            if(y_test[i]==0 and y_testing_pred1[i]==1 ):
                FP_test_indexes_set1.append(i)

        for i in FP_test_indexes_set1:
            FP_essay_test_set1.append(X_test['essay'].values[i])
            FP_price_test_set1.append(X_test['price'].values[i])
            FP_previous_posted_test_set1.append(X_test['teacher_number_of_previously_posted_projects'].values
[i])

    if(setNumber==2 and part=="test"):
        FP_test_indexes_set2=[]
        for i in range(len(y_test)):
            if(y_test[i]==0 and y_testing_pred2[i]==1 ):
                FP_test_indexes_set2.append(i)
        for i in FP_test_indexes_set2:
            FP_essay_test_set2.append(X_test['essay'].values[i])
            FP_price_test_set2.append(X_test['price'].values[i])
            FP_previous_posted_test_set2.append(X_test['teacher_number_of_previously_posted_projects'].values
[i])
```

```
    if(setNumber==3 and part=="test"):
        FP_test_indexes_set3=[]
        for i in range(len(y_test)):
            if(y_test[i]==0 and y_testing_pred3[i]==1 ):
                FP_test_indexes_set3.append(i)
        for i in FP_test_indexes_set3:
            FP_essay_test_set3.append(X_test['essay'].values[i])
            FP_price_test_set3.append(X_test['price'].values[i])
            FP_previous_posted_test_set3.append(X_test['teacher_number_of_previously_posted_projects'].values
[i])

# source: https://www.kaggle.com/nikhilparmar9/decision-tree-donorschoose-dataset#Drawing-the-WordCloud-of-th
e-Words-in-Essay-Text
# Made necessary modifications
```

WordCloud

In [42]:
```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
from PIL import Image # for masking i.e print word in the pattern we want
import pandas as pd

def printWordCloud(FP_list):
    comment_words = ''
    stopwords = set(STOPWORDS)

    for val in FP_list:

        val = str(val)

        tokens = val.split()

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

        for words in tokens:
            comment_words = comment_words + words + ' '


    wordcloud = WordCloud(width = 500, height = 500,
                    background_color ='white',
                    stopwords = stopwords,
                    min_font_size = 10).generate(comment_words)

    # plot the WordCloud image
    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad = 0)
```

In [43]:
```
def printBoxPlot(FP_list):
    plt.boxplot(FP_list)
    plt.title('Box Plot for PRICE in False Positives')
    plt.ylabel('Price')
    plt.grid()
    plt.show()
```

In [44]:
```
def printPDF(FP_list):
    plt.figure(figsize=(10,3))
    sns.distplot(FP_list)
    plt.title('PDF for Teacher number who previously posted projects in False Positives')
    plt.xlabel('Teacher number who previously posted projects')
    plt.legend()
    plt.show()
```

SET 2 TF-IDF W2V

In [45]:
```
clf2 = DecisionTreeClassifier(max_depth=best_model_s2.best_estimator_.get_params()['max_depth'], min_samples_
split=best_model_s2.best_estimator_.get_params()['min_samples_split'])
clf2.fit(X_tr_tfidf_w2v, y_train)
```

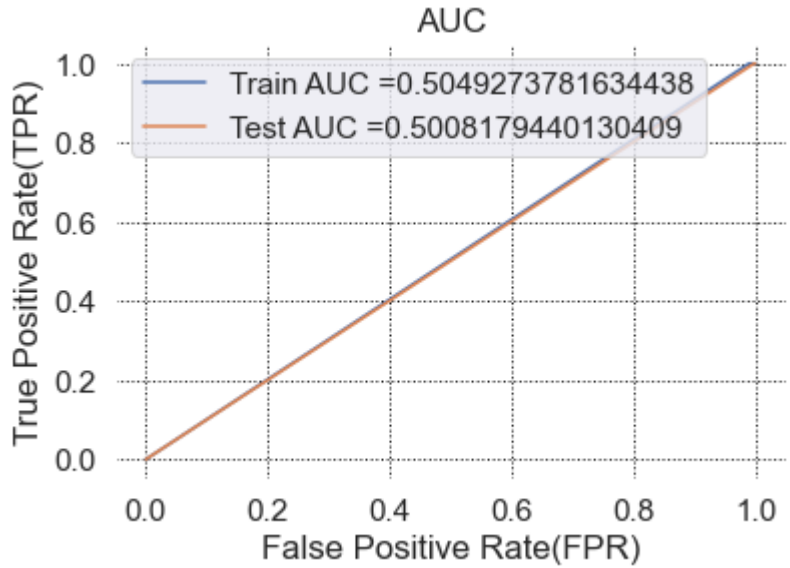
Out[45]:
```
DecisionTreeClassifier(max_depth=5, min_samples_split=10)
```

In [46]:
```
y_testing_pred2 = clf2.predict(X_te_tfidf_w2v)
best_model_accuracy2 = roc_auc_score(y_testing_pred2, y_test)
print('The AUC score for the best model: ', best_model_accuracy2)
```

```
The AUC score for the best model:  0.5567522437904404
```

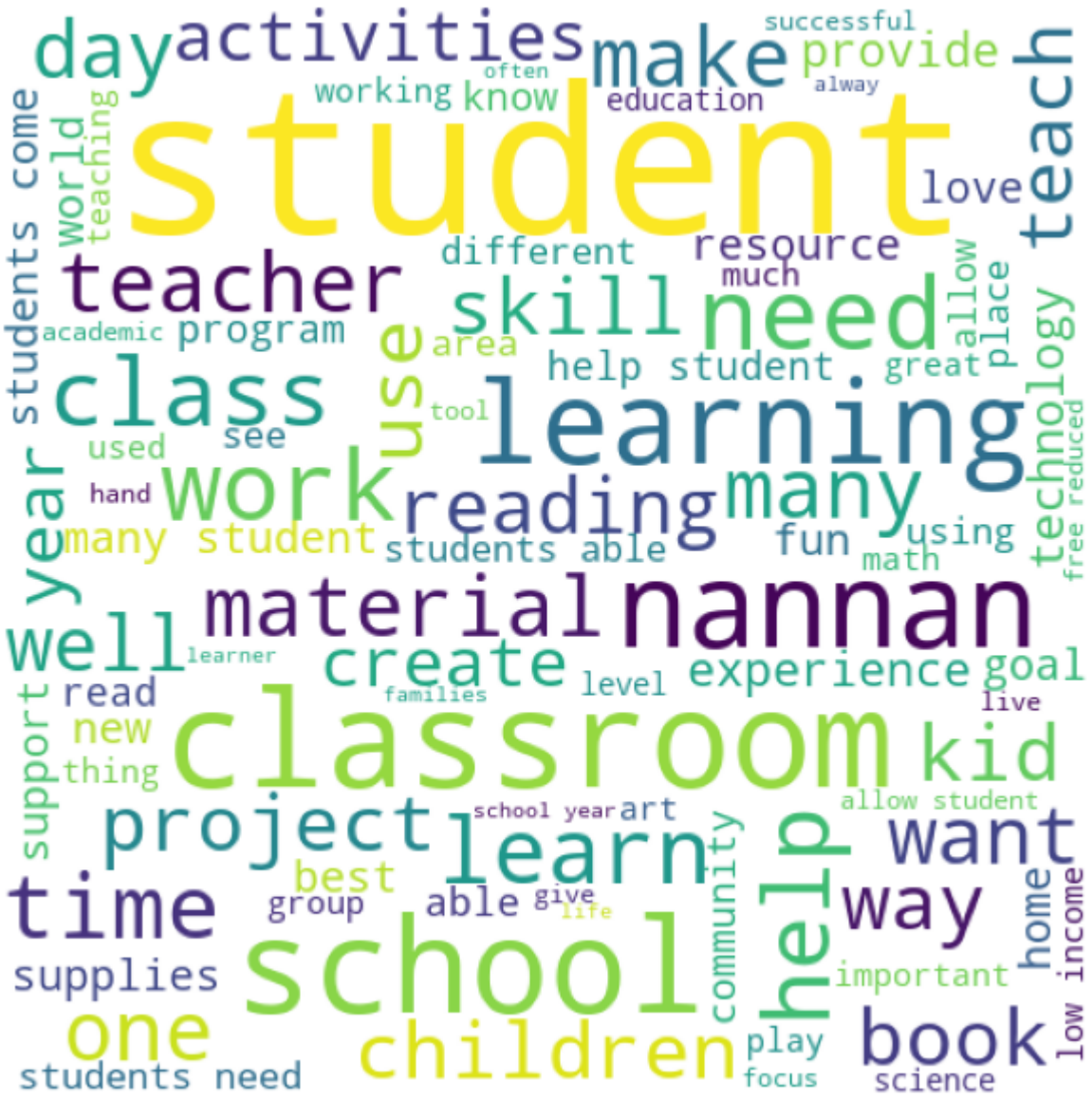
In [47]:
```
y_training_pred2 = clf2.predict(X_tr_tfidf_w2v)
```

In [48]:
```
y_train_pred = clf2.predict(X_tr_tfidf_w2v)
y_test_pred = clf2.predict(X_te_tfidf_w2v)

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

auc_set1_train = auc(train_fpr, train_tpr)
auc_set1_test = auc(test_fpr, test_tpr)

ax = plt.subplot()
ax.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
ax.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid(b=True, which='major', color='k', linestyle=':')
ax.set_facecolor("white")
plt.show()
```

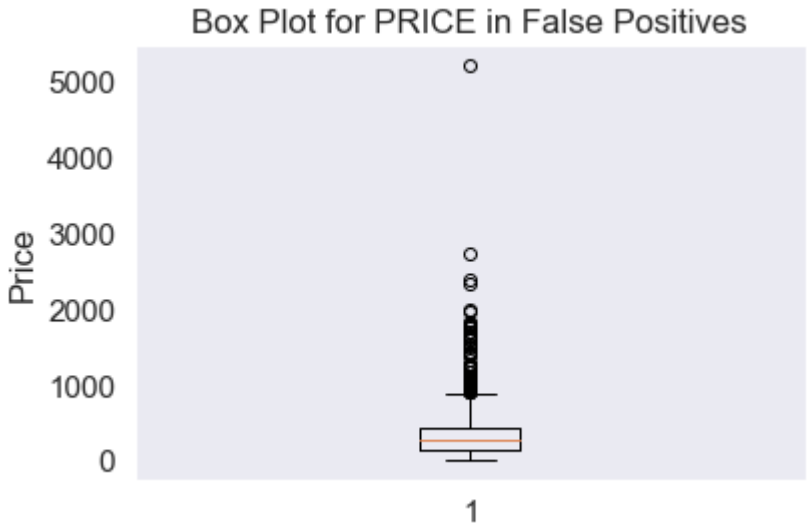


In [49]:
```
retrievingFalsePositives(2, "test")
```

In [50]:
```
printWordCloud(FP_essay_test_set2)
```



In [51]:
```
printBoxPlot(FP_price_test_set2)
```



In [52]:
```
printPDF(FP_previous_posted_test_set2)
```

```
No handles with labels found to put in legend.
```

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 'featureimportances` (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (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>

In [53]:
```
def Importance(model, X, k):
    return X[:,model.feature_importances_.argsort()[::-1][:k]]
```

In [54]:
```
another_model = DecisionTreeClassifier(class_weight='balanced')
another_model.fit(X_tr_tfidf, y_train)
```

Out[54]:
```
DecisionTreeClassifier(class_weight='balanced')
```

In [55]:
```
nonZeroFeatures=0

for i in range (len(another_model.feature_importances_)):
    if (another_model.feature_importances_[i]>0):
        nonZeroFeatures = nonZeroFeatures+1
```

In [56]:
```
X_train_imp_features = Importance(another_model, X_tr_tfidf, nonZeroFeatures)
X_test_imp_features = Importance(another_model, X_te_tfidf, nonZeroFeatures)
```

Hyperparameter tuning

In [57]:
```
parameters = {'max_depth':[1, 5, 10, 50],'min_samples_split':[5, 10, 100, 500]}

another_best_model = RandomizedSearchCV(another_model, parameters, cv= 3, scoring='roc_auc', verbose=1, retur
n_train_score=True, n_jobs=-1)

another_best_model.fit(X_train_imp_features, y_train)

train_auc = another_best_model.cv_results_['mean_train_score']
train_auc_std = another_best_model.cv_results_['std_train_score']
cv_auc = another_best_model.cv_results_['mean_test_score']
cv_auc_std = another_best_model.cv_results_['std_test_score']
bestMaxDepth_3 = another_best_model.best_params_['max_depth']
bestMinSampleSplit_3 = another_best_model.best_params_['min_samples_split']
bestScore_3 = another_best_model.best_score_
print("BEST MAX DEPTH: ", another_best_model.best_params_['max_depth'],
      " BEST AUC: ", another_best_model.best_score_,
      "BEST SPLIT: ", another_best_model.best_params_['min_samples_split']) #clf.best_estimator_.alpha
```

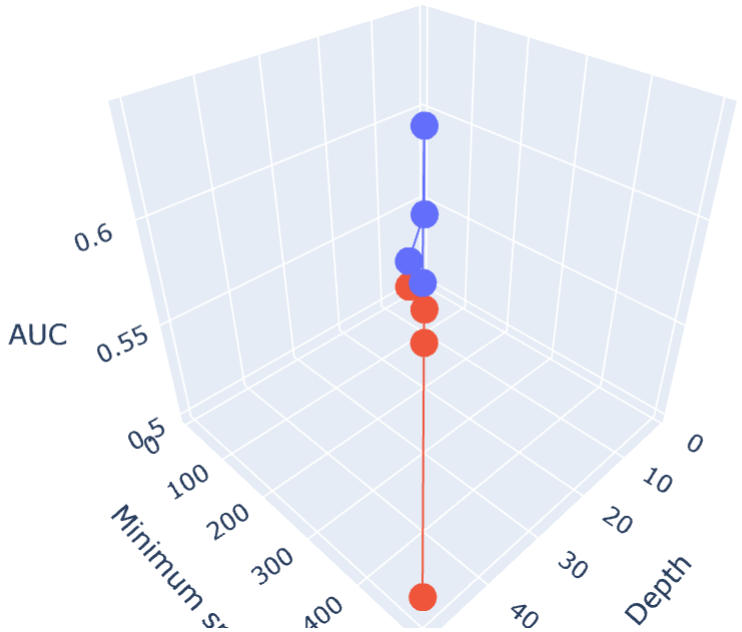
```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
BEST MAX DEPTH:  5  BEST AUC:  0.6071517687344876 BEST SPLIT:  500
```

In [60]:
```
from sklearn.metrics import roc_auc_score

depths = [1, 5, 10,  20, 50]
min_samples_splits = [5, 10, 100, 200, 500]

train_scores_for_s3 = []
test_scores_for_s3 = []

for depth, split in zip(depths, min_samples_splits):

    clf = DecisionTreeClassifier(max_depth=depth, min_samples_split=split)
    clf.fit(X_tr_tfidf, y_train)

    train_sc = roc_auc_score(y_train, clf.predict(X_tr_tfidf))
    test_sc = roc_auc_score(y_test, clf.predict(X_te_tfidf))

    test_scores_for_s3.append(test_sc)
    train_scores_for_s3.append(train_sc)
    print('Depth = ', depth, 'Split = ', split, 'Train Score: ', train_sc, 'Test Score: ', test_sc)

plt.plot(depths , train_scores_for_s3, label='Train Score')
plt.plot(depths, test_scores_for_s3, label='Test Score')

plt.xlabel('Estimators')
plt.ylabel('Score')
plt.legend()
plt.title('Estimators vs Score')
```

```
Depth =  1 Split =  5 Train Score:  0.5 Test Score:  0.5
Depth =  5 Split =  10 Train Score:  0.5204044191157557 Test Score:  0.5041960317291282
Depth =  10 Split =  100 Train Score:  0.5654280004105664 Test Score:  0.5106775536549012
Depth =  20 Split =  200 Train Score:  0.6282358123865663 Test Score:  0.5228345060105498
Depth =  50 Split =  500 Train Score:  0.6427649728477083 Test Score:  0.5140023952914341
```

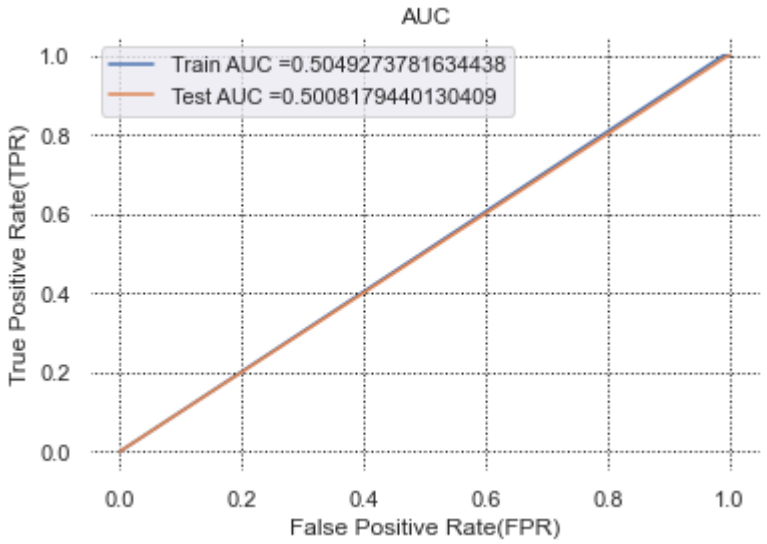
Out[60]: `Text(0.5, 1.0, 'Estimators vs Score')`

In [63]:
```
trace1 = go.Scatter3d(x=depths,y=min_samples_splits,z=train_scores_for_s3, name = 'Train')
trace2 = go.Scatter3d(x=depths,y=min_samples_splits,z=test_scores_for_s3, name = 'Cross validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(
        xaxis = dict(title='Depth'),
        yaxis = dict(title='Minimum splits'),
        zaxis = dict(title='AUC'),))

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



ROC curve

In [66]:
```
y_training_pred3 = another_best_model.predict(X_train_imp_features)
y_testing_pred3 = another_best_model.predict(X_test_imp_features)

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

auc_set1_train = auc(train_fpr, train_tpr)
auc_set1_test = auc(test_fpr, test_tpr)

ax = plt.subplot()
ax.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
ax.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid(b=True, which='major', color='k', linestyle=':')
ax.set_facecolor("white")
plt.show()
```

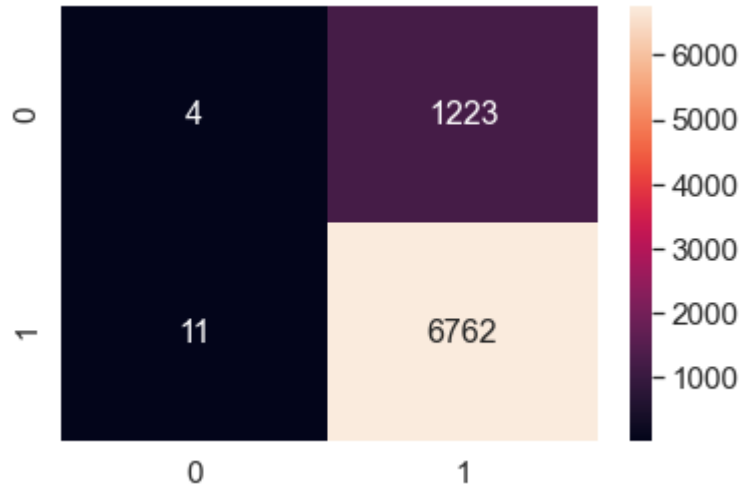


Confusion matrix

In [67]:
```
print("="*100)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))

conf_matr_df_test = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_
tpr)), range(2),range(2))
sns.set(font_scale=1.4)#for label size
sns.heatmap(conf_matr_df_test, annot=True, annot_kws={"size": 16}, fmt='g')
```

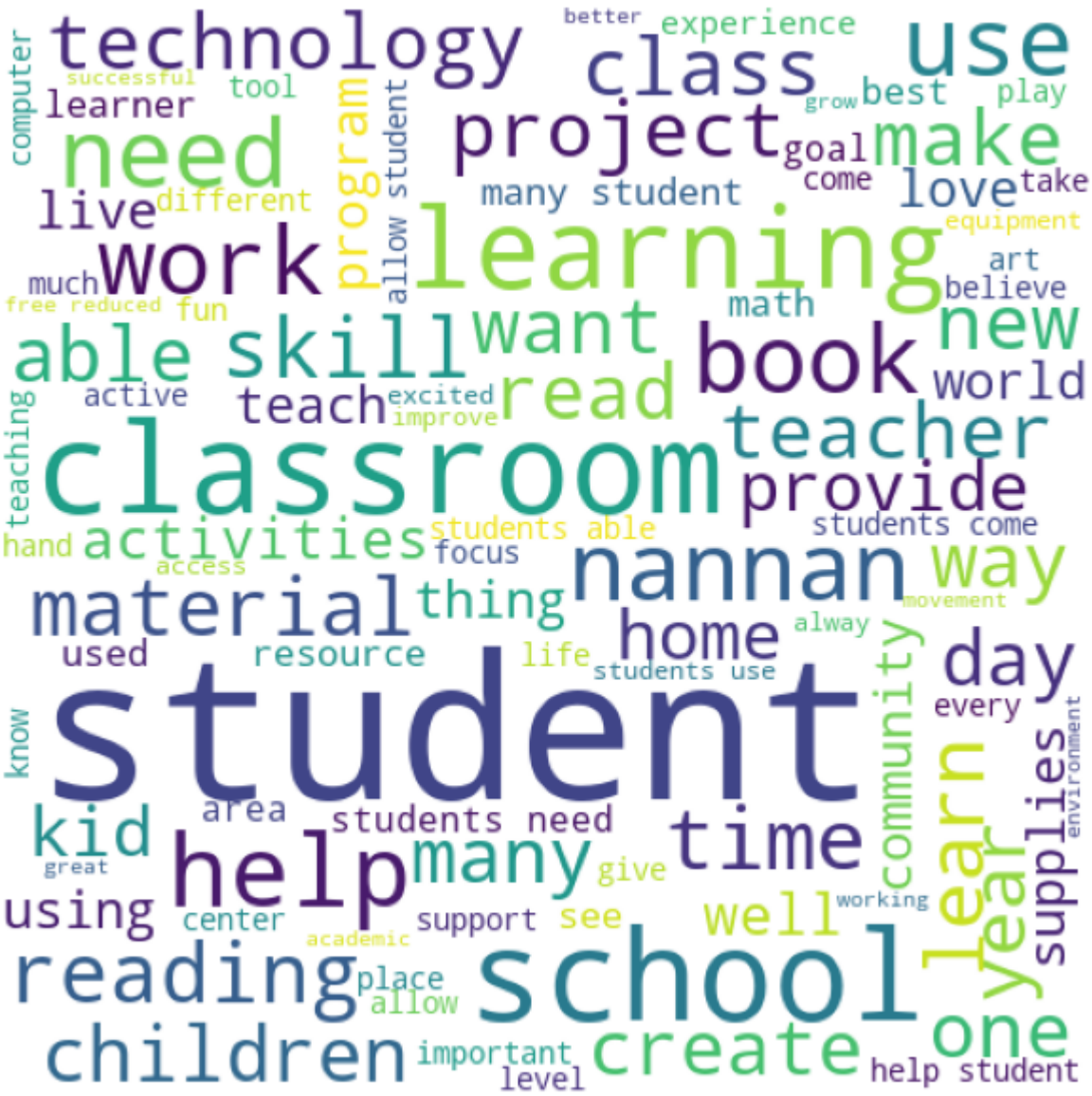
```
====================================================================================================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.003254689174656903 for threshold 1
[[   4 1223]
 [  11 6762]]
the maximum value of tpr*(1-fpr) 0.003254689174656903 for threshold 1
```

Out[67]: <AxesSubplot:>

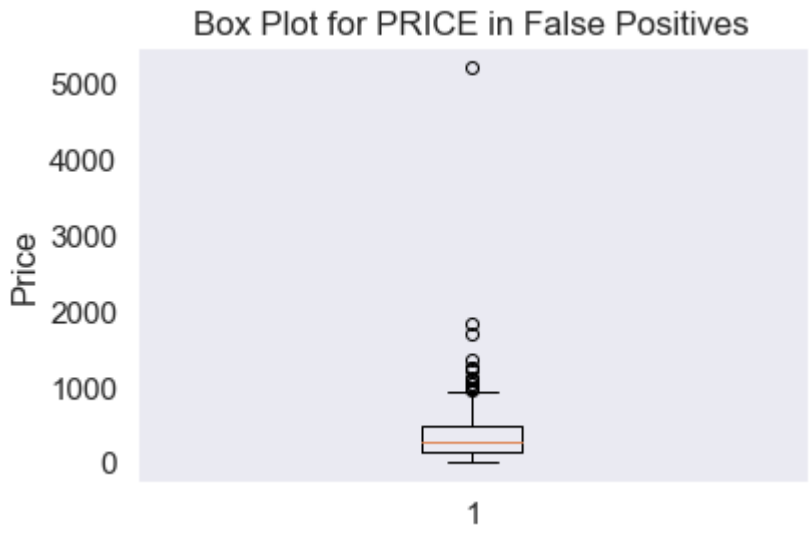


In [68]:
```
retrievingFalsePositives(3,"test")
```

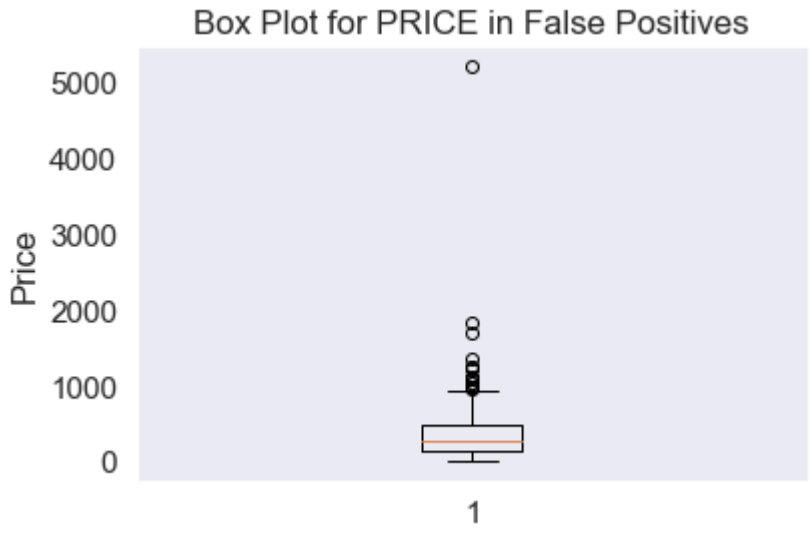
In [69]:
```
printWordCloud(FP_essay_test_set3)
```



In [70]:
```
printBoxPlot(FP_price_test_set3)
```



In [71]:
```
printBoxPlot(FP_price_test_set3)
```



In [72]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Min. Sample Split","Max Dept", "Train AUC", "Test AUC"]

x.add_row(["TF-IDF", "Decision Trees", best_model_s1.best_estimator_.get_params()['min_samples_split'], best_
model_s1.best_estimator_.get_params()['max_depth'], max(train_scores_for_s1), max(test_scores_for_s1)])
x.add_row(["TF-IDF W2V", "Decision Trees", best_model_s2.best_estimator_.get_params()['min_samples_split'], b
est_model_s1.best_estimator_.get_params()['max_depth'], max(train_scores_for_s2), max(test_scores_for_s2)])
x.add_row(["TF-IDF (Important Feature)", "Decision Trees",another_best_model.best_params_['min_samples_split'
], another_best_model.best_params_['max_depth'], max(train_scores_for_s3),max(test_scores_for_s3)])
```

In [73]:
```
print(x)
```

```
+---------------------------+---------------+------------------+----------+--------------------+---------
-----------+
|         Vectorizer        |     Model     | Min. Sample Split | Max Dept |     Train AUC      |      Tes
t AUC      |
+---------------------------+---------------+------------------+----------+--------------------+---------
-----------+
|           TF-IDF          | Decision Trees |        500        |    5     | 0.6437689672863709 | 0.520029
2498463686 |
|         TF-IDF W2V        | Decision Trees |         10        |    5     | 0.6689521167532702 | 0.519376
5792576618 |
| TF-IDF (Important Feature) | Decision Trees |        500        |    5     | 0.6427649728477083 | 0.522834
5060105498 |
+---------------------------+---------------+------------------+----------+--------------------+---------
-----------+
```