Donors Choose - Decision Tree

Objective : Predict whether teachers' project proposals are accepted

Importing packages

In [1]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

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

Reading the data

In [2]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

Project data

In [3]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("Attributes :", project_data.columns.values)
project_data.head(2)
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
Attributes : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

Out[3]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	

Handling Missing Value in "Teacher prefix" column

In [4]:

```
a = project_data['teacher_prefix'].mode().values
```

In [5]:

```
#Replacin nan with the most frequently occured value in that column
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(a[0])
```

Total number of null values in each column

In [6]:

```
#Total number of null values in each column
project_data.isnull().sum(axis = 0)
```

Out[6]:

```
Unnamed: 0                                          0
id                                                  0
teacher_id                                          0
teacher_prefix                                      0
school_state                                        0
project_submitted_datetime                          0
project_grade_category                              0
project_subject_categories                          0
project_subject_subcategories                       0
project_title                                       0
project_essay_1                                     0
project_essay_2                                     0
project_essay_3                                105490
project_essay_4                                105490
project_resource_summary                            0
teacher_number_of_previously_posted_projects        0
project_is_approved                                 0
dtype: int64
```

Resource data

In [7]:

```
print("Number of data points in train data", resource_data.shape)
print('-'*50)
print("Attributes: ", resource_data.columns.values)
resource_data.head(2)
```

```
Number of data points in train data (1541272, 4)
--------------------------------------------------
Attributes:  ['id' 'description' 'quantity' 'price']
```

Out[7]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

Replacing date-time with date

In [8]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]


project_data.head(2)
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	00:
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	00:

In [9]:

```
# https://matplotlib.org/gallery/pie_and_polar_charts/pie_and_donut_labels.html#sphx-glr-ga

y_value_counts = project_data['project_is_approved'].value_counts()
print("Number of projects thar are approved for funding ", y_value_counts[1], ", (", (y_val
print("Number of projects thar are not approved for funding ", y_value_counts[0], ", (", (y
```

```
Number of projects thar are approved for funding  92706 , ( 84.8583040421792
7 %)
Number of projects thar are not approved for funding  16542 , ( 15.141695957
820739 %)
```

NOTE: This is an imbalance dataset that containes 85% approved project's data and 15% not approved project's data

Preprocessing Categorical Data

Project Subject Categories

In [10]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "
        if 'The' in j.split(): # this will split each of the catogory based on space "Math
            j=j.replace('The','') # if we have the words "The" we are going to replace it w
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
        temp = temp.replace('-','_') # we are replacing - & _
        temp = temp.lower()
    cat_list.append(temp.strip())

project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

In [11]:

```
print(sorted_cat_dict)
```

```
{'warmth': 1388, 'care_hunger': 1388, 'history_civics': 5914, 'music_arts':
10293, 'appliedlearning': 12135, 'specialneeds': 13642, 'health_sports': 142
23, 'math_science': 41421, 'literacy_language': 52239}
```

Project Subject Sub-Categories

In [12]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "
        if 'The' in j.split(): # this will split each of the catogory based on space "Math
            j=j.replace('The','') # if we have the words "The" we are going to replace it w
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
        temp = temp.replace('-','_') # we are replacing - & _
        temp = temp.lower()
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

In [13]:

```
print(sorted_sub_cat_dict)
```

```
{'economics': 269, 'communityservice': 441, 'financialliteracy': 568, 'paren
tinvolvement': 677, 'extracurricular': 810, 'civics_government': 815, 'forei
gnlanguages': 890, 'nutritioneducation': 1355, 'warmth': 1388, 'care_hunge
r': 1388, 'socialsciences': 1920, 'performingarts': 1961, 'charactereducatio
n': 2065, 'teamsports': 2192, 'other': 2372, 'college_careerprep': 2568, 'mu
sic': 3145, 'history_geography': 3171, 'health_lifescience': 4235, 'earlydev
elopment': 4254, 'esl': 4367, 'gym_fitness': 4509, 'environmentalscience': 5
591, 'visualarts': 6278, 'health_wellness': 10234, 'appliedsciences': 10816,
'specialneeds': 13642, 'literature_writing': 22179, 'mathematics': 28074, 'l
iteracy': 33700}
```

Teacher Prefix

In [14]:

```
prefix = list(project_data['teacher_prefix'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

prefix_list = []
for i in prefix:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('.','')
        temp = temp.lower()
    prefix_list.append(temp.strip())

project_data['prefix_teacher'] = prefix_list
project_data.drop(['teacher_prefix'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['prefix_teacher'].values:
    my_counter.update(word.split())

prefix_dict = dict(my_counter)
sorted_prefix_dict = dict(sorted(prefix_dict.items(), key=lambda kv: kv[1]))
```

In [15]:

```
print(sorted_prefix_dict)
```

```
{'dr': 13, 'teacher': 2360, 'mr': 10648, 'ms': 38955, 'mrs': 57272}
```

Project Grade categories

In [16]:

```
grades = list(project_data["project_grade_category"].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

grades_list = []
for i in grades:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "
        j = j.replace(' ','_') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('-','_')
        temp = temp.lower()
    grades_list.append(temp.strip())

project_data['project_grade'] = grades_list
project_data.drop(["project_grade_category"], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in project_data['project_grade'].values:
    my_counter.update(word.split())

grade_dict = dict(my_counter)
sorted_grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))
```

In [17]:

```
print(sorted_grade_dict)
```

```
{'grades_9_12': 10963, 'grades_6_8': 16923, 'grades_3_5': 37137, 'grades_pre
k_2': 44225}
```

Preprocessing Text Data

Project Essay

In [18]:

```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [20]:

```
# https://stackoverflow.com/a/47091490/4084039
import re


def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [23]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"]


from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('nannan', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = sent.lower()
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split(" ") if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████
██| 109248/109248 [02:11<00:00, 828.61it/s]
```

In [24]:

```
# placing the preprocessed essay into the dataframe
project_data['clean_essays'] = preprocessed_essays
project_data.drop(['essay'], axis=1, inplace=True)
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [25]:

```
#Printing random cleaned essay
project_data['clean_essays'].values[23]
```

Out[25]:

```
'everyday students interact technology enhance learning experience days tech
nology not available classroom ipad minis would technology count students di
verse group whose learning needs range need individualized attention master
concepts require additional extension maintain interest love learn explore s
tudents experiences technology range students whose experience school fluent
use devices home students enthusiastic learning particularly excited learnin
g together lot collaborative learning order experience like work team since
team work wave future students love learn work teams use ipad minis protecti
ve cases dig research use qr codes apps learning not benefits technology aff
ords also learn important social skills sharing working together developing
leaders learned work hard collaborate groups addition two ipads classroom st
udents opportunity use technology frequently enhance learning especially tru
e little no technology home compete 21st century need level playing field ev
eryone students awesome deserve technology available learn compete future'
```

Project title

In [26]:

```
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = sent.lower()
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████████
| 109248/109248 [00:06<00:00, 18181.59it/s]
```

In [27]:

```
# placing the preprocessed essay into the dataframe
project_data['clean_titles'] = preprocessed_titles
project_data.drop(['project_title'], axis=1, inplace=True)
```

In [28]:

```
#Printing random cleaned title
project_data['clean_titles'].values[12]
```

Out[28]:

```
'robots taking 2nd grade'
```

Merging Price and quantity data to Project data (left joining price data)

In [29]:

```
# reference : https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexe
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index
price_data.head(2)
```

Out[29]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [30]:

```
# join two dataframes(project_data and price_data) in python
# reference : https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.m
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

Splitting Data and Starifying the sampling

In [31]:

```
y = project_data['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
X = project_data

print(X.shape)
print(y.shape)
```

```
(109248, 18)
(109248,)
```

In [32]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_spli
#https://stackoverflow.com/questions/34842405/parameter-stratify-from-method-train-test-spl
from sklearn.model_selection import train_test_split

# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Flase)#
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify = y) # t
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify

print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(49041, 18) (49041,)
(24155, 18) (24155,)
(36052, 18) (36052,)
```

Vectorizing Categorical Data

Clean Categories

In [184]:

```
# we use count vectorizer to convert the values into one hot encoded features

# Vectorizing "clean_categories"
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_sbj = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False,
vectorizer_sbj.fit(X_train['clean_categories'].values)

X_train_categories_one_hot = vectorizer_sbj.transform(X_train['clean_categories'].values)
X_cv_categories_one_hot = vectorizer_sbj.transform(X_cv['clean_categories'].values)
X_test_categories_one_hot = vectorizer_sbj.transform(X_test['clean_categories'].values)

print("After verctorizing")
print(X_train_categories_one_hot.shape, y_train.shape)
print(X_cv_categories_one_hot.shape, y_cv.shape)
print(X_test_categories_one_hot.shape, y_test.shape)

print(vectorizer_sbj.get_feature_names())
#printing first five elemets of the array of vectors
print(X_train_categories_one_hot[0:3])
```

```
After verctorizing
(49041, 9) (49041,)
(24155, 9) (24155,)
(36052, 9) (36052,)
['warmth', 'care_hunger', 'history_civics', 'music_arts', 'appliedlearning',
'specialneeds', 'health_sports', 'math_science', 'literacy_language']
  (0, 7)        1
  (1, 7)        1
  (1, 8)        1
  (2, 8)        1
```

Clean sub Categories

In [183]:

```
# Vectorizing "clean_subcategories"
vectorizer_sub_sbj = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase
vectorizer_sub_sbj.fit(X_train['clean_subcategories'].values)

X_train_sub_categories_one_hot = vectorizer_sub_sbj.transform(X_train['clean_subcategories'
X_cv_sub_categories_one_hot = vectorizer_sub_sbj.transform(X_cv['clean_subcategories'].valu
X_test_sub_categories_one_hot = vectorizer_sub_sbj.transform(X_test['clean_subcategories'].

print("After verctorizing")
print(X_train_sub_categories_one_hot.shape, y_train.shape)
print(X_cv_sub_categories_one_hot.shape, y_cv.shape)
print(X_test_sub_categories_one_hot.shape, y_test.shape)

print(vectorizer_sub_sbj.get_feature_names())

print(X_train_sub_categories_one_hot[0:3])
```

```
After verctorizing
(49041, 30) (49041,)
(24155, 30) (24155,)
(36052, 30) (36052,)
['economics', 'communityservice', 'financialliteracy', 'parentinvolvement',
'extracurricular', 'civics_government', 'foreignlanguages', 'nutritioneducat
ion', 'warmth', 'care_hunger', 'socialsciences', 'performingarts', 'characte
reducation', 'teamsports', 'other', 'college_careerprep', 'music', 'history_
geography', 'health_lifescience', 'earlydevelopment', 'esl', 'gym_fitness',
'environmentalscience', 'visualarts', 'health_wellness', 'appliedsciences',
'specialneeds', 'literature_writing', 'mathematics', 'literacy']
  (0, 28)       1
  (1, 28)       1
  (1, 29)       1
  (2, 29)       1
```

Teacher Prefix

In [185]:

```
# Vectorizing "teacher_prefix"
vectorizer_teacher = CountVectorizer(vocabulary=list(sorted_prefix_dict.keys()), lowercase=
vectorizer_teacher.fit(X_train['prefix_teacher'].values)

X_train_prefix_one_hot = vectorizer_teacher.transform(X_train['prefix_teacher'])
X_cv_prefix_one_hot = vectorizer_teacher.transform(X_cv['prefix_teacher'])
X_test_prefix_one_hot = vectorizer_teacher.transform(X_test['prefix_teacher'])

print("After verctorizing")
print(X_train_prefix_one_hot.shape, y_train.shape)
print(X_cv_prefix_one_hot.shape, y_cv.shape)
print(X_test_prefix_one_hot.shape, y_test.shape)

print(vectorizer_teacher.get_feature_names())
print(X_train_prefix_one_hot[0:3])
```

```
After verctorizing
(49041, 5) (49041,)
(24155, 5) (24155,)
(36052, 5) (36052,)
['dr', 'teacher', 'mr', 'ms', 'mrs']
  (0, 3)        1
  (1, 4)        1
  (2, 1)        1
```

school state

In [186]:

```
# Vectorizing "school_state"
from collections import Counter
my_counter = Counter()
for word in X_train['school_state'].values:
    my_counter.update(word.split())

state_dict = dict(my_counter)
sorted_state_dict = dict(sorted(state_dict.items(), key=lambda kv: kv[1]))

vectorizer_state = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=Fal
vectorizer_state.fit(X_train['school_state'].values)

X_train_state_one_hot = vectorizer_state.transform(X_train['school_state'].values)
X_cv_state_one_hot = vectorizer_state.transform(X_cv['school_state'].values)
X_test_state_one_hot = vectorizer_state.transform(X_test['school_state'].values)

print("After verctorizing")
print(X_train_state_one_hot.shape, y_train.shape)
print(X_cv_state_one_hot.shape, y_cv.shape)
print(X_test_state_one_hot.shape, y_test.shape)

print(vectorizer_state.get_feature_names())

print(X_train_state_one_hot[0:3])
```

```
After verctorizing
(49041, 51) (49041,)
(24155, 51) (24155,)
(36052, 51) (36052,)
['VT', 'WY', 'ND', 'MT', 'NE', 'SD', 'RI', 'AK', 'NH', 'DE', 'WV', 'ME', 'H
I', 'DC', 'NM', 'KS', 'IA', 'ID', 'AR', 'CO', 'OR', 'MN', 'KY', 'MS', 'NV',
'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ', 'NJ', 'OK', 'WA', 'LA', 'M
A', 'OH', 'IN', 'MO', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'NY', 'TX',
'CA']
  (0, 32)       1
  (1, 44)       1
  (2, 47)       1
```

Project Grade Category

In [187]:

```
# Vectorizing "project_grade_category"
vectorizer_grade = CountVectorizer(vocabulary=list(sorted_grade_dict.keys()), lowercase=Fal
vectorizer_grade.fit(X_train['project_grade'])

X_train_grade_one_hot = vectorizer_grade.transform(X_train['project_grade'])
X_cv_grade_one_hot = vectorizer_grade.transform(X_cv['project_grade'])
X_test_grade_one_hot = vectorizer_grade.transform(X_test['project_grade'])

print("After verctorizing")
print(X_train_grade_one_hot.shape, y_train.shape)
print(X_cv_grade_one_hot.shape, y_cv.shape)
print(X_test_grade_one_hot.shape, y_test.shape)

print(vectorizer_grade.get_feature_names())
print(X_train_grade_one_hot[0:3])
```

```
After verctorizing
(49041, 4) (49041,)
(24155, 4) (24155,)
(36052, 4) (36052,)
['grades_9_12', 'grades_6_8', 'grades_3_5', 'grades_prek_2']
  (0, 3)        1
  (1, 2)        1
  (2, 2)        1
```

Price

In [86]:

```
price_train = X_train['price'].values.reshape(-1,1)
price_cv = X_cv['price'].values.reshape(-1,1)
price_test = X_test['price'].values.reshape(-1,1)

print(price_train.shape, y_train.shape)
print(price_cv.shape, y_cv.shape)
print(price_test.shape, y_test.shape)
```

```
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

Resource Quantity

In [84]:

```
quantity_train = X_train['quantity'].values.reshape(-1,1)
quantity_cv = X_cv['quantity'].values.reshape(-1,1)
quantity_test = X_test['quantity'].values.reshape(-1,1)

print(quantity_train.shape, y_train.shape)
print(quantity_cv.shape, y_cv.shape)
print(quantity_test.shape, y_test.shape)
```

```
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

Number of previously posted assignmnets by the teaxhers

In [85]:

```
number_projects_train = X_train['teacher_number_of_previously_posted_projects'].values.resh
number_projects_cv = X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-1
number_projects_test = X_test['teacher_number_of_previously_posted_projects'].values.reshap

print(number_projects_train.shape, y_train.shape)
print(number_projects_cv.shape, y_cv.shape)
print(number_projects_test.shape, y_test.shape)
```

```
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
```

Text Vectorization: Making data ready for models

BoW on Clean Essay

In [41]:

```
# We are considering only the words which appeared in at least 10 documents(rows or project
vectorizer_bow_essay = CountVectorizer(min_df=10)
vectorizer_bow_essay.fit(X_train['clean_essays'].values)

X_train_essay_bow = vectorizer_bow_essay.transform(X_train['clean_essays'].values)
X_cv_essay_bow = vectorizer_bow_essay.transform(X_cv['clean_essays'].values)
X_test_essay_bow = vectorizer_bow_essay.transform(X_test['clean_essays'].values)

print("After vectorizing")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
After vectorizing
(49041, 12070) (49041,)
(24155, 12070) (24155,)
(36052, 12070) (36052,)
```

BoW on Clean Title

In [42]:

```
vectorizer_bow_title = CountVectorizer(min_df=10)
vectorizer_bow_title.fit(X_train['clean_titles'].values)

X_train_titles_bow = vectorizer_bow_title.transform(X_train['clean_titles'].values)
X_cv_titles_bow = vectorizer_bow_title.transform(X_cv['clean_titles'].values)
X_test_titles_bow = vectorizer_bow_title.transform(X_test['clean_titles'].values)

print("After vectorizing")
print(X_train_titles_bow.shape, y_train.shape)
print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)
```

```
After vectorizing
(49041, 2016) (49041,)
(24155, 2016) (24155,)
(36052, 2016) (36052,)
```

Tfidf on Clean Essay

In [43]:

```
from sklearn.feature_extraction.text import TfidfVectorizer


vectorizer_tfidf_essay = TfidfVectorizer(min_df=10)
vectorizer_tfidf_essay.fit(X_train['clean_essays'].values)

X_train_essay_tfidf = vectorizer_tfidf_essay.transform(X_train['clean_essays'])
X_cv_essay_tfidf = vectorizer_tfidf_essay.transform(X_cv['clean_essays'])
X_test_essay_tfidf = vectorizer_tfidf_essay.transform(X_test['clean_essays'])

print("After vectorizing")
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 vectorizing
(49041, 12070) (49041,)
(24155, 12070) (24155,)
(36052, 12070) (36052,)
```

Tfidf on Clean Title

In [44]:

```
vectorizer_tfidf_title = TfidfVectorizer(min_df=10)
vectorizer_tfidf_title.fit(X_train['clean_titles'].values)

X_train_title_tfidf = vectorizer_tfidf_title.transform(X_train['clean_titles'])
X_cv_title_tfidf = vectorizer_tfidf_title.transform(X_cv['clean_titles'])
X_test_title_tfidf = vectorizer_tfidf_title.transform(X_test['clean_titles'])

print("After vectorizing")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
```

```
After vectorizing
(49041, 2016) (49041,)
(24155, 2016) (24155,)
(36052, 2016) (36052,)
```

Avg W2V on Clean Essay

In [45]:

```
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')


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

for i in preprocessed_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-pickl

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

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

```
Loading Glove Model

1917495it [10:50, 2947.85it/s]

Done. 1917495  words loaded!
all the words in the coupus 15390794
the unique words in the coupus 58252
The number of words that are present in both glove vectors and our coupus 51
409 ( 88.253 %)
word 2 vec length 51409
```

In [46]:

```
train_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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 words_glove:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_essay_avg_w2v.append(vector)

print(len(train_essay_avg_w2v))
print(len(train_essay_avg_w2v[0]))


cv_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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 words_glove:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_essay_avg_w2v.append(vector)

print(len(cv_essay_avg_w2v))
print(len(cv_essay_avg_w2v[0]))


test_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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 words_glove:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_essay_avg_w2v.append(vector)

print(len(test_essay_avg_w2v))
print(len(test_essay_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████
███| 49041/49041 [00:39<00:00, 1228.61it/s]

49041
300

100%|████████████████████████████████████████████████████████████████
███| 24155/24155 [00:18<00:00, 1323.91it/s]

24155
300

100%|████████████████████████████████████████████████████████████████
```

```
100%|| 36052/36052 [00:20<00:00, 1725.96it/s]

36052
300
```

In [47]:

```
# Changing list to numpy arrays
train_essay_avg_w2v = np.array(train_essay_avg_w2v)
cv_essay_avg_w2v = np.array(cv_essay_avg_w2v)
test_essay_avg_w2v = np.array(test_essay_avg_w2v)
```

Avg W2V on Clean Title

In [48]:

```
train_title_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_title_avg_w2v.append(vector)

print(len(train_title_avg_w2v))
print(len(train_title_avg_w2v[0]))


cv_title_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_title_avg_w2v.append(vector)

print(len(cv_title_avg_w2v))
print(len(cv_title_avg_w2v[0]))


test_title_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =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:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_title_avg_w2v.append(vector)

print(len(test_title_avg_w2v))
print(len(test_title_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██| 49041/49041 [00:02<00:00, 20098.41it/s]

49041
300

100%|████████████████████████████████████████████████████████████████
██| 24155/24155 [00:00<00:00, 27685.00it/s]

24155
300
```

```
100%|████████████████████████████████████████████████████████████████████
████| 36052/36052 [00:01<00:00, 31674.90it/s]

36052
300
```

In [49]:

```
# Changing list to numpy arrays
train_title_avg_w2v = np.array(train_title_avg_w2v)
cv_title_avg_w2v = np.array(cv_title_avg_w2v)
test_title_avg_w2v = np.array(test_title_avg_w2v)
```

Tfidf W2V on Clean essay

In [50]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['clean_essays'])
# 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 [51]:

```
train_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_essays']): # 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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_essay_tfidf_w2v.append(vector)

print(len(train_essay_tfidf_w2v))
print(len(train_essay_tfidf_w2v[0]))


cv_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_essays']): # 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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_essay_tfidf_w2v.append(vector)

print(len(cv_essay_tfidf_w2v))
print(len(cv_essay_tfidf_w2v[0]))


test_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_essays']): # 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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_essay_tfidf_w2v.append(vector)

print(len(test_essay_tfidf_w2v))
print(len(test_essay_tfidf_w2v[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
████| 49041/49041 [04:40<00:00, 174.91it/s]
```

```
49041
300

100%|████████████████████████████████████████████████████████████
████| 24155/24155 [02:12<00:00, 182.02it/s]

24155
300

100%|████████████████████████████████████████████████████████████
████| 36052/36052 [03:24<00:00, 176.53it/s]

36052
300
```

In [52]:

```
# Changing list to numpy arrays
train_essay_tfidf_w2v = np.array(train_essay_tfidf_w2v)
cv_essay_tfidf_w2v = np.array(cv_essay_tfidf_w2v)
test_essay_tfidf_w2v = np.array(test_essay_tfidf_w2v)
```

Tfidf W2V on Clean Title

In [53]:

```
train_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['clean_titles']): # 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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_title_tfidf_w2v.append(vector)

print(len(train_title_tfidf_w2v))
print(len(train_title_tfidf_w2v[0]))


cv_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['clean_titles']): # 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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_title_tfidf_w2v.append(vector)

print(len(cv_title_tfidf_w2v))
print(len(cv_title_tfidf_w2v[0]))



test_title_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['clean_titles']): # 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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_title_tfidf_w2v.append(vector)

print(len(test_title_tfidf_w2v))
print(len(test_title_tfidf_w2v[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
██| 49041/49041 [00:04<00:00, 11759.34it/s]
```

```
49041
300

100%|████████████████████████████████████████████████████████████████
██| 24155/24155 [00:02<00:00, 10179.91it/s]

24155
300

100%|████████████████████████████████████████████████████████████████
██| 36052/36052 [00:03<00:00, 11194.87it/s]

36052
300
```

In [54]:

```
# Changing list to numpy arrays
train_title_tfidf_w2v = np.array(train_title_tfidf_w2v)
cv_title_tfidf_w2v = np.array(cv_title_tfidf_w2v)
test_title_tfidf_w2v = np.array(test_title_tfidf_w2v)
```

Applying Decision Tree (2 Class Classification)

Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay(BOW)

Hstacking features

In [87]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)+
X_train_bow = hstack((X_train_categories_one_hot, X_train_sub_categories_one_hot, X_train_e
X_cv_bow = hstack((X_cv_categories_one_hot, X_cv_sub_categories_one_hot, X_cv_essay_bow, qu
X_test_bow = hstack((X_test_categories_one_hot, X_test_sub_categories_one_hot, X_test_essay

print('Final matrix')
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
```

```
Final matrix
(49041, 14188) (49041,)
(24155, 14188) (24155,)
(36052, 14188) (36052,)
```

Hyperparameter tuning using simple for loop

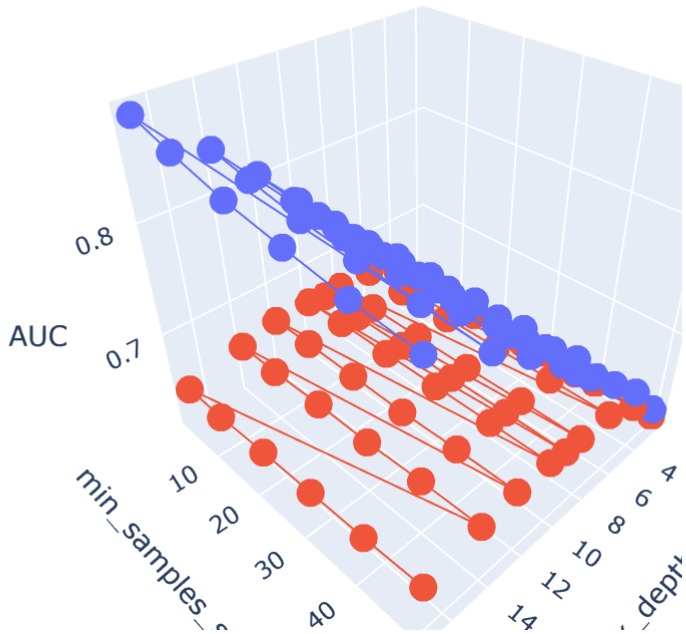
In [111]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifier
#https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

train_auc = []
cv_auc = []



max_depth = [2,4,6,8,9,10,12,14,17]
min_samples_split = [2,10,20,30,40,50]


for i in (max_depth):
    for j in (min_samples_split):

        dt_bow = DecisionTreeClassifier(criterion='gini', max_depth=i, min_samples_split=j,

        dt_bow.fit(X_train_bow, y_train)


        y_train_pred = dt_bow.predict_proba(X_train_bow)[:,1]
        y_cv_pred = dt_bow.predict_proba(X_cv_bow)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))

        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

3-D AUC plot

In [112]:

```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scatte
from mpl_toolkits.mplot3d import Axes3D
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np

max_depth_val = []
min_samples_split_val = []

for i in (max_depth):
    for j in (min_samples_split):
        max_depth_val.append(i)
        min_samples_split_val.append(j)


trace1 = go.Scatter3d(x=max_depth_val,y=min_samples_split_val,z=train_auc, name = 'train au
trace2 = go.Scatter3d(x=max_depth_val,y=min_samples_split_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]

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

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



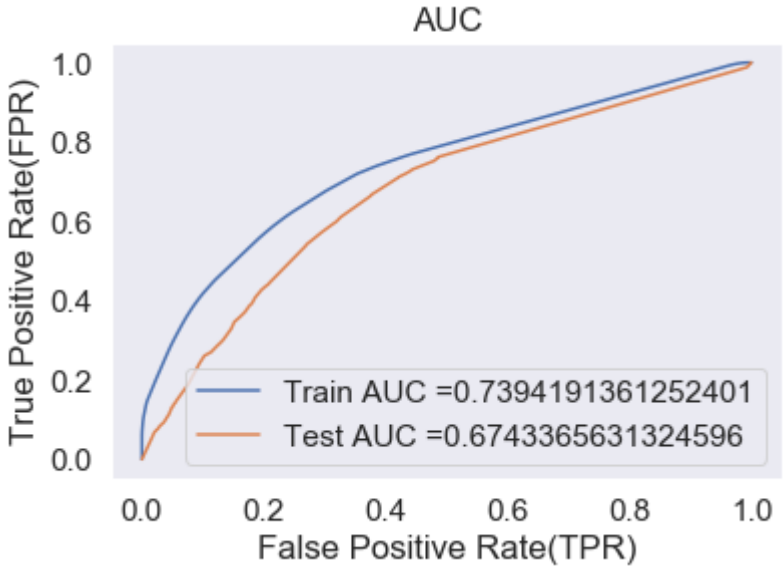
In [113]:

```
#max_depth = 9
#min_samples_split = 50
#cv auc = 0.67
#train auc = 0.73
```

Training model with optimal value of hyperparameter

In [114]:

```
from sklearn.metrics import roc_curve, auc

dt_bow = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=9, min_samples
                                    random_state=None, class_weight='balanced')
dt_bow.fit(X_train_bow, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs
y_train_pred = dt_bow.predict_proba(X_train_bow)[:,1]
y_test_pred = dt_bow.predict_proba(X_test_bow)[:,1]


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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Getting confusion metrix for both train and test set

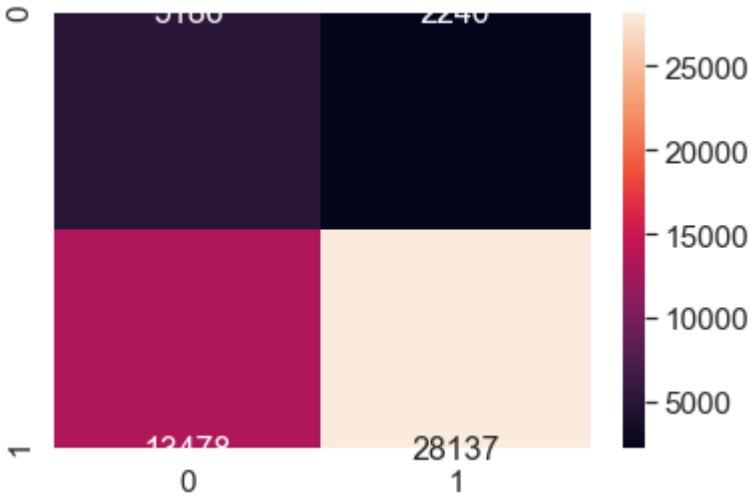
In [115]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t
    return t

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

from mlxtend.plotting import plot_confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
```

In [116]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.47217768562508494 for threshold 0.491
Train confusion matrix
[[ 5186  2240]
 [13478 28137]]
```



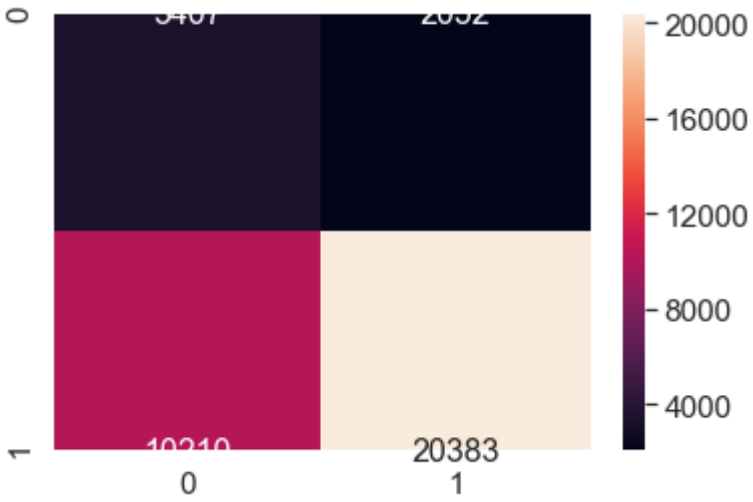
In [117]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)),r
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 0.41581971559104214 for threshold 0.472
Test confusion matrix
```

Out[117]:

```
array([[ 3407,  2052],
       [10210, 20383]], dtype=int64)
```



Evaluating model performance

In [118]:

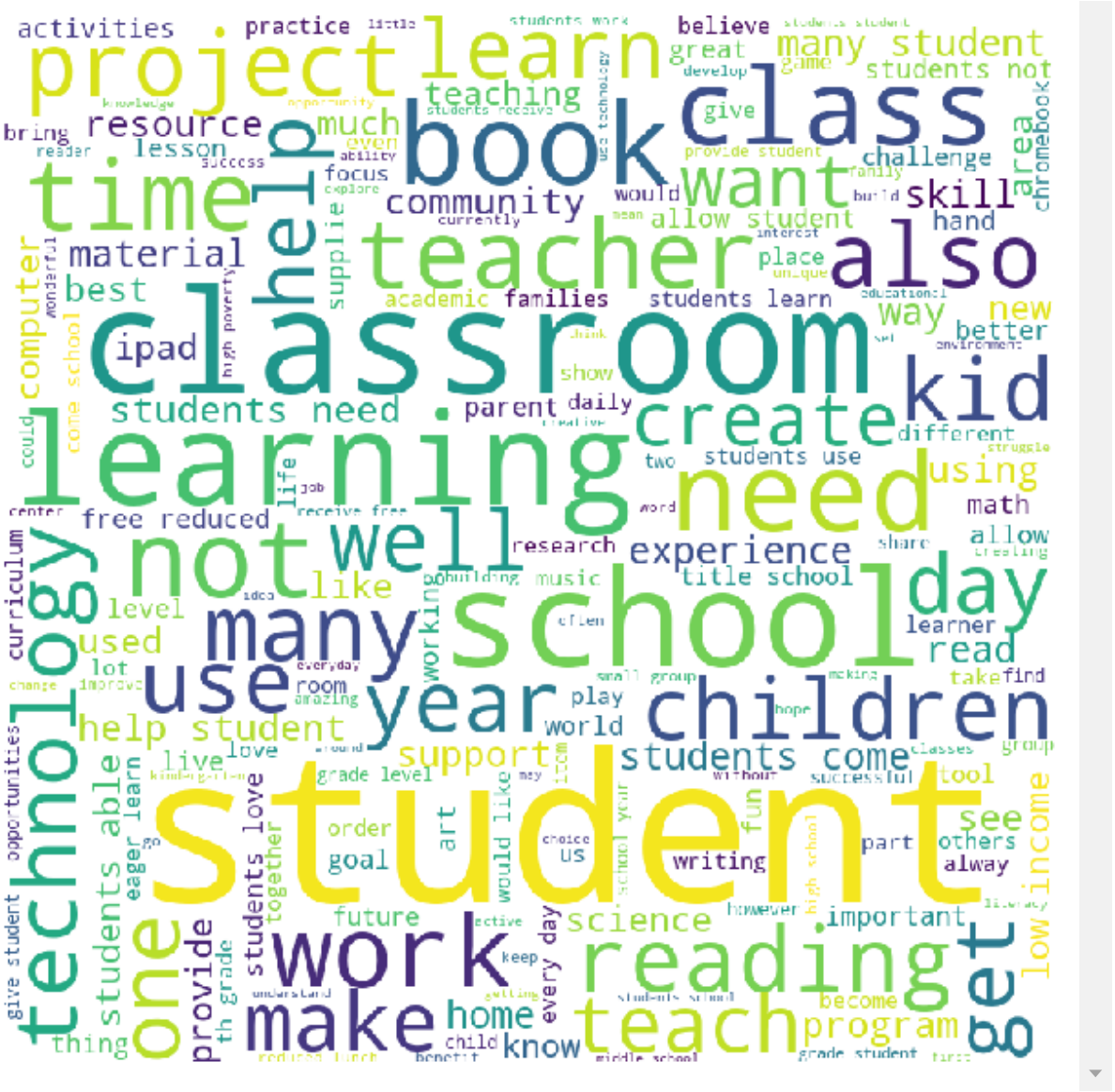
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = dt_bow.predict(X_test_bow)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 61.985%
Precision on test set: 0.913
Recall on test set: 0.610
F1-Score on test set: 0.731
```

Word cloud on Test essay -FPR

In [119]:

```
from wordcloud import WordCloud
essay_list_fpr = []
essay_list_words = []
essay_list = []
length = (X_test_bow.shape[0])-1
essay_list=X_test['clean_essays']    #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        essay_list_fpr.append(essay_list.values[i]) #append all the essay at which model pr

for sentance in (essay_list_fpr):
    sent = decontracted(sentance)
    sent = sent.replace('nannan','' ) #removing nanna and digits from the essay
    sent = re.sub('[^a-z]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = sent.split(" ")
    essay_list_words.append(sent) #appending the cleaned sentences

stream = len(essay_list_words)-1

word = []
word_string = ''
len(essay_list_words[0])
for j in range (0,stream):
    for k in range (0,len(essay_list_words[j])-1):
        word.append(essay_list_words[j][k])          #extracting each word from each senten

word_string = ' '.join(e for e in word)             #making a single string out of all the

#https://www.geeksforgeeks.org/generating-word-cloud-python/
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(word_string)

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

plt.show()
```



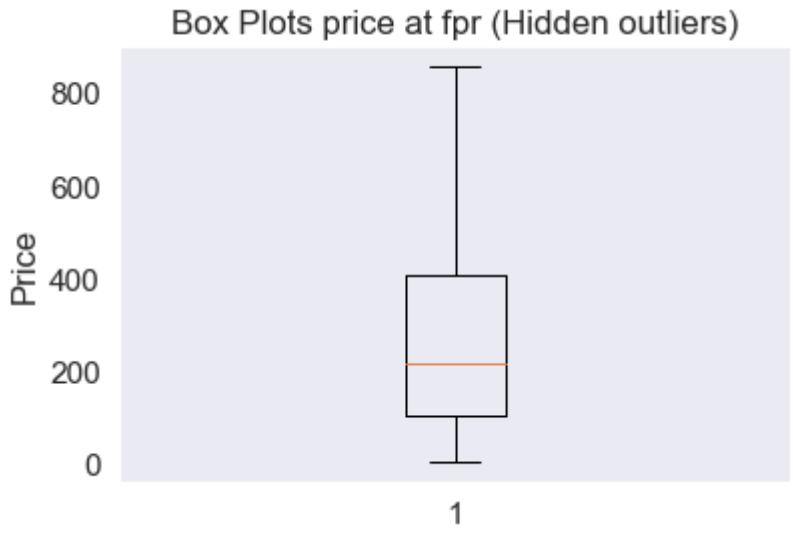
Box plot on Price -FPR

In [120]:

```
price_fpr = []
length = len(X_test['price'])-1 #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        price_fpr.append(X_test['price'].values[i])

plt.boxplot([price_fpr])   #hiding outliers
plt.title('Box Plots price at fpr')
plt.xticks([1])
plt.ylabel('Price')
plt.grid()
plt.show()
```

In [121]:

```
price_fpr = []
length = len(X_test['price'])-1 #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        price_fpr.append(X_test['price'].values[i])

plt.boxplot([price_fpr],showfliers=False)   #hiding outliers
plt.title('Box Plots price at fpr (Hidden outliers)')
plt.xticks([1])
plt.ylabel('Price')
plt.grid()
plt.show()
```

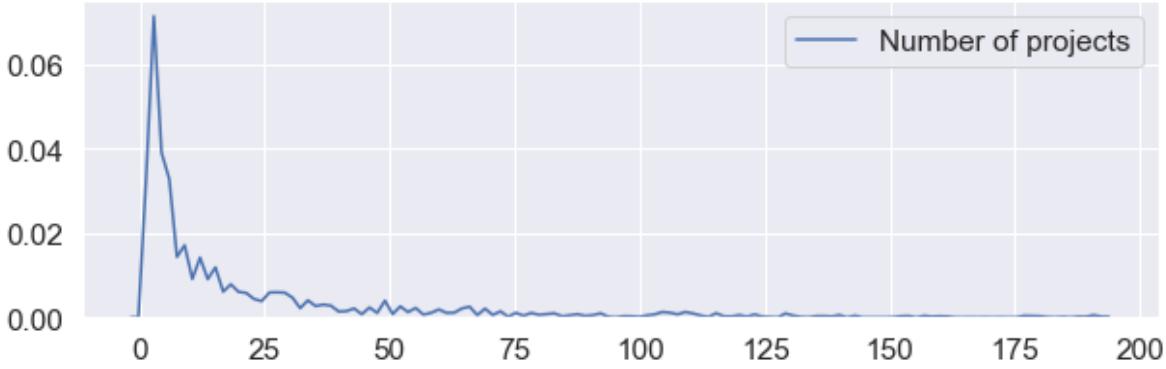


1. The model made 25 percent of wrong prediction when price are under the range of 100.
2. The model made 50 percent of wrong prediction when price are under the range of 200.
3. The model made 75 percent of wrong prediction when price are under the range of 390.
4. There are many outliers at price range above 800.

PDF (Number of previously posted asssignment by teachers) - FPR

In [122]:

```
pronum_fpr = []
length = len(X_test['teacher_number_of_previously_posted_projects'])-1 #listing out all the
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        pronum_fpr.append(X_test['teacher_number_of_previously_posted_projects'].values[i])

plt.figure(figsize=(10,3))
sns.kdeplot(pronum_fpr,label="Number of projects", bw=0.6)
plt.legend()
plt.show()
```



1. Mostly 1-6 projects have been previously posted by the teachers.
2. Few teaches have posted 10-30 projects.

Visualizing decision tree

In [123]:

```
features_list = []
#Appending features in same order as prob value of features stored

for x in vectorizer_sbj.get_feature_names():
    features_list.append(x)

for x in vectorizer_sub_sbj.get_feature_names():
    features_list.append(x)

for x in vectorizer_bow_essay.get_feature_names():
    features_list.append(x)

features_list.append("quantity")

for x in vectorizer_state.get_feature_names():
    features_list.append(x)

for x in vectorizer_teacher.get_feature_names():
    features_list.append(x)

for x in vectorizer_grade.get_feature_names():
    features_list.append(x)

for x in vectorizer_bow_title.get_feature_names():
    features_list.append(x)

features_list.append("price")

features_list.append("teacher_number_of_previously_posted_projects")

len(features_list)
```

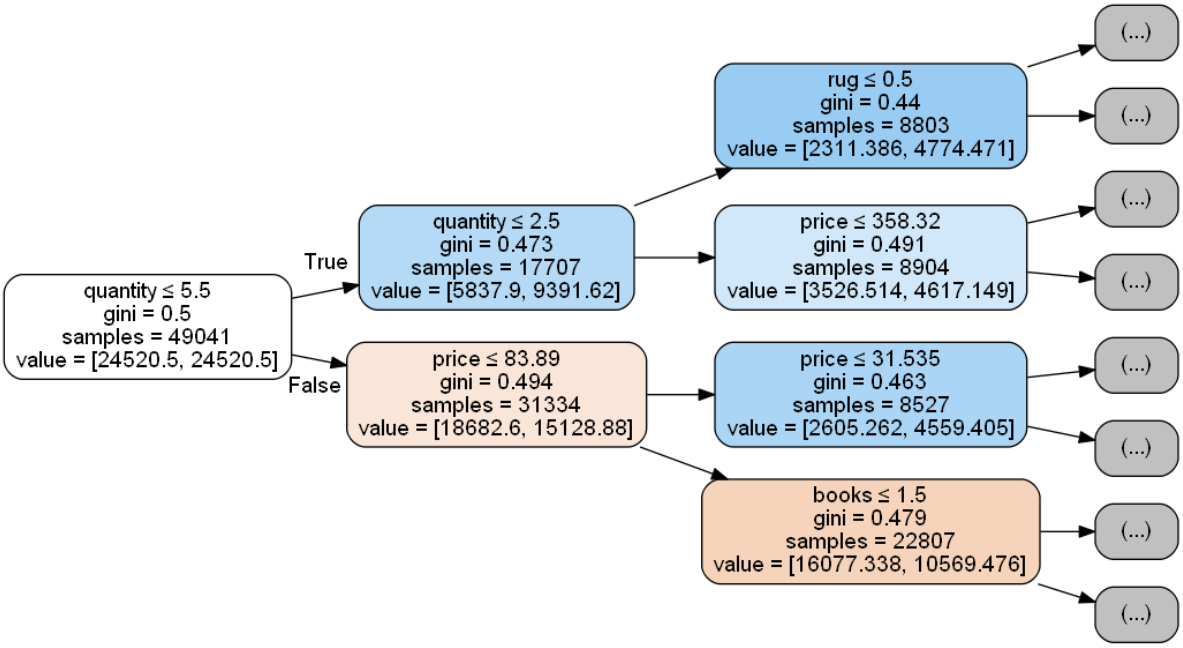
Out[123]:

```
14188
```

In [124]:

```
import os
os.environ["PATH"] += os.pathsep + 'C:/Users/VANSHIKA/Anaconda3/Library/bin/graphviz'

#https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html
import warnings
warnings.filterwarnings("ignore")
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
string_data = StringIO()
export_graphviz(dt_bow, max_depth=2, out_file=string_data, filled=True, rounded=True, speci
graph = pydotplus.graph_from_dot_data(string_data.getvalue())
Image(graph.create_png())
```

Out[124]:



Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay(TFIDF)

Hstacking features

In [125]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_tfidf = hstack((X_train_categories_one_hot, X_train_sub_categories_one_hot, X_train
X_cv_tfidf = hstack((X_cv_categories_one_hot, X_cv_sub_categories_one_hot, X_cv_essay_tfidf
X_test_tfidf = hstack((X_test_categories_one_hot, X_test_sub_categories_one_hot, X_test_ess

print('Final matrix')
print(X_train_tfidf.shape, y_train.shape)
print(X_cv_tfidf.shape, y_cv.shape)
print(X_test_tfidf.shape, y_test.shape)
```

```
Final matrix
(49041, 14188) (49041,)
(24155, 14188) (24155,)
(36052, 14188) (36052,)
```

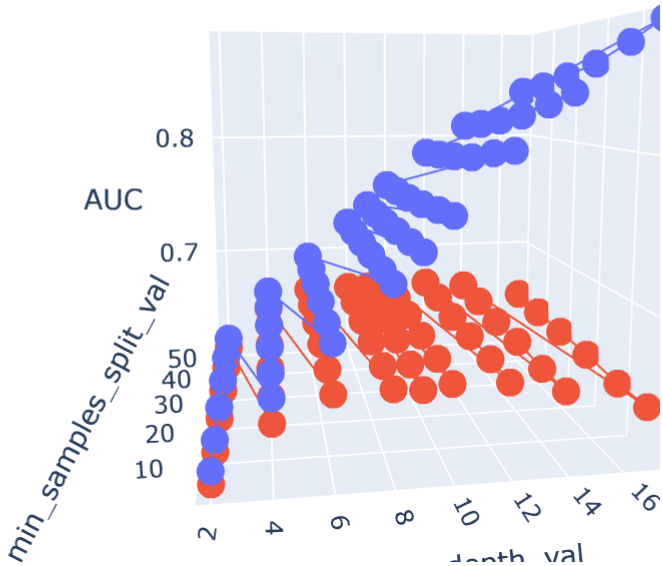
Hyperparameter tuning

In [126]:

```
from sklearn.linear_model import SGDClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
import math

max_depth = [2,4,6,8,9,10,12,14,17]
min_samples_split = [2,10,20,30,40,50]


for i in (max_depth):
    for j in (min_samples_split):

        dt_tfidf = DecisionTreeClassifier(criterion='gini', max_depth=i, min_samples_split=

        dt_tfidf.fit(X_train_tfidf, y_train)


        y_train_pred = dt_tfidf.predict_proba(X_train_tfidf)[:,1]
        y_cv_pred = dt_tfidf.predict_proba(X_cv_tfidf)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

In [127]:

```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scatte

trace1 = go.Scatter3d(x=max_depth_val,y=min_samples_split_val,z=train_auc, name = 'train au
trace2 = go.Scatter3d(x=max_depth_val,y=min_samples_split_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]

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

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

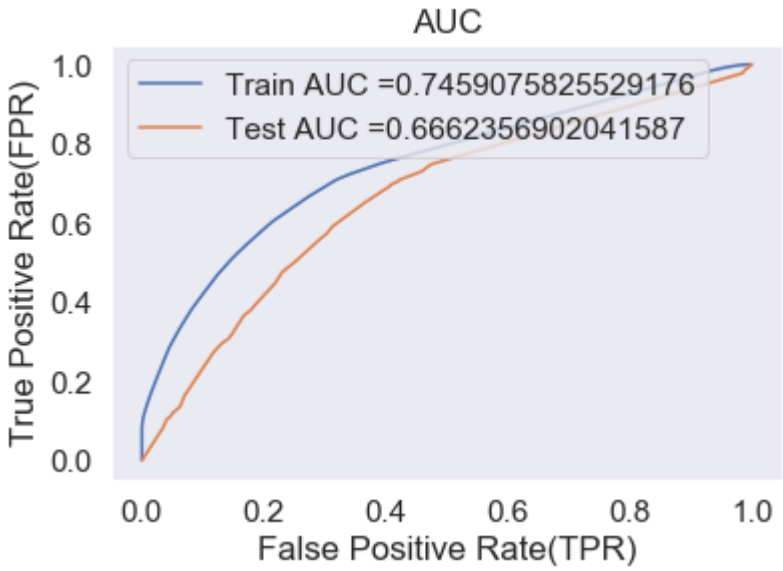


In [128]:

```
#max_depth = 9
#min_samples_split = 50
#cv auc = 0.66
#train auc = 0.74
```

In [129]:

```
from sklearn.metrics import roc_curve, auc

dt_tfidf = DecisionTreeClassifier(criterion='gini', max_depth=9, min_samples_split=50,class
dt_tfidf.fit(X_train_tfidf, y_train)

y_train_pred = dt_tfidf.predict_proba(X_train_tfidf)[:,1]
y_test_pred = dt_tfidf.predict_proba(X_test_tfidf)[:,1]

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

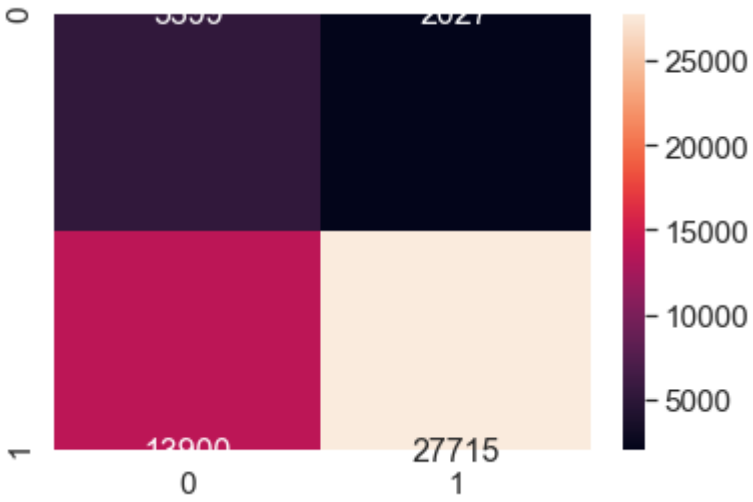


Confusion matrix

In [130]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.48419841842775424 for threshold 0.515
Train confusion matrix
[[ 5399  2027]
 [13900 27715]]
```



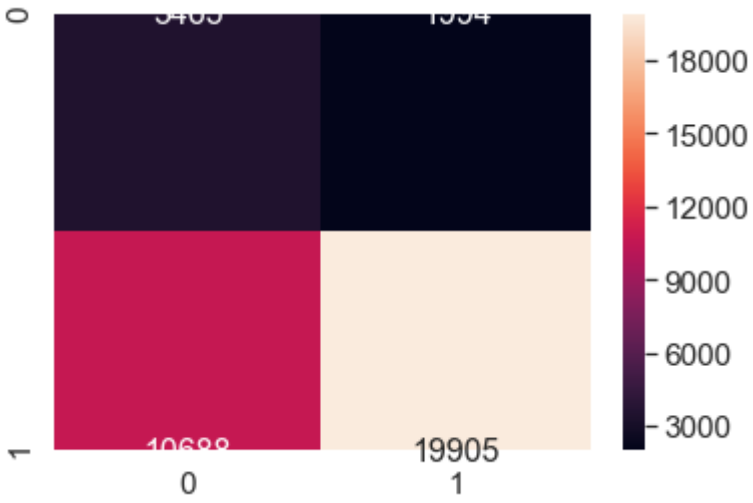
In [131]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)),r
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 0.41298117906746135 for threshold 0.515
Test confusion matrix
```

Out[131]:

```
array([[ 3465,  1994],
       [10688, 19905]], dtype=int64)
```



Evaluating model performance

In [132]:

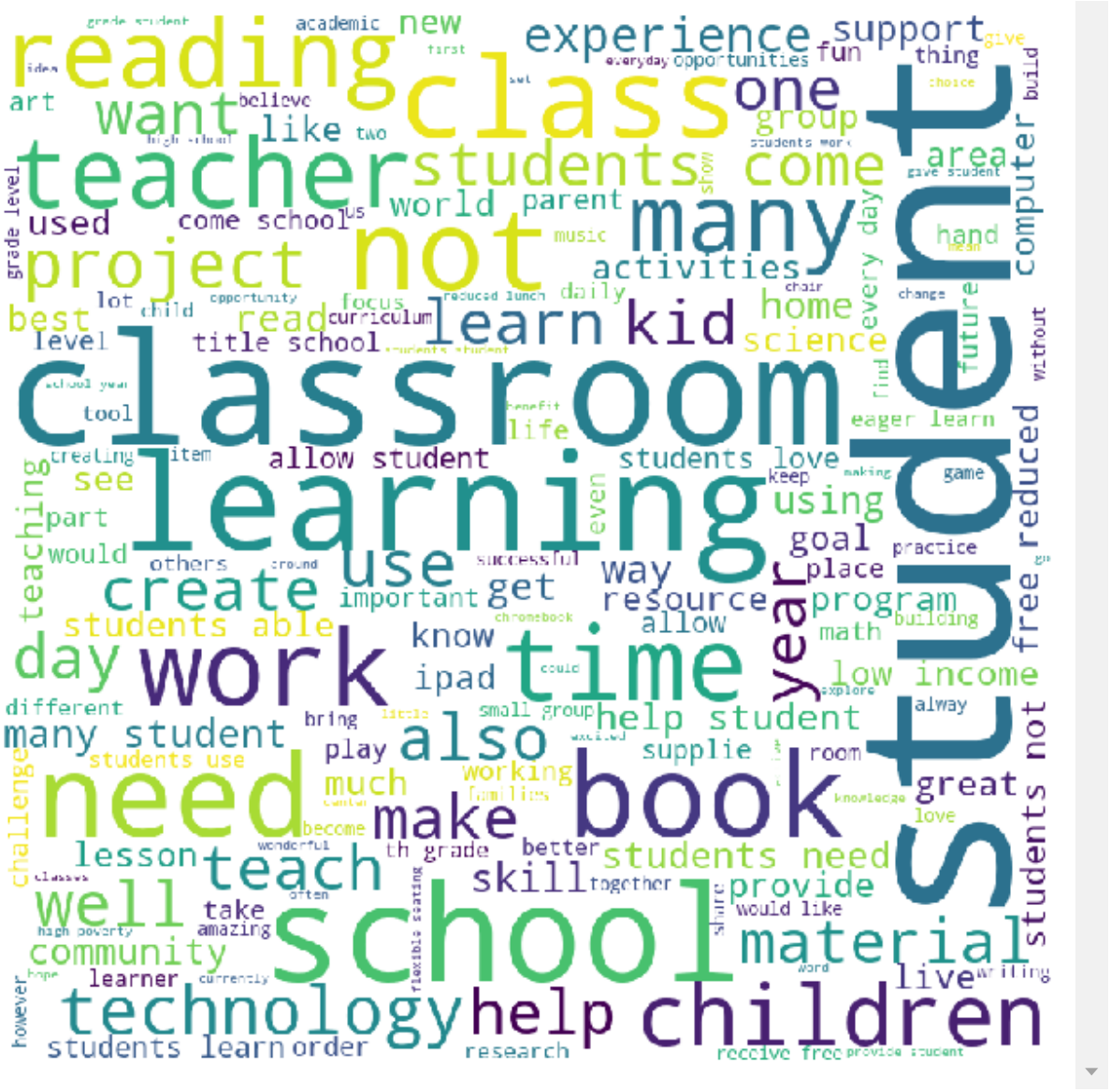
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = dt_tfidf.predict(X_test_tfidf)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 64.823%
Precision on test set: 0.909
Recall on test set: 0.651
F1-Score on test set: 0.758
```

Word Cloud on test essay -FPR

In [133]:

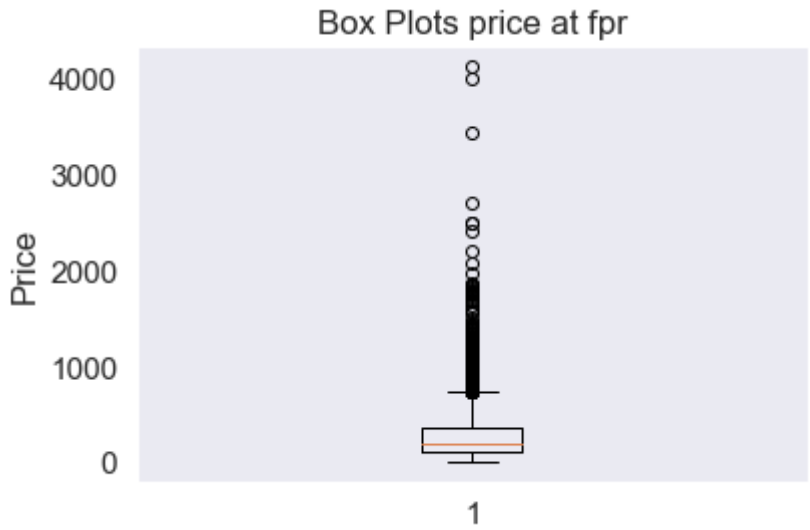
```
from wordcloud import WordCloud
essay_list_fpr = []
essay_list_words = []
essay_list = []
length = (X_test_tfidf.shape[0])-1
essay_list=X_test['clean_essays']    #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        essay_list_fpr.append(essay_list.values[i]) #append all the essay at which model pr

for sentance in (essay_list_fpr):
    sent = decontracted(sentance)
    sent = sent.replace('nannan','' ) #removing nanna and digits from the essay
    sent = re.sub('[^a-z]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = sent.split(" ")
    essay_list_words.append(sent) #appending the cleaned sentences

stream = len(essay_list_words)-1

word = []
word_string = ''
len(essay_list_words[0])
for j in range (0,stream):
    for k in range (0,len(essay_list_words[j])-1):
        word.append(essay_list_words[j][k])          #extracting each word from each senten

word_string = ' '.join(e for e in word)             #making a single string out of all the

#https://www.geeksforgeeks.org/generating-word-cloud-python/
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(word_string)

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

plt.show()
```



Box plot on price -FPR

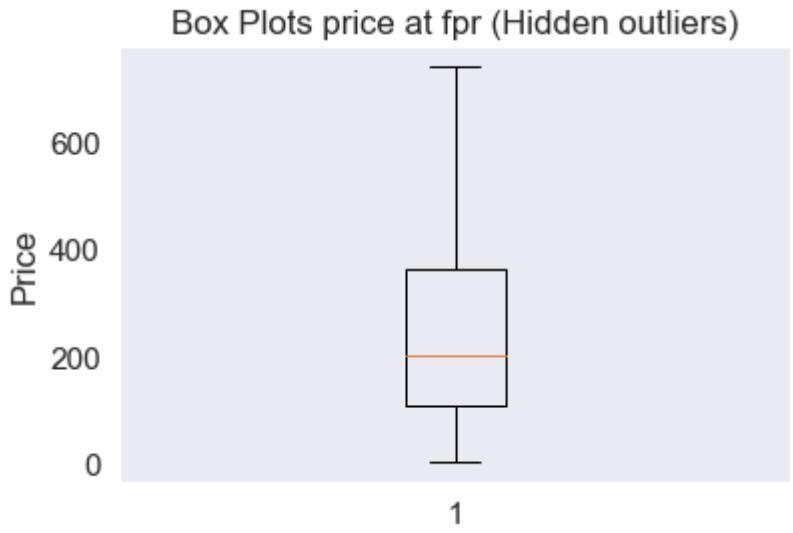
In [134]:

```
price_fpr = []
length = len(X_test['price'])-1 #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        price_fpr.append(X_test['price'].values[i])

plt.boxplot([price_fpr])
plt.title('Box Plots price at fpr')
plt.xticks([1])
plt.ylabel('Price')
plt.grid()
plt.show()
```



In [135]:

```
price_fpr = []
length = len(X_test['price'])-1 #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        price_fpr.append(X_test['price'].values[i])

plt.boxplot([price_fpr],showfliers=False)
plt.title('Box Plots price at fpr (Hidden outliers)')
plt.xticks([1])
plt.ylabel('Price')
plt.grid()
plt.show()
```

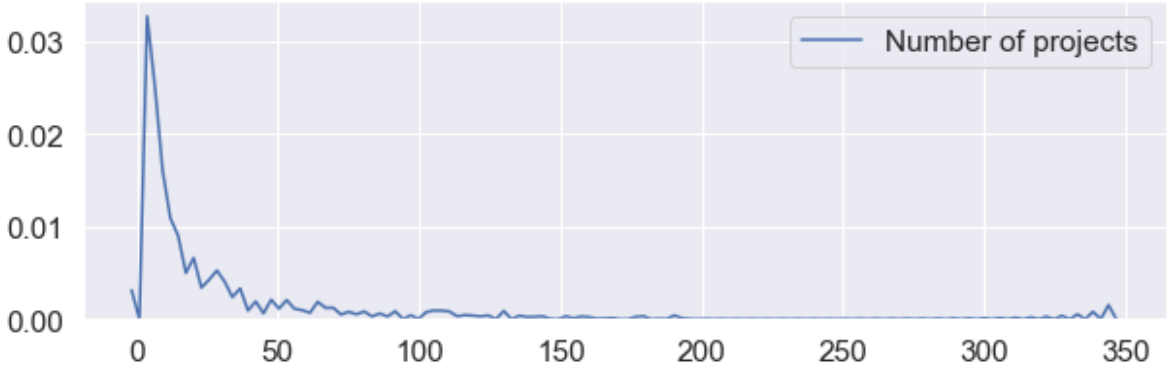


1. The model made 25 percent of wrong prediction when price are under the range of 100.
2. The model made 50 percent of wrong prediction when price are under the range of 210.
3. The model made 75 percent of wrong prediction when price are under the range of 400.
4. There are many outliers above 800 price range.

PDF (Number of previously posted asssignment by teachers) - FPR

In [136]:

```
pronum_fpr = []
length = len(X_test['teacher_number_of_previously_posted_projects'])-1 #listing out all the
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        pronum_fpr.append(X_test['teacher_number_of_previously_posted_projects'].values[i])

plt.figure(figsize=(10,3))
sns.kdeplot(pronum_fpr,label="Number of projects", bw=0.6)
plt.legend()
plt.show()
```



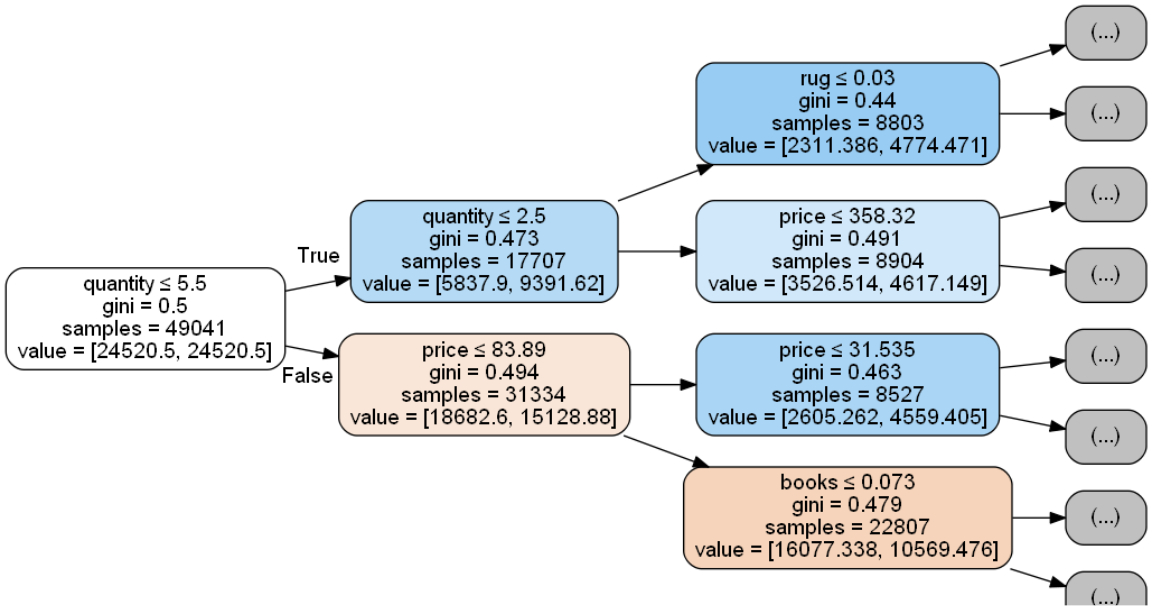
1. Mostly 1-7 projects have been previously posted by the teachers.
2. Few teaches have posted 10-40 projects

Visualizing Decision tree

In [137]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html
import warnings
warnings.filterwarnings("ignore")
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
string_data = StringIO()
export_graphviz(dt_tfidf, max_depth=2, out_file=string_data, filled=True, rounded=True, spe
graph = pydotplus.graph_from_dot_data(string_data.getvalue())
Image(graph.create_png())
```

Out[137]:



Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)

Hstacking features

In [138]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_avg = hstack((X_train_categories_one_hot, X_train_sub_categories_one_hot, train_ess
X_cv_avg = hstack((X_cv_categories_one_hot, X_cv_sub_categories_one_hot, cv_essay_avg_w2v,
X_test_avg = hstack((X_test_categories_one_hot, X_test_sub_categories_one_hot, test_essay_a

print('Final matrix')
print(X_train_avg.shape, y_train.shape)
print(X_cv_avg.shape, y_cv.shape)
print(X_test_avg.shape, y_test.shape)
```

```
Final matrix
(49041, 702) (49041,)
(24155, 702) (24155,)
(36052, 702) (36052,)
```

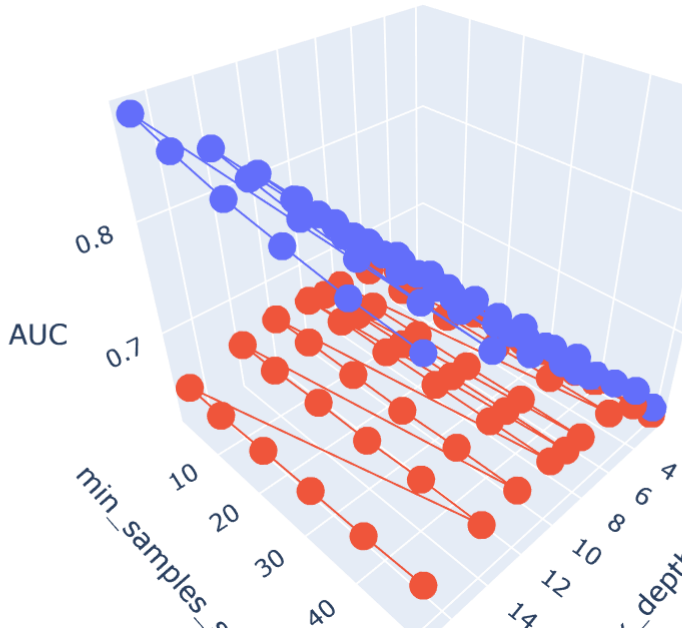
Hyperparameter Tuning

In [194]:

```
from sklearn.linear_model import SGDClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
import math
max_depth = [2,4,6,8,9,10,12,14,17]
min_samples_split = [2,10,20,30,40,50]

for i in (max_depth):
    for j in (min_samples_split):

        dt_avg = DecisionTreeClassifier(criterion='gini', max_depth=i, min_samples_split=j,

        dt_avg.fit(X_train_avg, y_train)


        y_train_pred = dt_avg.predict_proba(X_train_avg)[:,1]
        y_cv_pred = dt_avg.predict_proba(X_cv_avg)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

In [195]:

```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scatte

trace1 = go.Scatter3d(x=max_depth_val,y=min_samples_split_val,z=train_auc, name = 'train au
trace2 = go.Scatter3d(x=max_depth_val,y=min_samples_split_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]

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

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

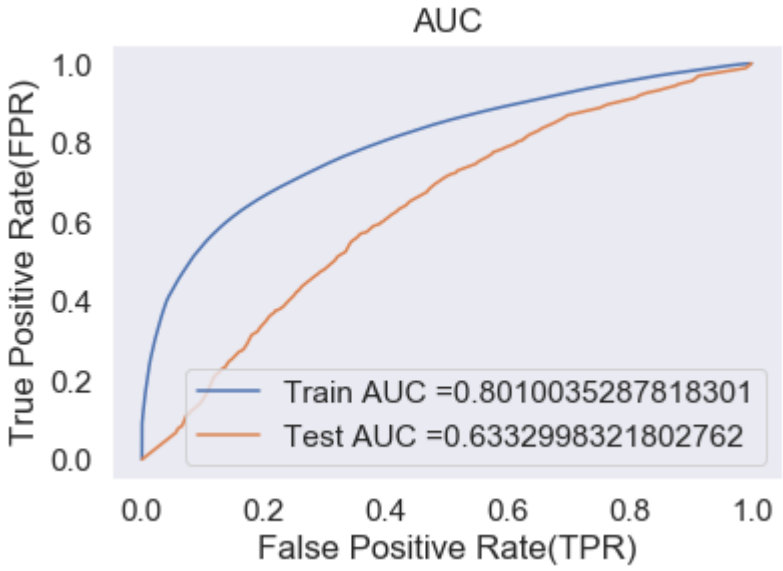


In [196]:

```
#max_depth = 9
#min_samples_split = 50
#cv auc = 0.64
#train auc = 0.79
```

In [197]:

```
from sklearn.metrics import roc_curve, auc

dt_avg = DecisionTreeClassifier(criterion='gini', max_depth=9, min_samples_split=50,class_w

dt_avg.fit(X_train_avg, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = dt_avg.predict_proba(X_train_avg)[:,1]
y_test_pred = dt_avg.predict_proba(X_test_avg)[:,1]

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

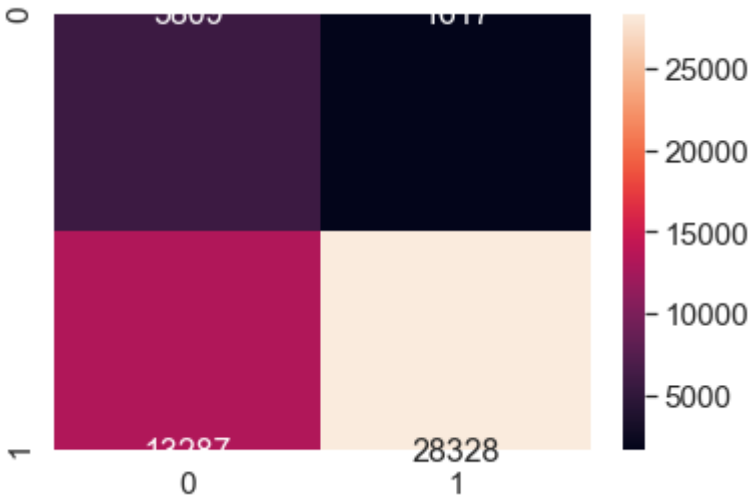


Confusion matrix

In [198]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.5324912139639202 for threshold 0.462
Train confusion matrix
[[ 5809  1617]
 [13287 28328]]
```



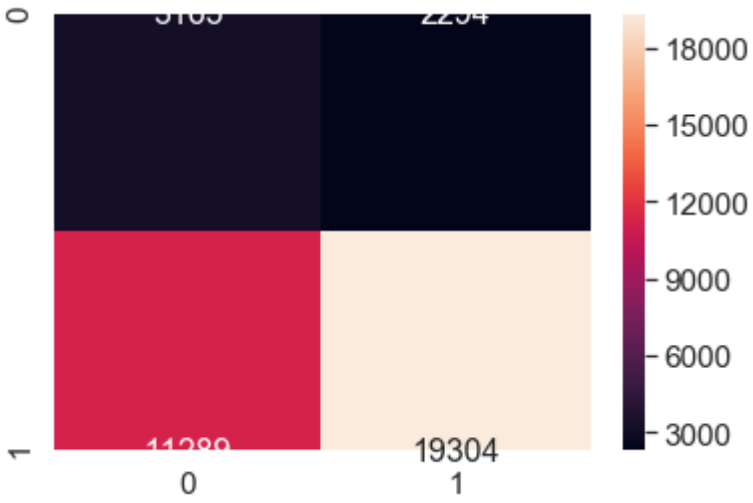
In [199]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)),r
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 0.3658355134141622 for threshold 0.465
Test confusion matrix
```

Out[199]:

```
array([[ 3165,  2294],
       [11289, 19304]], dtype=int64)
```



Evaluating model performance

In [200]:

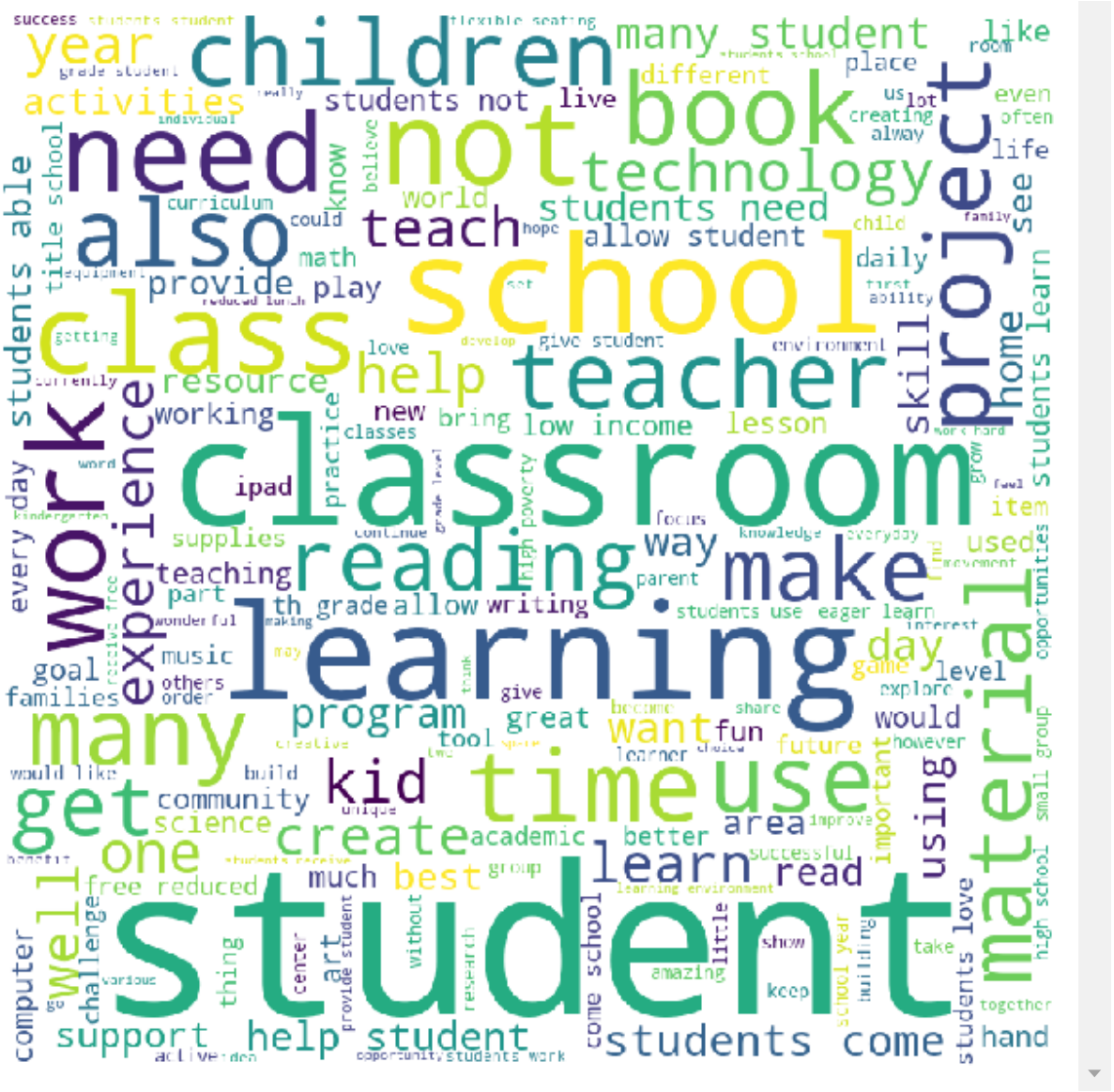
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = dt_avg.predict(X_test_avg)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 60.127%
Precision on test set: 0.895
Recall on test set: 0.601
F1-Score on test set: 0.719
```

Word cloud

In [201]:

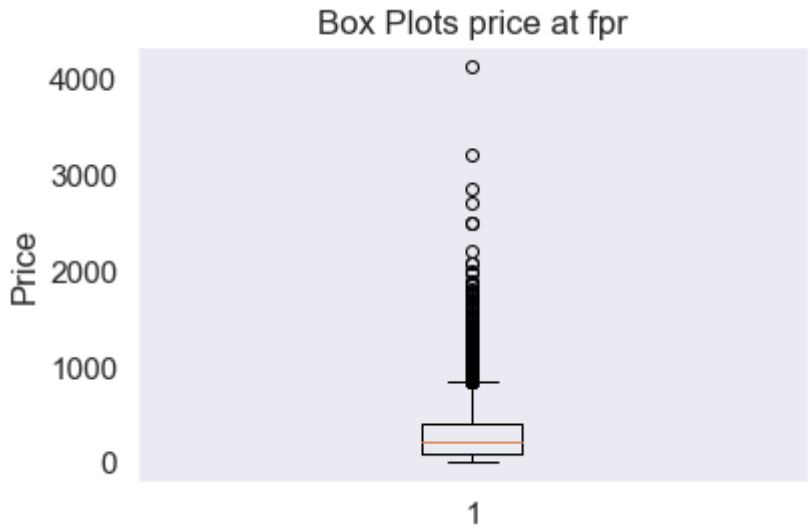
```
from wordcloud import WordCloud
essay_list_fpr = []
essay_list_words = []
length = (X_test_avg.shape[0])-1
essay_list=X_test['clean_essays']    #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        essay_list_fpr.append(essay_list.values[i]) #append all the essay at which model pr

for sentance in (essay_list_fpr):
    sent = decontracted(sentance)
    sent = sent.replace('nannan','' ) #removing nanna and digits from the essay
    sent = re.sub('[^a-z]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = sent.split(" ")
    essay_list_words.append(sent) #appending the cleaned sentences

stream = len(essay_list_words)-1

word = []
word_string = ''
len(essay_list_words[0])
for j in range (0,stream):
    for k in range (0,len(essay_list_words[j])-1):
        word.append(essay_list_words[j][k])          #extracting each word from each senten

word_string = ' '.join(e for e in word)             #making a single string out of all the

#https://www.geeksforgeeks.org/generating-word-cloud-python/
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(word_string)

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

plt.show()
```



Box plot on price -FPR

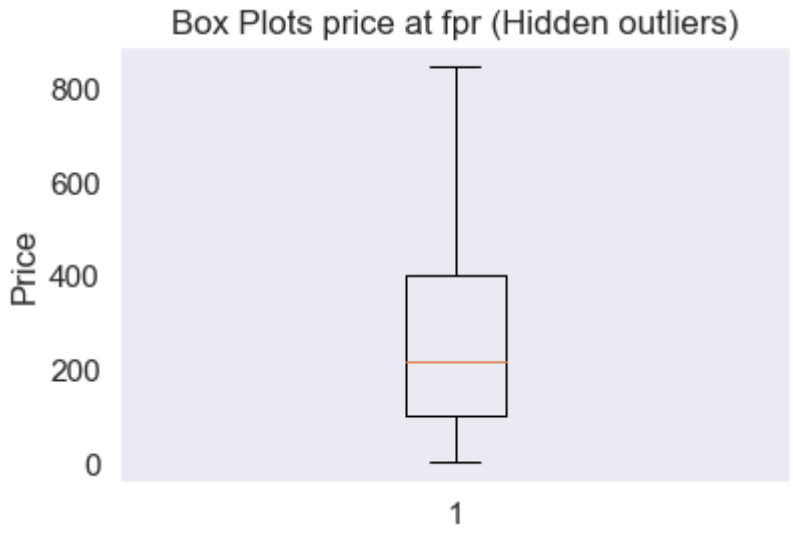
In [202]:

```
price_fpr = []
length = len(X_test['price'])-1 #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        price_fpr.append(X_test['price'].values[i])

plt.boxplot([price_fpr])
plt.title('Box Plots price at fpr')
plt.xticks([1])
plt.ylabel('Price')
plt.grid()
plt.show()
```



In [203]:

```
price_fpr = []
length = len(X_test['price'])-1 #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        price_fpr.append(X_test['price'].values[i])

plt.boxplot([price_fpr],showfliers=False)
plt.title('Box Plots price at fpr (Hidden outliers)')
plt.xticks([1])
plt.ylabel('Price')
plt.grid()
plt.show()
```

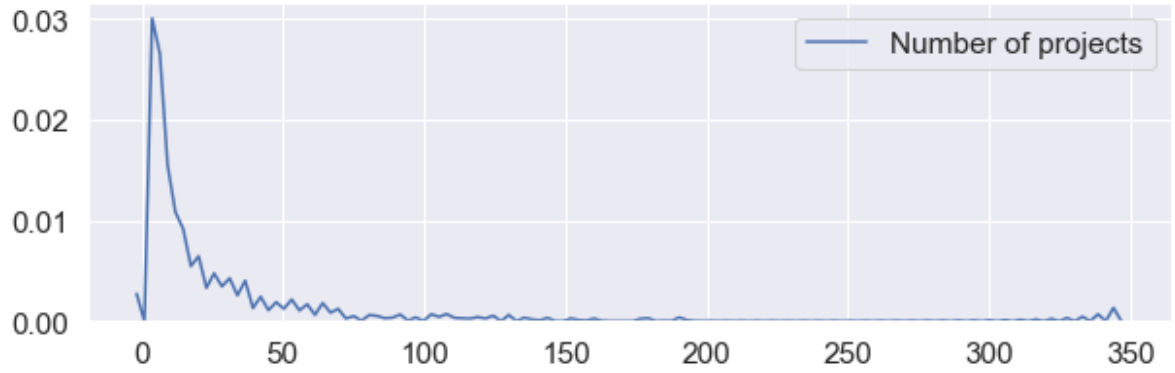


1. The model made 25 percent of wrong prediction when price are under the range of 100.
2. The model made 50 percent of wrong prediction when price are under the range of 210.
3. The model made 75 percent of wrong prediction when price are under the range of 390.
4. There are many outliers above 800 price range.

PDF (Number of previously posted asssignment by teachers) - FPR

In [204]:

```
pronum_fpr = []
length = len(X_test['teacher_number_of_previously_posted_projects'])-1 #listing out all the
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        pronum_fpr.append(X_test['teacher_number_of_previously_posted_projects'].values[i])

plt.figure(figsize=(10,3))
sns.kdeplot(pronum_fpr,label="Number of projects", bw=0.6)
plt.legend()
plt.show()
```



1. Mostly 1-5 projects have been previously posted by the teachers.
2. Few teaches have posted 10-30 projects.

Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

Hstacking features

In [205]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_tfidf_w2v = hstack((X_train_categories_one_hot, X_train_sub_categories_one_hot, tra
X_cv_tfidf_w2v = hstack((X_cv_categories_one_hot, X_cv_sub_categories_one_hot, cv_essay_tfi
X_test_tfidf_w2v = hstack((X_test_categories_one_hot, X_test_sub_categories_one_hot, test_e

print('Final matrix')
print(X_train_tfidf_w2v.shape, y_train.shape)
print(X_cv_tfidf_w2v.shape, y_cv.shape)
print(X_test_tfidf_w2v.shape, y_test.shape)
```

```
Final matrix
(49041, 702) (49041,)
(24155, 702) (24155,)
(36052, 702) (36052,)
```

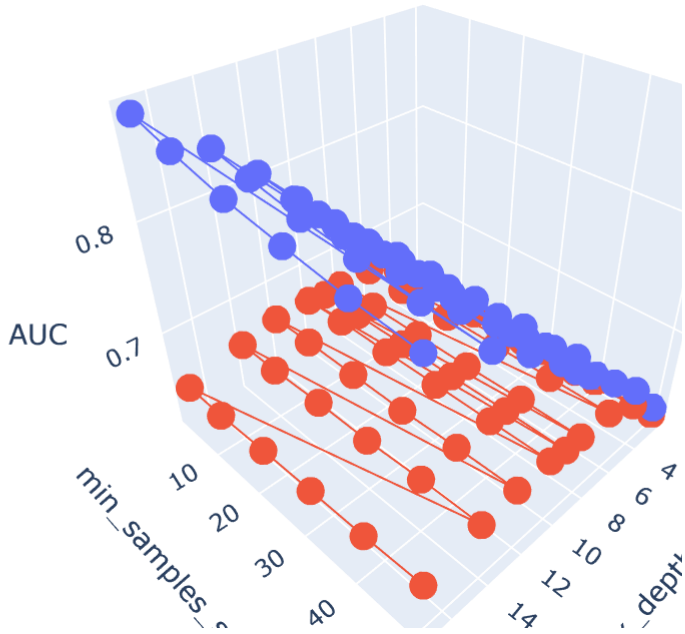
Hyperparameter tuning

In [206]:

```
from sklearn.linear_model import SGDClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
import math

max_depth = [2,4,6,8,9,10,12,14,17]
min_samples_split = [2,10,20,30,40,50]


for i in (max_depth):
    for j in (min_samples_split):

        dt_tfidf_w2v = DecisionTreeClassifier(criterion='gini', max_depth=i, min_samples_sp

        dt_tfidf_w2v.fit(X_train_tfidf_w2v, y_train)


        y_train_pred = dt_tfidf_w2v.predict_proba(X_train_tfidf_w2v)[:,1]
        y_cv_pred = dt_tfidf_w2v.predict_proba(X_cv_tfidf_w2v)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

In [207]:

```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scatte

trace1 = go.Scatter3d(x=max_depth_val,y=min_samples_split_val,z=train_auc, name = 'train au
trace2 = go.Scatter3d(x=max_depth_val,y=min_samples_split_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]

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

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

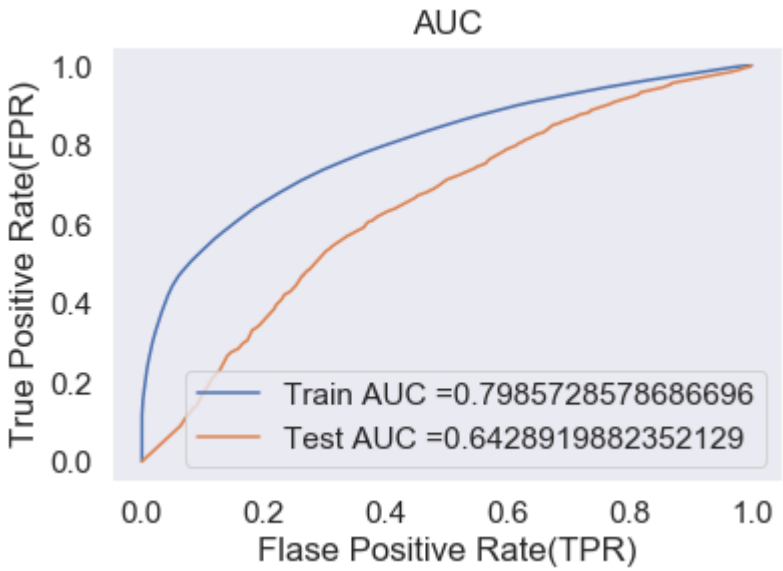


In [208]:

```
#max_depth = 9
#min_samples_split = 50
#cv auc = 0.64
#train auc = 0.79
```

In [209]:

```
from sklearn.metrics import roc_curve, auc

dt_tfidf_w2v = DecisionTreeClassifier(criterion='gini', max_depth=9, min_samples_split=50,c

dt_tfidf_w2v.fit(X_train_tfidf_w2v, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = dt_tfidf_w2v.predict_proba(X_train_tfidf_w2v)[:,1]
y_test_pred = dt_tfidf_w2v.predict_proba(X_test_tfidf_w2v)[:,1]

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("Flase Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

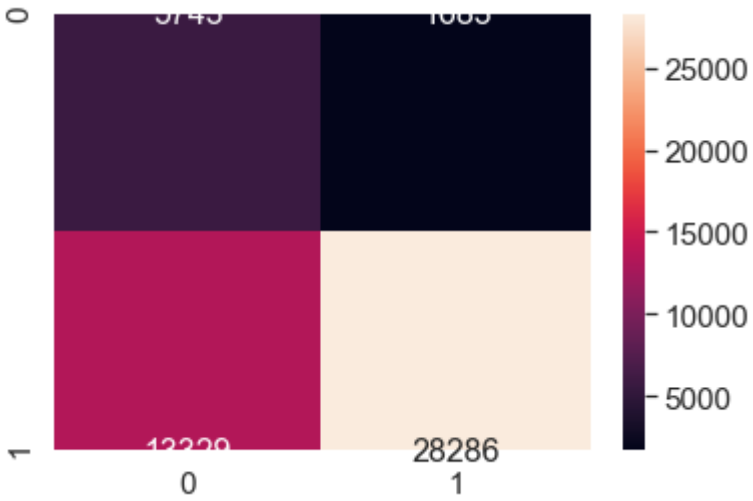


Confusion matrix

In [210]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.5256607004967333 for threshold 0.472
Train confusion matrix
[[ 5743  1683]
 [13329 28286]]
```



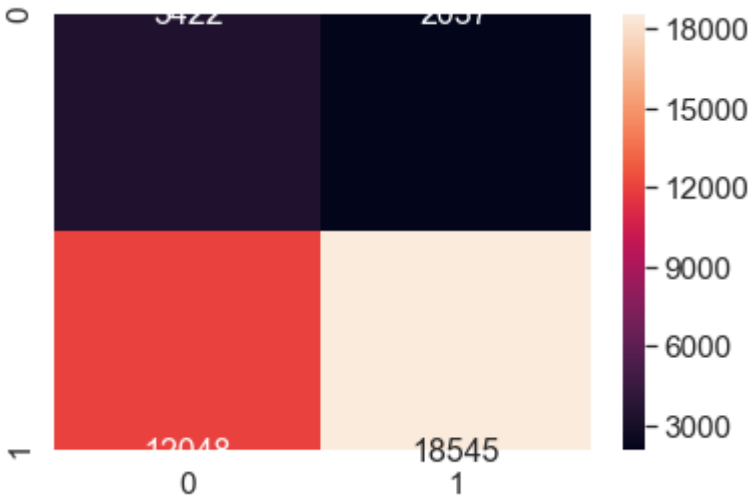
In [211]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)),r
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 0.3799895749396701 for threshold 0.494
Test confusion matrix
```

Out[211]:

```
array([[ 3422,  2037],
       [12048, 18545]], dtype=int64)
```



Evaluating Model performance

In [212]:

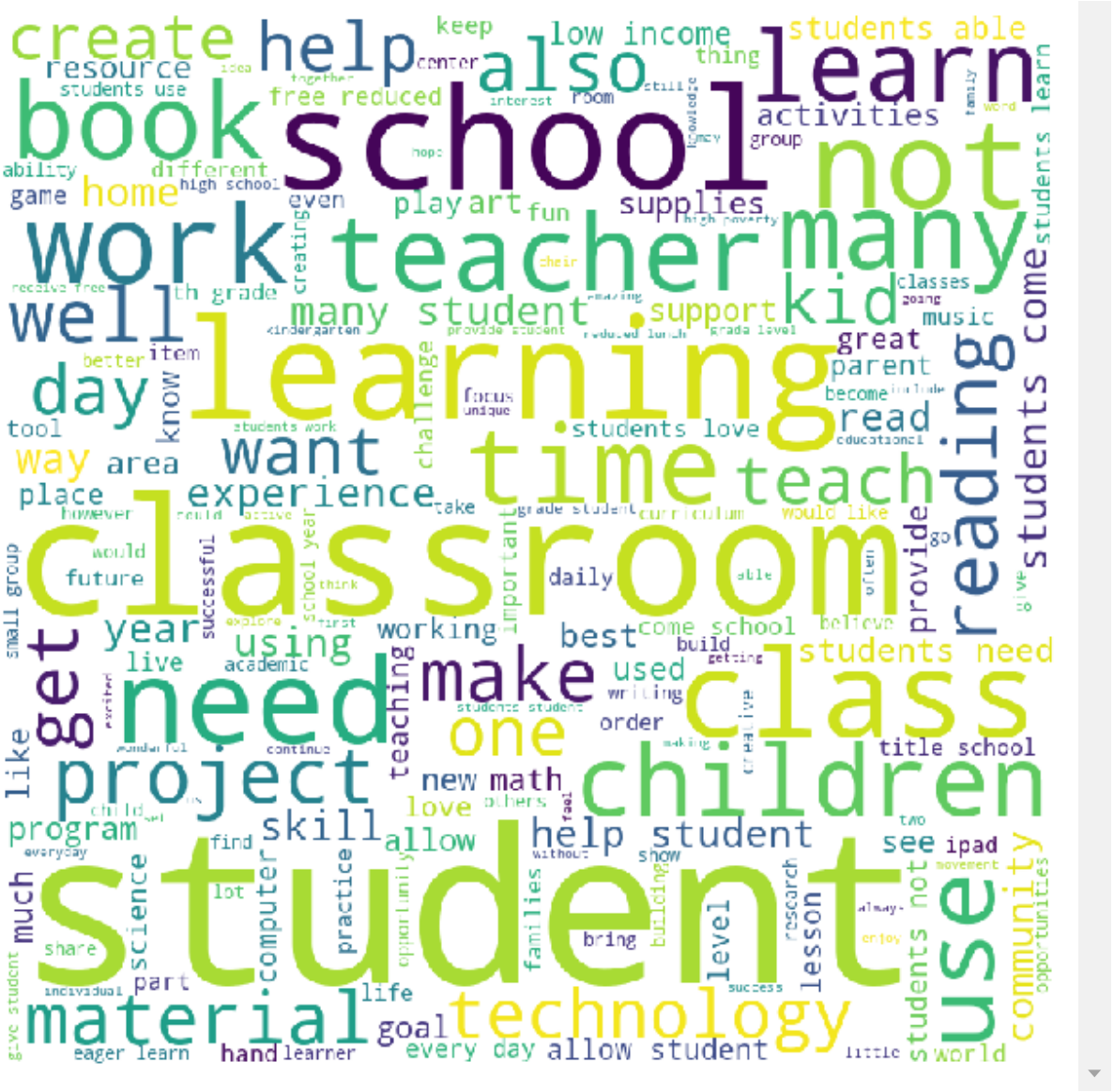
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = dt_tfidf_w2v.predict(X_test_tfidf_w2v)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 60.637%
Precision on test set: 0.901
Recall on test set: 0.602
F1-Score on test set: 0.722
```

Word Cloud on Test Essay-FPR

In [213]:

```
from wordcloud import WordCloud
essay_list_fpr = []
essay_list_words = []
length = (X_test_tfidf_w2v.shape[0])-1
essay_list=X_test['clean_essays']    #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        essay_list_fpr.append(essay_list.values[i]) #append all the essay at which model pr

for sentance in (essay_list_fpr):
    sent = decontracted(sentance)
    sent = sent.replace('nannan','' ) #removing nanna and digits from the essay
    sent = re.sub('[^a-z]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = sent.split(" ")
    essay_list_words.append(sent) #appending the cleaned sentences

stream = len(essay_list_words)-1

word = []
word_string = ''
len(essay_list_words[0])
for j in range (0,stream):
    for k in range (0,len(essay_list_words[j])-1):
        word.append(essay_list_words[j][k])          #extracting each word from each senten

word_string = ' '.join(e for e in word)             #making a single string out of all the

#https://www.geeksforgeeks.org/generating-word-cloud-python/
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(word_string)

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

plt.show()
```



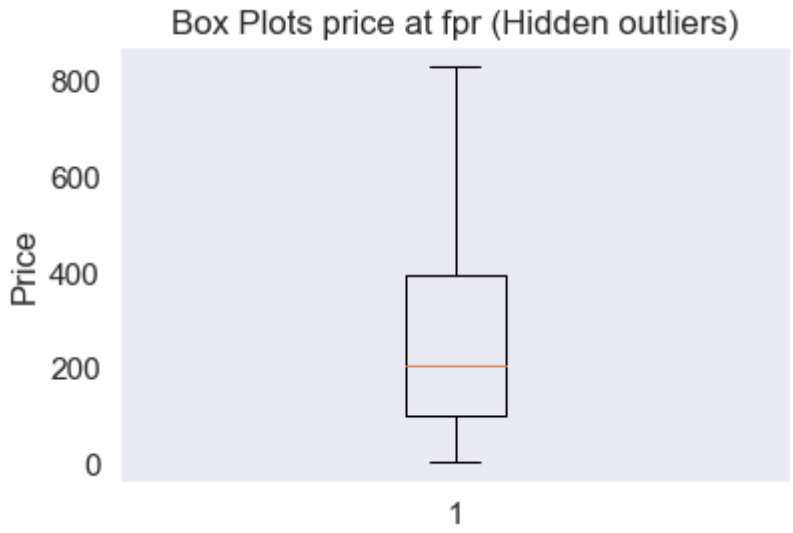
Box plot on price - FPR

In [214]:

```
price_fpr = []
length = len(X_test['price'])-1 #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        price_fpr.append(X_test['price'].values[i])

plt.boxplot([price_fpr])
plt.title('Box Plots price at fpr')
plt.xticks([1])
plt.ylabel('Price')
plt.grid()
plt.show()
```

In [215]:

```
price_fpr = []
length = len(X_test['price'])-1 #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        price_fpr.append(X_test['price'].values[i])

plt.boxplot([price_fpr],showfliers=False)
plt.title('Box Plots price at fpr (Hidden outliers)')
plt.xticks([1])
plt.ylabel('Price')
plt.grid()
plt.show()
```

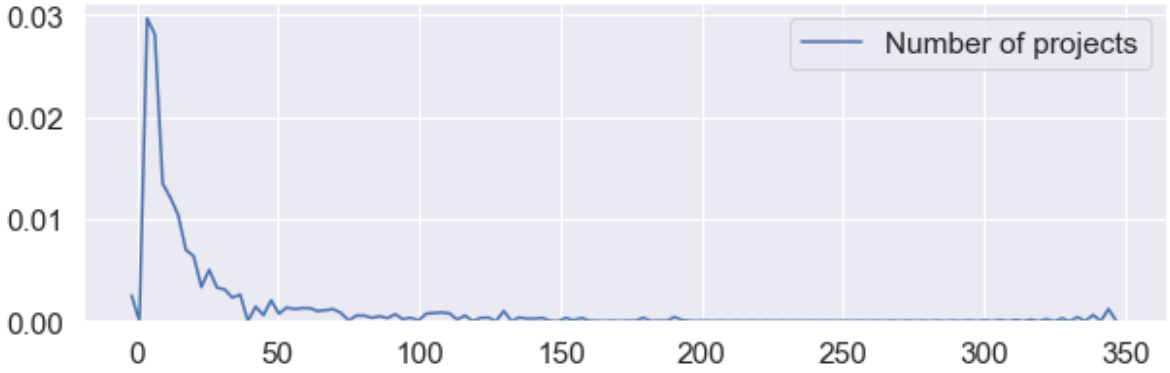


1. The model made 25 percent of wrong prediction when price are under the range of 100.
2. The model made 50 percent of wrong prediction when price are under the range of 210.
3. The model made 75 percent of wrong prediction when price are under the range of 400.
4. There are many outliers.

PDF (Number of previously posted asssignment by teachers) - FPR

In [216]:

```
pronum_fpr = []
length = len(X_test['teacher_number_of_previously_posted_projects'])-1 #listing out all the
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        pronum_fpr.append(X_test['teacher_number_of_previously_posted_projects'].values[i])

plt.figure(figsize=(10,3))
sns.kdeplot(pronum_fpr,label="Number of projects", bw=0.6)
plt.legend()
plt.show()
```



1. Mostly 1-5 projects have been previously posted by the teachers.
2. Few teaches have posted 10-40 projects

Set 5: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V) on 5000 best features

Selecting 5000 best features using Feature Importance

In [217]:

```
#https://datascience.stackexchange.com/questions/31406/tree-decisiontree-feature-importance
dt_tfidf.fit(X_train_tfidf,y_train)
train_features = X_train_tfidf[:,dt_tfidf.feature_importances_.argsort()[::-1][:5000]]
test_features =  X_test_tfidf[:,dt_tfidf.feature_importances_.argsort()[::-1][:5000]]
cv_features =  X_cv_tfidf[:,dt_tfidf.feature_importances_.argsort()[::-1][:5000]]
```

In [218]:

```
print('Final matrix')
print(train_features.shape, y_train.shape)
print(cv_features.shape, y_cv.shape)
print(test_features.shape, y_test.shape)
```

```
Final matrix
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
```

Hyperparameter tuning

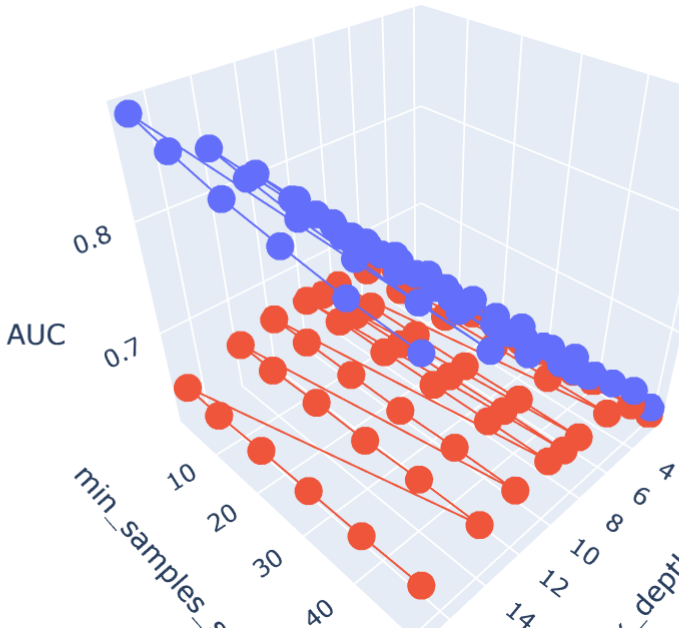
In [219]:

```
from sklearn.linear_model import SGDClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
import math

max_depth = [2,4,6,8,9,10,12,14,17]
min_samples_split = [2,10,20,30,40,50]


for i in (max_depth):
    for j in (min_samples_split):

        dt_tfidf = DecisionTreeClassifier(criterion='gini', max_depth=i, min_samples_split=

        dt_tfidf.fit(train_features, y_train)


        y_train_pred = dt_tfidf.predict_proba(train_features)[:,1]
        y_cv_pred = dt_tfidf.predict_proba(cv_features)[:,1]


        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
        # not the predicted outputs
        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

AUC 3D plot

In [220]:

```
#https://pythonprogramming.net/3d-scatter-plot-customizing/?completed=/matplotlib-3d-scatte

trace1 = go.Scatter3d(x=max_depth_val,y=min_samples_split_val,z=train_auc, name = 'train au
trace2 = go.Scatter3d(x=max_depth_val,y=min_samples_split_val,z=cv_auc, name = 'cv auc')
data = [trace1, trace2]

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

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



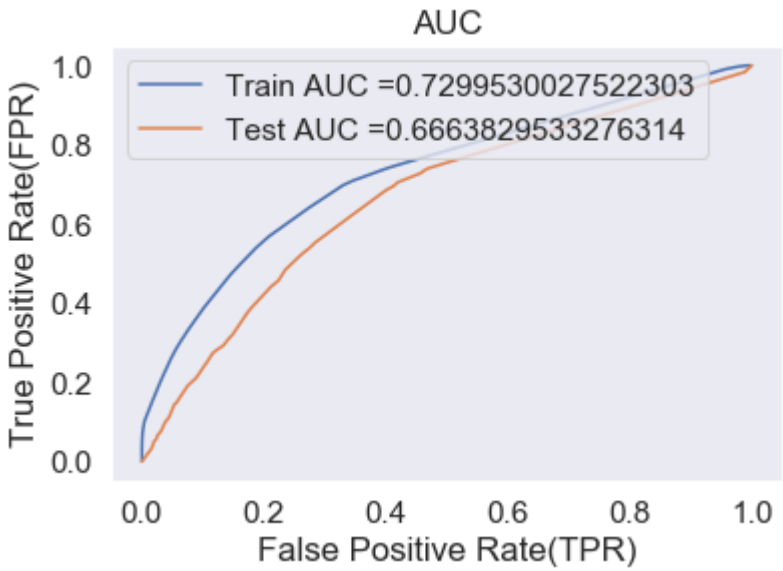
In [221]:

```
#max_depth = 8
#min_samples_split = 50
#cv auc = 0.66
#train auc = 0.73
```

Plotting Error plot

In [222]:

```
from sklearn.metrics import roc_curve, auc

dt_tfidf = DecisionTreeClassifier(criterion='gini', max_depth=8, min_samples_split=50,class
dt_tfidf.fit(train_features, y_train)

y_train_pred = dt_tfidf.predict_proba(train_features)[:,1]
y_test_pred = dt_tfidf.predict_proba(test_features)[:,1]

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

plt.plot(train_fpr, train_tpr, label="Train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(TPR)")
plt.ylabel("True Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```

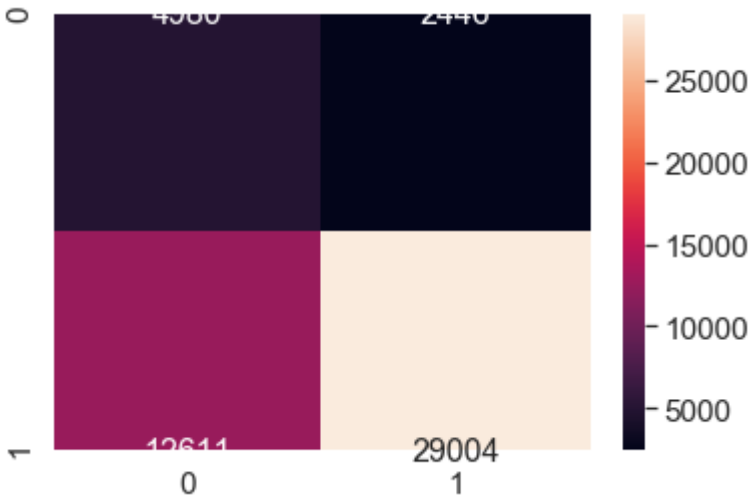


Confusion matrix

In [223]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
df_cmtr = pd.DataFrame(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmtr, annot=True,annot_kws={"size": 16}, fmt='g')
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
```

```
the maximum value of tpr*(1-fpr) 0.46739320614281343 for threshold 0.505
Train confusion matrix
[[ 4980  2446]
 [12611 29004]]
```



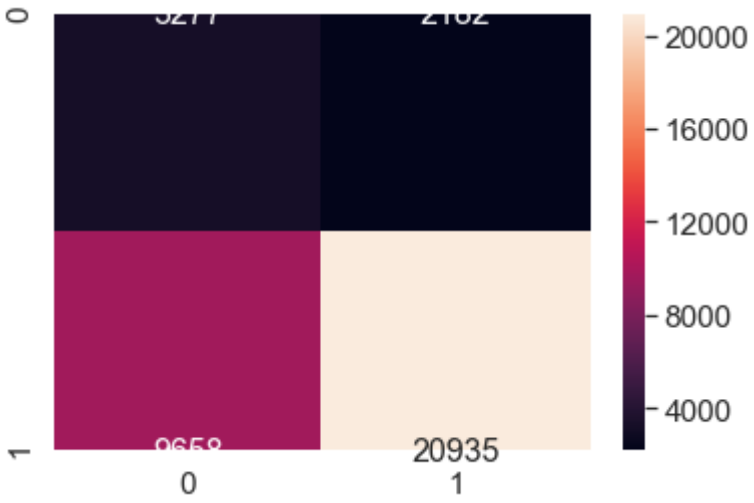
In [224]:

```
best_t = find_best_threshold(te_thresholds, test_fpr, test_tpr)
print("Test confusion matrix")
df_cmte = pd.DataFrame(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)),r
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cmte, annot=True,annot_kws={"size": 16}, fmt='g')
confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
```

```
the maximum value of tpr*(1-fpr) 0.4107846867691987 for threshold 0.472
Test confusion matrix
```

Out[224]:

```
array([[ 3277,  2182],
       [ 9658, 20935]], dtype=int64)
```



Evaluating model performance

In [225]:

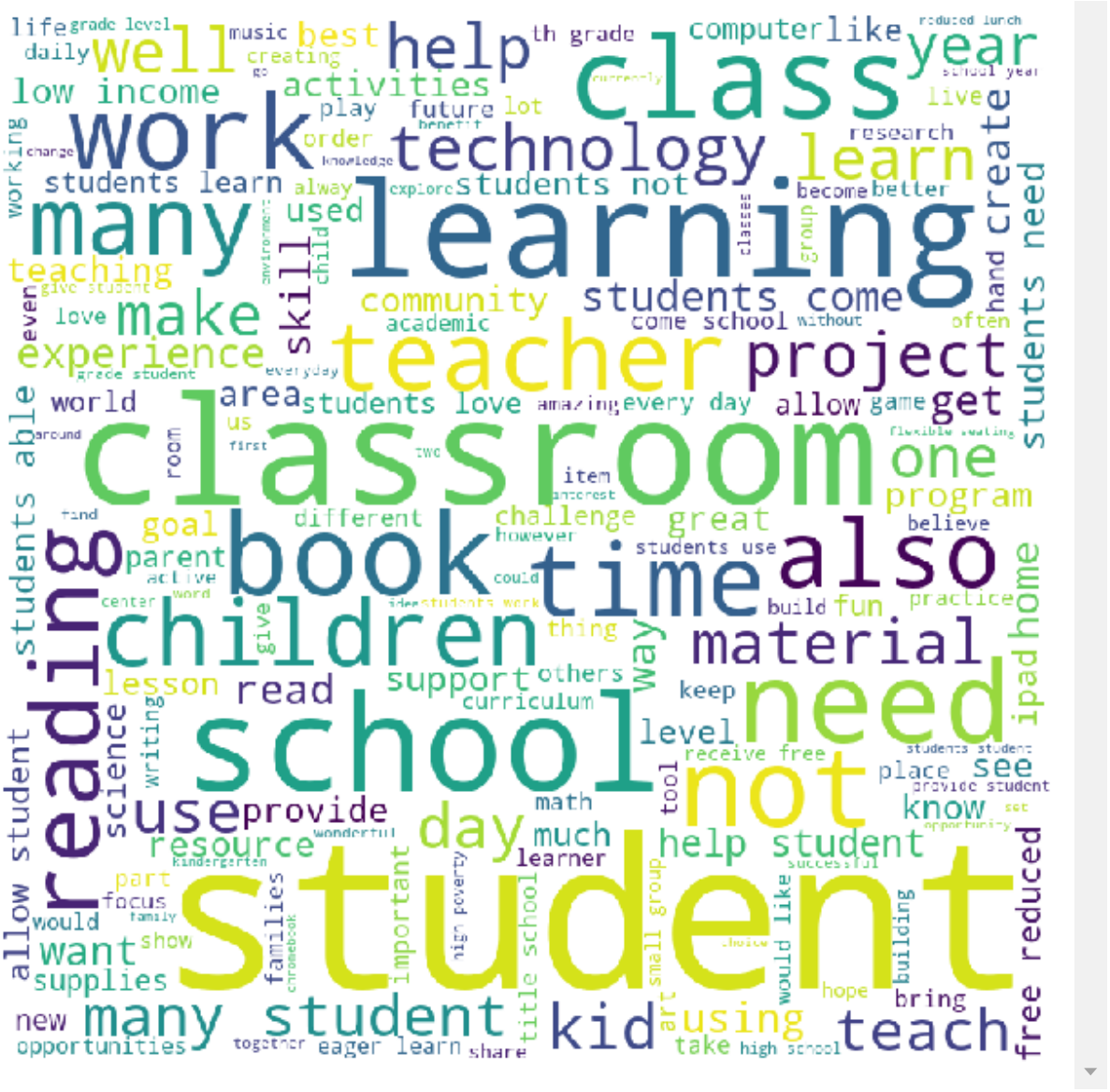
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = dt_tfidf.predict(test_features)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 67.139%
Precision on test set: 0.906
Recall on test set: 0.684
F1-Score on test set: 0.779
```

Word cloud on project essay -FPR

In [226]:

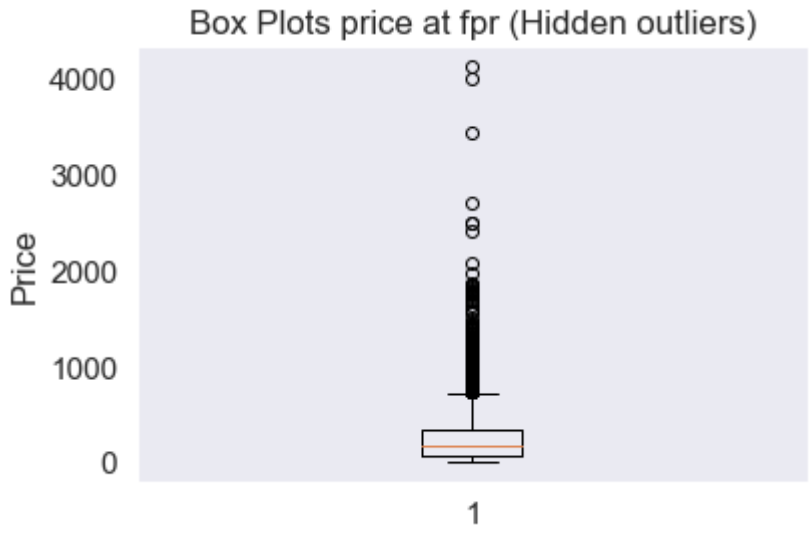
```
from wordcloud import WordCloud
essay_list_fpr = []
essay_list_words = []
length = (test_features.shape[0])-1
essay_list=X_test['clean_essays']    #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        essay_list_fpr.append(essay_list.values[i]) #append all the essay at which model pr

for sentance in (essay_list_fpr):
    sent = decontracted(sentance)
    sent = sent.replace('nannan','' ) #removing nanna and digits from the essay
    sent = re.sub('[^a-z]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = sent.split(" ")
    essay_list_words.append(sent) #appending the cleaned sentences

stream = len(essay_list_words)-1

word = []
word_string = ''
len(essay_list_words[0])
for j in range (0,stream):
    for k in range (0,len(essay_list_words[j])-1):
        word.append(essay_list_words[j][k])          #extracting each word from each senten

word_string = ' '.join(e for e in word)             #making a single string out of all the

#https://www.geeksforgeeks.org/generating-word-cloud-python/
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(word_string)

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

plt.show()
```



Box Plot on price -FPR

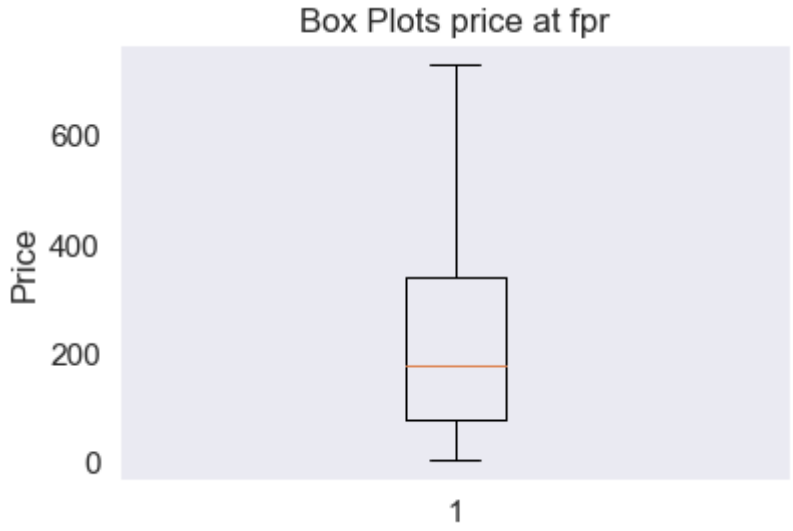
In [227]:

```
price_fpr = []
length = len(X_test['price'])-1 #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        price_fpr.append(X_test['price'].values[i])

plt.boxplot([price_fpr])
plt.title('Box Plots price at fpr (Hidden outliers)')
plt.xticks([1])
plt.ylabel('Price')
plt.grid()
plt.show()
```



In [228]:

```
price_fpr = []
length = len(X_test['price'])-1 #listing out all the clean eassy in test set
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        price_fpr.append(X_test['price'].values[i])

plt.boxplot([price_fpr],showfliers=False)
plt.title('Box Plots price at fpr')
plt.xticks([1])
plt.ylabel('Price')
plt.grid()
plt.show()
```

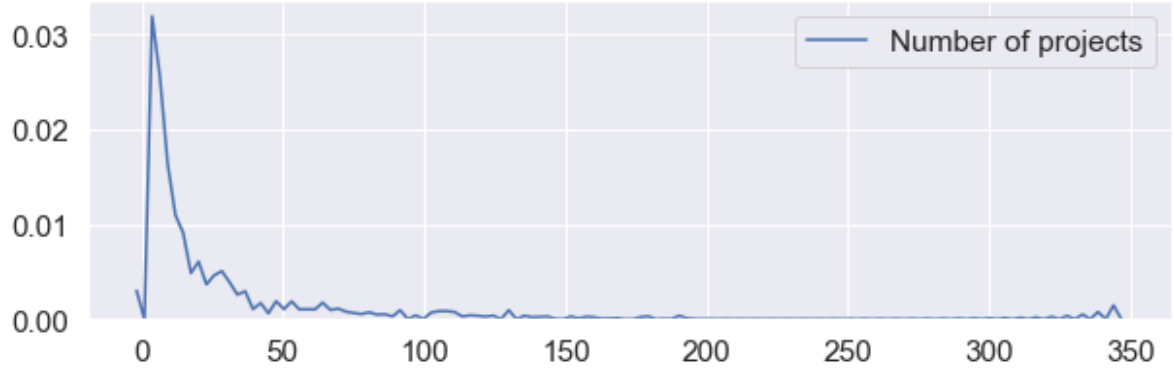


1. The model made 25 percent of wrong prediction when price are under the range of 100.
2. The model made 50 percent of wrong prediction when price are under the range of 210.
3. The model made 75 percent of wrong prediction when price are under the range of 400.
4. There are many outliers.

PDF on Number of previously posted projects by teacher -FPR

In [330]:

```
# pronum_fpr = []
length = len(X_test['teacher_number_of_previously_posted_projects'])-1 #listing out all the
for i in range (0,length):
    if(y_pred_new[i]==1 and y_test[i]==0):   #taking only those essay at which model predic
        pronum_fpr.append(X_test['teacher_number_of_previously_posted_projects'].values[i])

plt.figure(figsize=(10,3))
sns.kdeplot(pronum_fpr,label="Number of projects", bw=0.6)
plt.legend()
plt.show()
```



1. Mostly 1-5 projects have been previously posted by the teachers.
2. Few teaches have posted 10-40 projects

Visualizing Decision tree

In [327]:

```
print(dt_tfidf.feature_importances_)
#train_features = train_features.toarray()
train_features = pd.DataFrame(train_features) #converting to dataframe
sr = pd.Series(dt_tfidf.feature_importances_, index=train_features.columns) #extracting fea
```

```
[0.21847146 0.16407785 0.0734317  ... 0.         0.         0.        ]
```

In [328]:

```
#geeting feture names for the indices
features  = []
print(index)
for i in index:
    features.append(features_list[i])

features[0:10]
```

```
[   0    1    2 ... 3333 3334 2499]
```

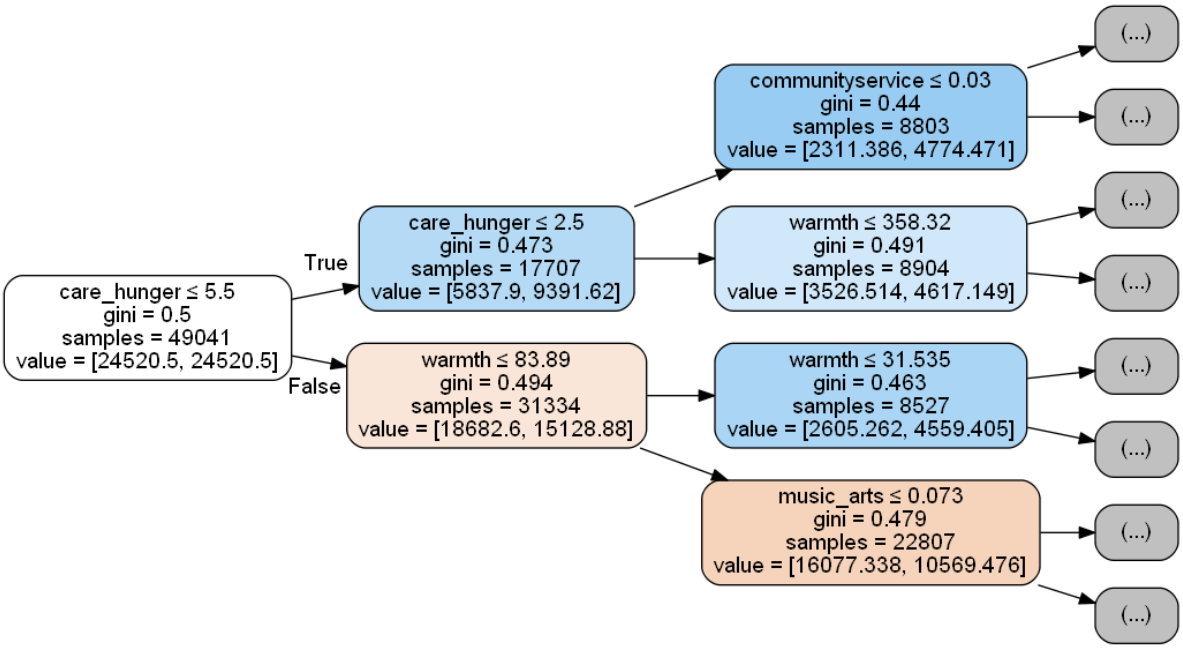
Out[328]:

```
['warmth',
 'care_hunger',
 'history_civics',
 'music_arts',
 'appliedlearning',
 'specialneeds',
 'health_sports',
 'math_science',
 'literacy_language',
 'economics']
```

In [329]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html
import warnings
warnings.filterwarnings("ignore")
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
string_data = StringIO()
export_graphviz(dt_tfidf, max_depth=2, out_file=string_data, filled=True, rounded=True, spe
graph = pydotplus.graph_from_dot_data(string_data.getvalue())
Image(graph.create_png())
```

Out[329]:



Conclusion:

1. All the model has more than 56% accuracy on test set.
2. Tfidf model has performed better than other models.

In [331]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "max depth","min sample spit", "Train AUC","Test AUC","F1 Sc

x.add_row(["BoW (set 1)",8,50, 0.73, 0.67,0.73,"61.9%"])
x.add_row(["TFIDF (set 2)",9,50, 0.74, 0.66,0.75,"64.%"])
x.add_row(["AVG W2V (set 3)",9,50, 0.80, 0.63,0.71,"60.03%"])
x.add_row(["TFIDF W2V (set 4)",9,50, 0.79, 0.64, 0.72,"60.6%"])
x.add_row(["TFIDF (set 5)",9,50, 0.72, 0.66, 0.77,"67.13%"])

print(x)
```

```
+------------------+-----------+----------------+-----------+----------+--
--------+----------------------+
|     Vectorizer    | max depth | min sample spit | Train AUC | Test AUC | F
1 Score | Accuracy on test set |
+------------------+-----------+----------------+-----------+----------+--
--------+----------------------+
|    BoW (set 1)    |     8     |        50       |    0.73   |   0.67   |
0.73   |        61.9%         |
|   TFIDF (set 2)   |     9     |        50       |    0.74   |   0.66   |
0.75   |         64.%         |
|  AVG W2V (set 3)  |     9     |        50       |    0.8    |   0.63   |
0.71   |        60.03%        |
| TFIDF W2V (set 4) |     9     |        50       |    0.79   |   0.64   |
0.72   |        60.6%         |
|   TFIDF (set 5)   |     9     |        50       |    0.72   |   0.66   |
0.77   |        67.13%        |
+------------------+-----------+----------------+-----------+----------+--
--------+----------------------+
```