DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*

Feature	Description
`project_essay_4`	Fourth application essay
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

* See the section **Notes on the Essay Data** for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
`id`	A `project_id` value from the `train.csv` file. **Example:** `p036502`
`description`	Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25`
`quantity`	Quantity of the resource required. **Example:** `3`
`price`	Price of the resource required. **Example:** `9.95`

Note: Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
`project_is_approved`	A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved.

Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_3:__ "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

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
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
from sklearn.model_selection import train_test_split
import sklearn.model_selection as model_selection
```

1.1 Reading Data

In [2]:

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

In [3]:

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

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['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']
```

In [4]:

```
project_data['project_is_approved'].value_counts()
```

Out[4]:

```
1    92706
0    16542
Name: project_is_approved, dtype: int64
```

In [5]:

```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[5]:

	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

1.2 preprocessing of `project_subject_categories`

In [6]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# 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", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    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]))
```

1.3 preprocessing of `project_subject_subcategories`

In [7]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039

# 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", "Care & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    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]))
```

1.3 Text preprocessing

In [8]:

```
# 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 [9]:

```
project_data.head(2)
```

Out[9]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_cate
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades P
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grade

In [10]:

```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

```
My students are English learners that are working on English as their second or third languages. W
e are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of langua
ge to our school. \r\n\r\n We have over 24 languages represented in our English Learner program wi
th students at every level of mastery.  We also have over 40 countries represented with the
families within our school.  Each student brings a wealth of knowledge and experiences to us that
open our eyes to new cultures, beliefs, and respect.\"The limits of your language are the limits o
f your world.\" Ludwig Wittgenstein  Our English learner's have a strong support system at home th
```

```
r your world.\"-Ludwig Wittgenstein  Our English learner's have a strong support system at home th
at begs for more resources.  Many times our parents are learning to read and speak English along s
ide of their children.  Sometimes this creates barriers for parents to be able to help their child
learn phonetics, letter recognition, and other reading skills.\r\n\r\nBy providing these dvd's and
players, students are able to continue their mastery of the English language even if no one at hom
e is able to assist.  All families with students within the Level 1 proficiency status, will be a
offered to be a part of this program.  These educational videos will be specially chosen by the En
glish Learner Teacher and will be sent home regularly to watch.  The videos are to help the child
develop early reading skills.\r\n\r\nParents that do not have access to a dvd player will have the
opportunity to check out a dvd player to use for the year.  The plan is to use these videos and ed
ucational dvd's for the years to come for other EL students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this year all love learning, at l
east most of the time. At our school, 97.3% of the students receive free or reduced price lunch. O
f the 560 students, 97.3% are minority students. \r\nThe school has a vibrant community that loves
to get together and celebrate. Around Halloween there is a whole school parade to show off the bea
utiful costumes that students wear. On Cinco de Mayo we put on a big festival with crafts made by
the students, dances, and games. At the end of the year the school hosts a carnival to celebrate t
he hard work put in during the school year, with a dunk tank being the most popular activity.My st
udents will use these five brightly colored Hokki stools in place of regular, stationary, 4-legged
chairs. As I will only have a total of ten in the classroom and not enough for each student to hav
e an individual one, they will be used in a variety of ways. During independent reading time they
will be used as special chairs students will each use on occasion. I will utilize them in place of
chairs at my small group tables during math and reading times. The rest of the day they will be us
ed by the students who need the highest amount of movement in their life in order to stay focused
on school.\r\n\r\nWhenever asked what the classroom is missing, my students always say more Hokki
Stools. They can't get their fill of the 5 stools we already have. When the students are sitting i
n group with me on the Hokki Stools, they are always moving, but at the same time doing their
work. Anytime the students get to pick where they can sit, the Hokki Stools are the first to be ta
ken. There are always students who head over to the kidney table to get one of the stools who are
disappointed as there are not enough of them. \r\n\r\nWe ask a lot of students to sit for 7 hours
a day. The Hokki stools will be a compromise that allow my students to do desk work and move at th
e same time. These stools will help students to meet their 60 minutes a day of movement by
allowing them to activate their core muscles for balance while they sit. For many of my students,
these chairs will take away the barrier that exists in schools for a child who can't sit
still.nannan
==================================================
How do you remember your days of school? Was it in a sterile environment with plain walls, rows of
desks, and a teacher in front of the room? A typical day in our room is nothing like that. I work
hard to create a warm inviting themed room for my students look forward to coming to each day.\r\n
\r\nMy class is made up of 28 wonderfully unique boys and girls of mixed races in
Arkansas.\r\nThey attend a Title I school, which means there is a high enough percentage of free a
nd reduced-price lunch to qualify. Our school is an \"open classroom\" concept, which is very uniq
ue as there are no walls separating the classrooms. These 9 and 10 year-old students are very eage
r learners; they are like sponges, absorbing all the information and experiences and keep on wanti
ng more.With these resources such as the comfy red throw pillows and the whimsical nautical hangin
g decor and the blue fish nets, I will be able to help create the mood in our classroom setting to
be one of a themed nautical environment. Creating a classroom environment is very important in the
success in each and every child's education. The nautical photo props will be used with each child
as they step foot into our classroom for the first time on Meet the Teacher evening. I'll take pic
tures of each child with them, have them developed, and then hung in our classroom ready for their
first day of 4th grade.  This kind gesture will set the tone before even the first day of school!
The nautical thank you cards will be used throughout the year by the students as they create thank
you cards to their team groups.\r\n\r\nYour generous donations will help me to help make our
classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of
my own pocket on resources to get our classroom ready. Please consider helping with this project t
o make our new school year a very successful one. Thank you!nannan
==================================================
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir hardest working past their limitations. \r\n\r\nThe materials we have are the ones I seek out
for my students. I teach in a Title I school where most of the students receive free or reduced pr
ice lunch.  Despite their disabilities and limitations, my students love coming to school and come
eager to learn and explore.Have you ever felt like you had ants in your pants and you needed to gr
oove and move as you were in a meeting? This is how my kids feel all the time. The want to be able
to move as they learn or so they say.Wobble chairs are the answer and I love then because they dev
elop their core, which enhances gross motor and in Turn fine motor skills. \r\nThey also want to l
earn through games, my kids don't want to sit and do worksheets. They want to learn to count by ju
mping and playing. Physical engagement is the key to our success. The number toss and color and sh
ape mats can make that happen. My students will forget they are doing work and just have the fun a
6 year old deserves.nannan
==================================================
The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates. The grea
t teacher inspires. -William A. Ward\r\n\r\nMy school has 803 students which is makeup is 97.6% Af
rican-American, making up the largest segment of the student body. A typical school in Dallas is m
ade up of 23.2% African-American students. Most of the students are on free or reduced lunch. We a
ren't receiving doctors, lawyers, or engineers children from rich backgrounds or neighborhoods. As
```

```
an educator I am inspiring minds of young children and we focus not only on academics but one smar
t, effective, efficient, and disciplined students with good character.In our classroom we can util
ize the Bluetooth for swift transitions during class. I use a speaker which doesn't amplify the so
und enough to receive the message. Due to the volume of my speaker my students can't hear videos o
r books clearly and it isn't making the lessons as meaningful. But with the bluetooth speaker my s
tudents will be able to hear and I can stop, pause and replay it at any time.\r\nThe cart will all
ow me to have more room for storage of things that are needed for the day and has an extra part to
it I can use.  The table top chart has all of the letter, words and pictures for students to learn
about different letters and it is more accessible.nannan
==================================================
```

In [11]:

```python
# 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 [12]:

```python
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir hardest working past their limitations. \r\n\r\nThe materials we have are the ones I seek out
for my students. I teach in a Title I school where most of the students receive free or reduced pr
ice lunch.  Despite their disabilities and limitations, my students love coming to school and come
eager to learn and explore.Have you ever felt like you had ants in your pants and you needed to gr
oove and move as you were in a meeting? This is how my kids feel all the time. The want to be able
to move as they learn or so they say.Wobble chairs are the answer and I love then because they dev
elop their core, which enhances gross motor and in Turn fine motor skills. \r\nThey also want to l
earn through games, my kids do not want to sit and do worksheets. They want to learn to count by j
umping and playing. Physical engagement is the key to our success. The number toss and color and s
hape mats can make that happen. My students will forget they are doing work and just have the fun
a 6 year old deserves.nannan
==================================================
```

In [13]:

```python
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir hardest working past their limitations.     The materials we have are the ones I seek out for
my students. I teach in a Title I school where most of the students receive free or reduced price
lunch.  Despite their disabilities and limitations, my students love coming to school and come eag
er to learn and explore.Have you ever felt like you had ants in your pants and you needed to groov
e and move as you were in a meeting? This is how my kids feel all the time. The want to be able to
move as they learn or so they say.Wobble chairs are the answer and I love then because they develo
p their core, which enhances gross motor and in Turn fine motor skills.   They also want to learn t
hrough games, my kids do not want to sit and do worksheets. They want to learn to count by jumping
and playing. Physical engagement is the key to our success. The number toss and color and shape ma
ts can make that happen. My students will forget they are doing work and just have the fun a 6 yea
```

```
r old deserves.nannan
```

In [14]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays cognitiv
e delays gross fine motor delays to autism They are eager beavers and always strive to work their
hardest working past their limitations The materials we have are the ones I seek out for my studen
ts I teach in a Title I school where most of the students receive free or reduced price lunch
Despite their disabilities and limitations my students love coming to school and come eager to lea
rn and explore Have you ever felt like you had ants in your pants and you needed to groove and mov
e as you were in a meeting This is how my kids feel all the time The want to be able to move as th
ey learn or so they say Wobble chairs are the answer and I love then because they develop their co
re which enhances gross motor and in Turn fine motor skills They also want to learn through games
my kids do not want to sit and do worksheets They want to learn to count by jumping and playing Ph
ysical engagement is the key to our success The number toss and color and shape mats can make that
happen My students will forget they are doing work and just have the fun a 6 year old deserves nan
nan
```

In [15]:

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

In [16]:

```
# Combining all the above stundents
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('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    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 [01:15<00:00, 1450.68it/s]
```

In [17]:

```
# after preprocesing
project_data['processed_essay'] = preprocessed_essays;
project_data.drop(['essay'], axis=1, inplace=True)
preprocessed_essays[20000]
```

Out[17]:

```
'my kindergarten students varied disabilities ranging speech language delays cognitive delays gros
s fine motor delays autism they eager beavers always strive work hardest working past limitations
the materials ones i seek students i teach title i school students receive free reduced price lunc
h despite disabilities limitations students love coming school come eager learn explore have ever
felt like ants pants needed groove move meeting this kids feel time the want able move learn say w
obble chairs answer i love develop core enhances gross motor turn fine motor skills they also want
learn games kids not want sit worksheets they want learn count jumping playing physical engagement
key success the number toss color shape mats make happen my students forget work fun 6 year old de
serves nannan'
```

1.4 Preprocessing of `project_title`

In [18]:

```
processed_titles = [];
for title in tqdm(project_data['project_title'].values):
    sent = decontracted(title)
    sent = re.sub('\S*\d\S*', '', sent);
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    processed_titles.append(sent.strip())
```

```
100%|
████████████████████████████████████████████████████████████████████████████████████
109248/109248 [00:02<00:00, 48732.23it/s]
```

In [19]:

```
project_data.drop(['project_title'], axis=1, inplace=True)
project_data['processed_titles'] = processed_titles

#testing after preprocessing project_title column
print(processed_titles[3])

print(processed_titles[40]);

print(processed_titles[500]);

print(processed_titles[4000]);

project_data.columns
```

```
Techie Kindergarteners
Leveling Books in a Multi Age Class
Classroom Chromebooks for College Bound Seniors
Inspire Summer Reading
```

Out[19]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'processed_essay',
       'processed_titles'],
      dtype='object')
```

Preprocessing of project_grade_category

In [20]:

```
print(project_data['project_grade_category'][1])
print(project_data['project_grade_category'][223])
print(project_data['project_grade_category'][134])
```

```
Grades 6-8
Grades PreK-2
Grades PreK-2
```

In [21]:

```
processed_grades = [];
for grades in project_data['project_grade_category']:
    grades = grades.replace('-', '');
    processed_grades.append(grades)
```

In [22]:

```
print(processed_grades[1])
print(processed_grades[223])
print(processed_grades[134])

project_data.drop(['project_grade_category'], axis=1, inplace=True)
project_data['processed_grades'] = processed_grades
```

```
Grades 68
Grades PreK2
Grades PreK2
```

Preprocessing of teacher_prefix

In [23]:

```
print(project_data['teacher_prefix'][2]);
print(project_data['teacher_prefix'][234]);
print(project_data['teacher_prefix'][425]);
```

```
Ms.
Ms.
Ms.
```

In [24]:

```
preprocessed_teacher_prefix = [];
for prefix in project_data['teacher_prefix']:
    prefix = str(prefix).replace('.', '');
    preprocessed_teacher_prefix.append(prefix);
```

In [25]:

```
project_data.drop(['teacher_prefix'], axis=1, inplace=True)
project_data['processed_teacher_prefix'] = preprocessed_teacher_prefix

print(preprocessed_teacher_prefix[321])
print(preprocessed_teacher_prefix[310])
```

```
Mrs
Ms
```

1.5 Preparing data for models

In [26]:

In [26]:

```
project_data.columns
```

Out[26]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'school_state',
       'project_submitted_datetime', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'processed_essay',
       'processed_titles', 'processed_grades', 'processed_teacher_prefix'],
      dtype='object')
```

we are going to consider

```
- school_state : categorical data
- clean_categories : categorical data
- clean_subcategories : categorical data
- project_grade_category : categorical data
- teacher_prefix : categorical data

- project_title : text data
- text : text data
- project_resource_summary: text data (optinal)

- quantity : numerical (optinal)
- teacher_number_of_previously_posted_projects : numerical
- price : numerical
```

Merging Price from resource_Data

In [27]:

```
price = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index();
project_data = pd.merge(project_data, price, on='id', how='left');
project_data.columns
```

Out[27]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'school_state',
       'project_submitted_datetime', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'processed_essay',
       'processed_titles', 'processed_grades', 'processed_teacher_prefix',
       'price', 'quantity'],
      dtype='object')
```

Importing Glove, pretrained model, which we use for word2vec

In [28]:

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

In [ ]:

```

```

Assignment 8: DT

Assignment 8: DT

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

    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
    - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
    - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

2. **Hyper paramter tuning (best `depth` in range [1, 5, 10, 50, 100, 500, 100], and the best `min_samples_split` in range [5, 10, 100, 500])**

    - Find the best hyper parameter which will give the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Graphviz**

    - Visualize your decision tree with Graphviz. It helps you to understand how a decision is being made, given a new vector.
    - Since feature names are not obtained from word2vec related models, visualize only BOW & TFIDF decision trees using Graphviz
    - Make sure to print the words in each node of the decision tree instead of printing its index.
    - Just for visualization purpose, limit max_depth to 2 or 3 and either embed the generated images of graphviz in your notebook, or directly upload them as .png files.

4. **Representation of results**

    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points
    - Once after you plot the confusion matrix with the test data, get all the `false positive data points`
        - Plot the WordCloud WordCloud
        - Plot the box plot with the `price` of these `false positive data points`
        - Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points`

5. **[Task-2]**

    - Select 5k best features from features of Set 2 using`feature_importances_`, discard all the other remaining features and then apply any of the model of you choice i.e. (Dession tree, Logistic Regression, Linear SVM), you need to do hyperparameter tuning corresponding to the model you selected and procedure in step 2 and step 3

6. **Conclusion**

    - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakage, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

2. Decision Tree

2.1 Splitting data into Train and cross validation(or test): Stratified Sampling

In [29]:

```
#splitting project_data into x and y, y=project_is_approved.
```

```
#fetching all the columns except project_is_approved.
cols_to_select = [col for col in project_data.columns if col != 'project_is_approved'];
X = project_data[cols_to_select]
print(X.columns)
y = project_data['project_is_approved'];
print(y.shape)
```

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'school_state',
       'project_submitted_datetime', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'processed_essay', 'processed_titles',
       'processed_grades', 'processed_teacher_prefix', 'price', 'quantity'],
      dtype='object')
(109248,)
```

In [30]:

```
#splitting project_data into train and test and CV data.
X_1, X_test, y_1, y_test = model_selection.train_test_split(X, y, test_size=0.3, random_state=1, st
ratify=y)
X_train, X_cv, y_train, y_cv = model_selection.train_test_split(X_1, y_1, test_size=0.3, random_sta
te=1, stratify=y_1);

print('shape of train data ', X_train.shape);
print('shape of test data ', X_test.shape);
print('shape of cross validation data ', X_cv.shape)
```

```
shape of train data  (53531, 19)
shape of test data  (32775, 19)
shape of cross validation data  (22942, 19)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

Vectorizing Categorical features

In [31]:

```
#vectorizing school_state
from sklearn.feature_extraction.text import CountVectorizer


#creating dictionary for school_state as state as keys along with no. of projects from that state
as values.
school_state_dict = dict(X_train['school_state'].value_counts());

#configuring CountVectorizer for school state, in which vocabulary will be name of states.
vectorizer = CountVectorizer(vocabulary=list(school_state_dict.keys()), lowercase=False, binary=Tr
ue);

#applying vectorizer on school_state column to obtain numerical value for each state.
vectorizer.fit(X_train['school_state'].values);

school_state_vector = vectorizer.transform(X_train['school_state'].values);
test_school_state_vector = vectorizer.transform(X_test['school_state'].values);
cv_school_state_vector = vectorizer.transform(X_cv['school_state'].values);

print('shape of matrix after one hot encoding of school_state for train data ',
school_state_vector.shape);
print('shape of matrix after one hot encoding of school_state for test data ',
test_school_state_vector.shape);
print('shape of matrix after one hot encoding of school_state for cv data ',
cv_school_state_vector.shape);

features_name_list = vectorizer.get_feature_names();
```

```
shape of matrix after one hot encoding of school_state for train data  (53531, 51)
shape of matrix after one hot encoding of school_state for test data  (32775, 51)
shape of matrix after one hot encoding of school_state for cv data  (22942, 51)
```

In [32]:

```
#vectorizing categories

vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
);

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

categories_vector = vectorizer.transform(X_train['clean_categories'].values);
test_categories_vector = vectorizer.transform(X_test['clean_categories'].values);
cv_categories_vector = vectorizer.transform(X_cv['clean_categories'].values);

print('shape of matrix after one hot encoding of clean_categories for train data',
categories_vector.shape)
print('shape of matrix after one hot encoding of clean_categories for test data',
test_categories_vector.shape)
print('shape of matrix after one hot encoding of clean_categories for cv data',
cv_categories_vector.shape)

features_name_list.extend( vectorizer.get_feature_names());
```

```
shape of matrix after one hot encoding of clean_categories for train data (53531, 9)
shape of matrix after one hot encoding of clean_categories for test data (32775, 9)
shape of matrix after one hot encoding of clean_categories for cv data (22942, 9)
```

In [33]:

```
#vectorizing subcategories

vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True);

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

subcategories_vector = vectorizer.transform(X_train['clean_subcategories'].values);
test_subcategories_vector = vectorizer.transform(X_test['clean_subcategories'].values);
cv_subcategories_vector = vectorizer.transform(X_cv['clean_subcategories'].values);

print('shape of matrix after one hot encoding of clean_subcategories for train data',
subcategories_vector.shape)
print('shape of matrix after one hot encoding of clean_subcategories for test data',
test_subcategories_vector.shape)
print('shape of matrix after one hot encoding of clean_subcategories for cv data',
cv_subcategories_vector.shape)

features_name_list.extend( vectorizer.get_feature_names());
```

```
shape of matrix after one hot encoding of clean_subcategories for train data (53531, 30)
shape of matrix after one hot encoding of clean_subcategories for test data (32775, 30)
shape of matrix after one hot encoding of clean_subcategories for cv data (22942, 30)
```

In [34]:

```
#vectorizing project_grade_category

grade_dict = dict(X_train['processed_grades'].value_counts());

vectorizer = CountVectorizer(vocabulary=list(grade_dict.keys()), lowercase=False, binary=True);

vectorizer.fit(X_train['processed_grades'].values);

grade_vector = vectorizer.transform(X_train['processed_grades'].values);
test_grade_vector = vectorizer.transform(X_test['processed_grades'].values);
cv_grade_vector = vectorizer.transform(X_cv['processed_grades'].values);

print('shape of matrix after one hot encoding of grade_category for train data', grade_vector.shap
e)
print('shape of matrix after one hot encoding of grade_category for test data', test_grade_vector.
shape)
print('shape of matrix after one hot encoding of grade_category for cv data', cv_grade_vector.shap
e)
```

```
features_name_list.extend( vectorizer.get_feature_names());
```

```
shape of matrix after one hot encoding of grade_category for train data (53531, 4)
shape of matrix after one hot encoding of grade_category for test data (32775, 4)
shape of matrix after one hot encoding of grade_category for cv data (22942, 4)
```

In [35]:

```
#vectorizing teacher_prefix

teacher_prefix_dict = dict(X_train['processed_teacher_prefix'].value_counts());

vectorizer = CountVectorizer(vocabulary=list(teacher_prefix_dict.keys()), lowercase=False, binary=
True);

vectorizer.fit(X_train['processed_teacher_prefix'].values.astype('U'));

teacher_prefix_vector = vectorizer.transform(X_train['processed_teacher_prefix'].values.astype('U')
);
test_teacher_prefix_vector = vectorizer.transform(X_test['processed_teacher_prefix'].values.astype(
'U'));
cv_teacher_prefix_vector = vectorizer.transform(X_cv['processed_teacher_prefix'].values.astype('U')
);

print('shape of matrix after one hot encoding of teacher_prefix for train data',
teacher_prefix_vector.shape)
print('shape of matrix after one hot encoding of teacher_prefix for test data',
test_teacher_prefix_vector.shape)
print('shape of matrix after one hot encoding of teacher_prefix for cv data',
cv_teacher_prefix_vector.shape)

features_name_list.extend( vectorizer.get_feature_names());
```

```
shape of matrix after one hot encoding of teacher_prefix for train data (53531, 6)
shape of matrix after one hot encoding of teacher_prefix for test data (32775, 6)
shape of matrix after one hot encoding of teacher_prefix for cv data (22942, 6)
```

Encoding Numerical data

In [36]:

```
#vectorizing price

from sklearn.preprocessing import StandardScaler
price_normalizer = StandardScaler()
#configuring StandarScaler to obtain the mean and variance.
price_normalizer.fit(X_train['price'].values.reshape(-1, 1));

# Now standardize the data with maen and variance obtained above.
price_standardized = price_normalizer.transform(X_train['price'].values.reshape(-1, 1))
test_price_standardized = price_normalizer.transform(X_test['price'].values.reshape(-1, 1))
cv_price_standardized = price_normalizer.transform(X_cv['price'].values.reshape(-1, 1))

features_name_list.append('price');
```

In [37]:

```
#vectorizing teacher_number_of_previously_posted_projects

teacher_normalizer = StandardScaler();

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

teacher_number_standardized =
teacher_normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape
(-1,1));

test_teacher_number_standardized =
teacher_normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape
(-1,1));

cv_teacher_number_standardized =
teacher_normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-
1,1));

features_name_list.append('teacher number of previously posted projects');
```

In [38]:

```
#vectorizing quantity:

quantity_normalizer = StandardScaler();

quantity_normalizer.fit(X_train['quantity'].values.reshape(-1, 1));

quantity_standardized = quantity_normalizer.transform(X_train['quantity'].values.reshape(-1, 1))

test_quantity_standardized = quantity_normalizer.transform(X_test['quantity'].values.reshape(-1, 1)
)

cv_quantity_standardized = quantity_normalizer.transform(X_cv['quantity'].values.reshape(-1, 1))

features_name_list.append('quantity');
```

2.3 Make Data Model Ready: encoding eassay, and project_title

Vectorizing using BOW on train data

In [39]:

```
#vectorizing essay

#configure CountVectorizer with word to occur in at least 10 documents.
vectorizer = CountVectorizer(min_df=10, ngram_range=(1,2), max_features=5000);

vectorizer.fit(X_train['processed_essay']);

#transforming essay into vector
essay_bow = vectorizer.transform(X_train['processed_essay']);
cv_essay_bow = vectorizer.transform(X_cv['processed_essay']);
test_essay_bow = vectorizer.transform(X_test['processed_essay']);

print('Shape of matrix after one hot encoding for train data: ', essay_bow.shape);
print('Shape of matrix after one hot encoding for test data: ', test_essay_bow.shape);
print('Shape of matrix after one hot encoding for cv data: ', cv_essay_bow.shape);
```

```
Shape of matrix after one hot encoding for train data:  (53531, 5000)
Shape of matrix after one hot encoding for test data:  (32775, 5000)
Shape of matrix after one hot encoding for cv data:  (22942, 5000)
```

In [40]:

```
bow_features_name = vectorizer.get_feature_names()
len(bow_features_name)
```

Out[40]:

```
5000
```

In [41]:

```
#vectorizing project_title

#configure CountVectorizer with word to occur in at least 10 documents.
vectorizer = CountVectorizer(min_df=10, ngram_range=(1,2), max_features=5000);

vectorizer.fit(X_train['processed_titles']);
```

```
#transforming title into vector
title_bow = vectorizer.transform(X_train['processed_titles']);
cv_title_bow = vectorizer.transform(X_cv['processed_titles']);
test_title_bow = vectorizer.transform(X_test['processed_titles']);

print('Shape of matrix after one hot encoding for train data: ', title_bow.shape);
print('Shape of matrix after one hot encoding for test data: ', test_title_bow.shape);
print('Shape of matrix after one hot encoding for cv data: ', cv_title_bow.shape);
```

```
Shape of matrix after one hot encoding for train data:  (53531, 5000)
Shape of matrix after one hot encoding for test data:  (32775, 5000)
Shape of matrix after one hot encoding for cv data:  (22942, 5000)
```

In [42]:

```
bow_features_name.extend(vectorizer.get_feature_names())
print(len(bow_features_name))
```

```
10000
```

In [43]:

```
len(features_name_list)
```

Out[43]:

```
103
```

In [44]:

```
final_bow_featues_name = [];
final_bow_featues_name.extend(features_name_list);
final_bow_featues_name.extend(bow_features_name);
print(len(final_bow_featues_name))
```

```
10103
```

Vectorizing using tf-idf

In [45]:

```
#vectorizing essay

#importing TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

#configuring TfidfVectorizer with a word to occur atleast in 10 documnets.
vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,2), max_features=5000)

vectorizer.fit(X_train['processed_essay']);

#vectorizing essay using tfidf
essay_tfidf = vectorizer.transform(X_train['processed_essay']);
test_essay_tfidf = vectorizer.transform(X_test['processed_essay']);
cv_essay_tfidf = vectorizer.transform(X_cv['processed_essay']);

print("Shape of matrix after one hot encoding for train data: ",essay_tfidf.shape)
print("Shape of matrix after one hot encoding for test data: ",test_essay_tfidf.shape)
print("Shape of matrix after one hot encoding for cv data: ",cv_essay_tfidf.shape)
```

```
Shape of matrix after one hot encoding for train data:  (53531, 5000)
Shape of matrix after one hot encoding for test data:  (32775, 5000)
Shape of matrix after one hot encoding for cv data:  (22942, 5000)
```

In [46]:

```
tfidf_features_name = vectorizer.get_feature_names()
len(tfidf_features_name)
len(tfidf_features_name)
```

Out[46]:

```
5000
```

In [47]:

```
#vectorizing project_title

vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,2), max_features=5000);

vectorizer.fit(X_train['processed_titles']);

title_tfidf = vectorizer.transform(X_train['processed_titles']);
test_title_tfidf = vectorizer.transform(X_test['processed_titles']);
cv_title_tfidf = vectorizer.transform(X_cv['processed_titles']);

print('Shape of title_tfidf after one hot encoding for train data ', title_tfidf.shape)
print('Shape of title_tfidf after one hot encoding for test data ', test_title_tfidf.shape)
print('Shape of title_tfidf after one hot encoding for cv data ', cv_title_tfidf.shape)
```

```
Shape of title_tfidf after one hot encoding for train data  (53531, 5000)
Shape of title_tfidf after one hot encoding for test data  (32775, 5000)
Shape of title_tfidf after one hot encoding for cv data  (22942, 5000)
```

In [48]:

```
tfidf_features_name.extend(vectorizer.get_feature_names())
print(len(tfidf_features_name))
```

```
10000
```

In [49]:

```
final_tfidf_featues_name = [];
final_tfidf_featues_name.extend(features_name_list);
final_tfidf_featues_name.extend(tfidf_features_name);
print(len(final_tfidf_featues_name))
```

```
10103
```

Vectorizing using avg w2v on train

In [50]:

```
#vectorizing essay

essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['processed_essay']): # 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
    essay_avg_w2v.append(vector)

#printing number of documents
print(len(essay_avg_w2v))

#printing dimension of each essay avg w2v
print(len(essay_avg_w2v[0]))
```

```
100%|
████████████████████████████████████████████████████████████████████████████████████████████████████
53531/53531 [00:21<00:00, 2507.56it/s]
```

```
53531
300
```

In [51]:

```
#vectorizing project_title

title_avg_w2v = [];
for sentance in tqdm(X_train['processed_titles']):
    vector = np.zeros(300);
    cnt_words = 0;
    for word in sentance.split():
        if word in glove_words:
            vector += model[word];
            cnt_words += 1;
    if cnt_words != 0:
        vector /= cnt_words;
    title_avg_w2v.append(vector);

print(len(title_avg_w2v));
print(len(title_avg_w2v[0]))
```

```
100%|
██████████████████████████████████████████████████████████████████████████████████████████████████████████
53531/53531 [00:00<00:00, 102022.12it/s]
```

```
53531
300
```

Vectorizing using avg w2v on CV

In [52]:

```
#vectorizing essay

cv_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['processed_essay']): # 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_essay_avg_w2v.append(vector)

#printing number of documents
print(len(cv_essay_avg_w2v))

#printing dimension of each essay avg w2v
print(len(cv_essay_avg_w2v[0]))
```

```
100%|
██████████████████████████████████████████████████████████████████████████████████████████████████████████
22942/22942 [00:07<00:00, 3005.56it/s]
```

```
22942
300
```

In [53]:

```
#vectorizing project_title

cv_title_avg_w2v = [];
for sentance in tqdm(X_cv['processed_titles']):
    vector = np.zeros(300);
    cnt_words = 0;
    for word in sentance.split():
        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]))
```

```
100%|
████████████████████████████████████████████████████████████████████████████████████████████████████
22942/22942 [00:00<00:00, 100678.85it/s]
```

```
22942
300
```

Vectorizing using avg w2v on test data

In [54]:

```
#vectorizing essay

test_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['processed_essay']): # 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_essay_avg_w2v.append(vector)

#printing number of documents
print(len(test_essay_avg_w2v))

#printing dimension of each essay avg w2v
print(len(test_essay_avg_w2v[0]))
```

```
100%|
████████████████████████████████████████████████████████████████████████████████████████████████████
32775/32775 [00:10<00:00, 3120.54it/s]
```

```
32775
300
```

In [55]:

```
#vectorizing project_title

test_title_avg_w2v = [];
for sentance in tqdm(X_test['processed_titles']):
    vector = np.zeros(300);
    cnt_words = 0;
    for word in sentance.split():
        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%|
████████████████████████████████████████████████████████████████████████████████████████████████████
```

```
32775/32775 [00:00<00:00, 118388.98it/s]
```

```
32775
300
```

Vectorizing using tfidf weighted w2v

In [56]:

```
#finding out tfidf words and corresponding idf value for essay

tfidf_model = TfidfVectorizer()

tfidf_model.fit(X_train['processed_essay'])

# 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 [57]:

```
#vectorizing essay

essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['processed_essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    essay_tfidf_w2v.append(vector)

print(len(essay_tfidf_w2v))
print(len(essay_tfidf_w2v[0]))
```

```
100%|
████████████████████████████████████████████████████████████████████████████████
| 53531/53531 [02:12<00:00, 403.82it/s]
```

```
53531
300
```

In [58]:

```
#vectorizing essay

cv_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['processed_essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_essay_tfidf_w2v.append(vector)
    cv_essay_tfidf_w2v.append(vector)

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

```
100%|
██████████████████████████████████████████████████████████████████████████████████████████████████
| 22942/22942 [01:09<00:00, 330.22it/s]
```

```
22942
300
```

In [59]:

```
#vectorizing essay

test_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['processed_essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_essay_tfidf_w2v.append(vector)

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

```
100%|
██████████████████████████████████████████████████████████████████████████████████████████████████
| 32775/32775 [01:28<00:00, 368.84it/s]
```

```
32775
300
```

In [60]:

```
#finding out tfidf words and corresponding idf value for project_title

tfidf_model = TfidfVectorizer()

tfidf_model.fit(X_train['processed_titles'])

# 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 [61]:

```
#vectorizing project_tile

title_tfidf_w2v = [];

for sentance in tqdm(X_train['processed_titles']):
    vector = np.zeros(300);
    tfidf_weight = 0;
    for word in sentance.split():
        if (word in glove_words) and (word in tfidf_words):
            tfidf = dictionary[word] * (sentance.count(word) / len(sentance.split()));
            vector = tfidf * model[word];
            tfidf_weight += tfidf;
    if tfidf_weight != 0:
        vector /= tfidf_weight;
    title_tfidf_w2v.append(vector);

print(len(title_tfidf_w2v))
print(len(title_tfidf_w2v[0]))
```

```
100%|
████████████████████████████████████████████████████████████████
53531/53531 [00:00<00:00, 90176.59it/s]
```

```
53531
300
```

In [62]:

```
#vectorizing project_tile

cv_title_tfidf_w2v = [];

for sentance in tqdm(X_cv['processed_titles']):
    vector = np.zeros(300);
    tfidf_weight = 0;
    for word in sentance.split():
        if (word in glove_words) and (word in tfidf_words):
            tfidf = dictionary[word] * (sentance.count(word) / len(sentance.split()));
            vector = tfidf * model[word];
            tfidf_weight += tfidf;
    if tfidf_weight != 0:
        vector /= tfidf_weight;
    cv_title_tfidf_w2v.append(vector);

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

```
100%|
████████████████████████████████████████████████████████████████
22942/22942 [00:00<00:00, 88974.07it/s]
```

```
22942
300
```

In [63]:

```
#vectorizing project_tile

test_title_tfidf_w2v = [];

for sentance in tqdm(X_test['processed_titles']):
    vector = np.zeros(300);
    tfidf_weight = 0;
    for word in sentance.split():
        if (word in glove_words) and (word in tfidf_words):
            tfidf = dictionary[word] * (sentance.count(word) / len(sentance.split()));
            vector = tfidf * model[word];
            tfidf_weight += tfidf;
    if tfidf_weight != 0:
        vector /= tfidf_weight;
    test_title_tfidf_w2v.append(vector);

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

```
100%|
████████████████████████████████████████████████████████████████
32775/32775 [00:00<00:00, 89360.82it/s]
```

```
32775
300
```

Merging data

Merging data

In [64]:

```
from scipy.sparse import hstack

#concatinating train data
#with bow
train_set_1 = hstack((school_state_vector, categories_vector, subcategories_vector, grade_vector, t
eacher_prefix_vector, price_standardized, teacher_number_standardized, quantity_standardized,
essay_bow, title_bow)).tocsr()

#with tfidf
train_set_2 = hstack((school_state_vector, categories_vector, subcategories_vector, grade_vector, t
eacher_prefix_vector, price_standardized, teacher_number_standardized, quantity_standardized,
essay_tfidf, title_tfidf)).tocsr()

#with avg w2v
train_set_3 = hstack((school_state_vector, categories_vector, subcategories_vector, grade_vector, t
eacher_prefix_vector, price_standardized, teacher_number_standardized, quantity_standardized,
essay_avg_w2v, title_avg_w2v)).tocsr()

#with tfidf wt w2v
train_set_4 = hstack((school_state_vector, categories_vector, subcategories_vector, grade_vector, t
eacher_prefix_vector, price_standardized, teacher_number_standardized, quantity_standardized,
essay_tfidf_w2v, title_tfidf_w2v)).tocsr()

#concatinating cv data
#with bow
cv_set_1 = hstack((cv_school_state_vector, cv_categories_vector, cv_subcategories_vector,
cv_grade_vector, cv_teacher_prefix_vector, cv_price_standardized, cv_teacher_number_standardized,
cv_quantity_standardized, cv_essay_bow, cv_title_bow)).tocsr()

#with tfidf
cv_set_2 = hstack((cv_school_state_vector, cv_categories_vector, cv_subcategories_vector,
cv_grade_vector, cv_teacher_prefix_vector, cv_price_standardized, cv_teacher_number_standardized,
cv_quantity_standardized, cv_essay_tfidf, cv_title_tfidf)).tocsr()

#with avg w2v
cv_set_3 = hstack((cv_school_state_vector, cv_categories_vector, cv_subcategories_vector,
cv_grade_vector, cv_teacher_prefix_vector, cv_price_standardized, cv_teacher_number_standardized,
cv_quantity_standardized, cv_essay_avg_w2v, cv_title_avg_w2v)).tocsr()

#with tfidf wt w2v
cv_set_4 = hstack((cv_school_state_vector, cv_categories_vector, cv_subcategories_vector,
cv_grade_vector, cv_teacher_prefix_vector, cv_price_standardized, cv_teacher_number_standardized,
cv_quantity_standardized, cv_essay_tfidf_w2v, cv_title_tfidf_w2v)).tocsr()


#concatinating test data
#with bow
test_set_1 = hstack((test_school_state_vector, test_categories_vector, test_subcategories_vector,
test_grade_vector, test_teacher_prefix_vector, test_price_standardized,
test_teacher_number_standardized, test_quantity_standardized, test_essay_bow,
test_title_bow)).tocsr()

#with tfidf
test_set_2 = hstack((test_school_state_vector, test_categories_vector, test_subcategories_vector,
test_grade_vector, test_teacher_prefix_vector, test_price_standardized,
test_teacher_number_standardized, test_quantity_standardized, test_essay_tfidf, test_title_tfidf))
.tocsr()

#with avg w2v
test_set_3 = hstack((test_school_state_vector, test_categories_vector, test_subcategories_vector,
test_grade_vector, test_teacher_prefix_vector, test_price_standardized,
test_teacher_number_standardized, test_quantity_standardized, test_essay_avg_w2v,
test_title_avg_w2v)).tocsr()

#with tfidf wt w2v
test_set_4 = hstack((test_school_state_vector, test_categories_vector, test_subcategories_vector,
test_grade_vector, test_teacher_prefix_vector, test_price_standardized,
test_teacher_number_standardized, test_quantity_standardized, test_essay_tfidf_w2v,
test_title_tfidf_w2v)).tocsr()
```

In [ ]:

```
train_set_2.shape
```

2.4 Appling Support Vector Machines on different kind of featurization as mentioned in the instructions

Apply Support Vector Machines on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

In [65]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [66]:

```
def get_fp_indices(y_pred, y_act):
    """
    function return index of only that rows/data points, which cause false positive
    """
    fp_indices = [];
    for i in np.arange(0, len(y_pred)):
        if y_pred[i] == 1 and y_act.iloc[i] == 0:
            fp_indices.append(i)
    return fp_indices;
```

Applying DT on set1

In [98]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

#defining parameters: alpha and penalty, whose different values we want to try.
tuned_parameters=[{'max_depth': [1, 5, 10, 50, 100, 500, 1000], 'min_samples_split': [5, 10, 100, 5
00]}]

#configuring DecisionTreeClassifier with hinge loss, to create linear SVC.
DT = DecisionTreeClassifier(class_weight="balanced");

#here GridSearchCV with metric score as auc, which is used for hyperparameter tuning, which gives
us optimal value of max_depth and min_samples_split
model = GridSearchCV(DT, tuned_parameters, scoring = 'roc_auc', cv=8, n_jobs = -1 )

model.fit(train_set_1, y_train);

print(model.best_estimator_)
print(model.score(test_set_1, y_test))
```

```
DecisionTreeClassifier(class_weight='balanced', criterion='gini',
            max_depth=10, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
0.6800753738168704
```

In [130]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score;
import multiprocessing as mp;

#creating list for holding auc value for train, cv
train_auc = [];
cv_auc = [];

#defining list of alpha's
max_depth = [1, 5, 10, 50, 100, 500];
min_samples_split = [5, 10, 100, 500];

for sample in min_samples_split:
    train_auc_temp = [];
    cv_auc_temp = [];
    for d in max_depth:
        DT = DecisionTreeClassifier(max_depth = d, min_samples_split = sample, class_weight="balanced");
        DT.fit(train_set_1, y_train);

        y_train_pred = DT.predict_proba(train_set_1)[:, 1];
        y_cv_pred = DT.predict_proba(cv_set_1)[:, 1];

        train_auc_temp.append(roc_auc_score(y_train, y_train_pred));
        cv_auc_temp.append(roc_auc_score(y_cv, y_cv_pred));

    train_auc.append(train_auc_temp)
    cv_auc.append(cv_auc_temp)
```

In [131]:

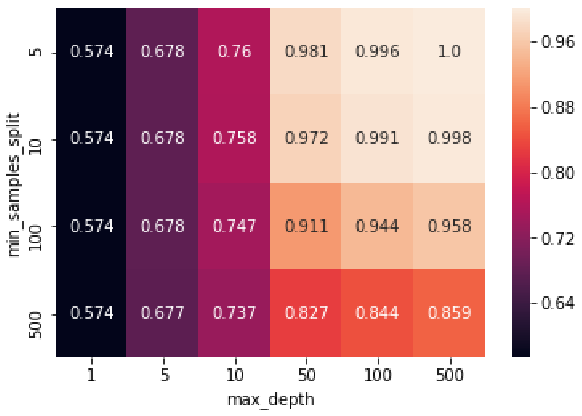
```
train_auc = [[round(train_auc[i][j], 3) for j in range(len(train_auc[1]))] for i in range(len(train_auc))];
cv_auc = [[round(cv_auc[i][j], 3) for j in range(len(cv_auc[1]))] for i in range(len(cv_auc))];
```

In [132]:

```
sns.heatmap(train_auc,  annot=True, fmt='', yticklabels=min_samples_split, xticklabels=max_depth);
plt.xlabel("max_depth");
plt.ylabel("min_samples_split")
```

Out[132]:

```
Text(33.0, 0.5, 'min_samples_split')
```

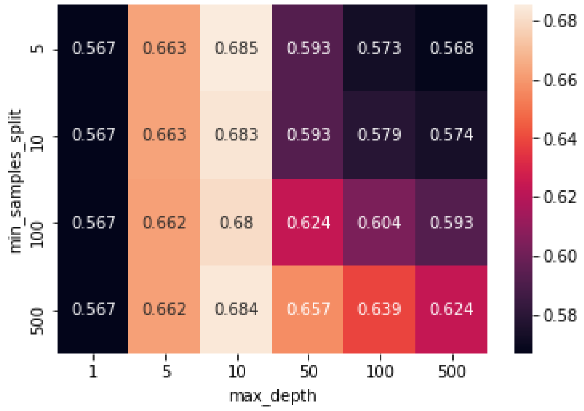


In [133]:

```
sns.heatmap(cv_auc,  annot=True, fmt='', yticklabels=min_samples_split, xticklabels=max_depth);
plt.xlabel("max_depth");
plt.ylabel("min_samples_split")
```

Out[133]:

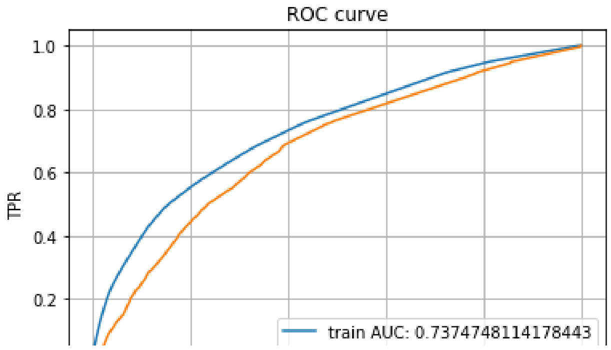
```
Text(33.0, 0.5, 'min_samples_split')
```

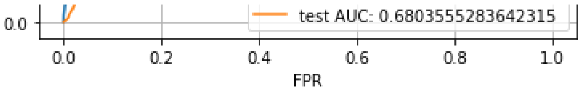


- As we can see, that we are getting optimal AUC with min_samples_split as 500 and with max_depth as 10

In [153]:

```python
from sklearn.metrics import roc_auc_score;

#obtained optimal values
set1_max_depth = 10;
set1_min_samples_split = 500;


DT = DecisionTreeClassifier(max_depth = set1_max_depth, min_samples_split = set1_min_samples_split
, class_weight="balanced");
DT.fit(train_set_1, y_train);

y_train_pred = DT.predict_proba(train_set_1)[:, 1];
y_test_pred = DT.predict_proba(test_set_1)[:, 1];

#obtaining auc value
train_auc = roc_auc_score(y_train, y_train_pred);
test_auc = roc_auc_score(y_test, y_test_pred);
set1_auc = test_auc;

#obtaining fpr, tpr and thresholds
train_fpr, train_tpr, train_thresholds = metrics.roc_curve(y_train, y_train_pred);
test_fpr, test_tpr, test_thresholds = metrics.roc_curve(y_test, y_test_pred);

#plotting ROC curve
plt.plot(train_fpr, train_tpr, label="train AUC: "+str(train_auc))
plt.plot(test_fpr, test_tpr, label="test AUC: "+str(test_auc))


plt.grid();
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend();
plt.show()
```

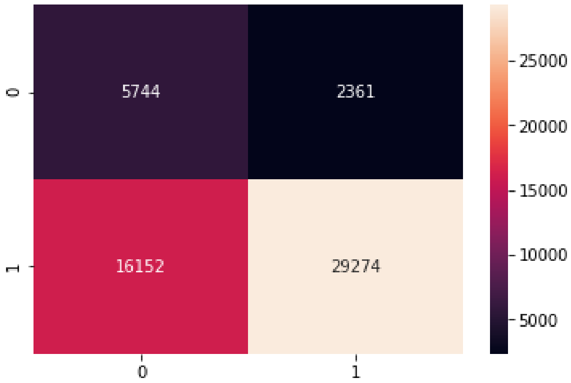




In [154]:

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
pred_output_train = predict(y_train_pred, train_thresholds, train_fpr, train_tpr);
print(confusion_matrix(y_train, pred_output_train))
train_cm = confusion_matrix(y_train, pred_output_train)
sns.heatmap(train_cm, annot=True, fmt="d");
```

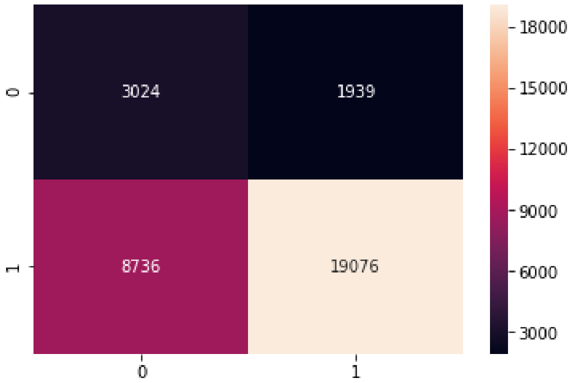
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.4567083837471647 for threshold 0.485
[[ 5744  2361]
 [16152 29274]]
```



In [155]:

```
print("Test confusion matrix")
pred_output_test = predict(y_test_pred, train_thresholds, test_fpr, test_tpr);
test_cm = confusion_matrix(y_test, pred_output_test)
sns.heatmap(test_cm, annot=True, fmt="d");
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.41791947018029774 for threshold 0.433
```



Plotting word cloud of false positive data point's essay feature

In [139]:

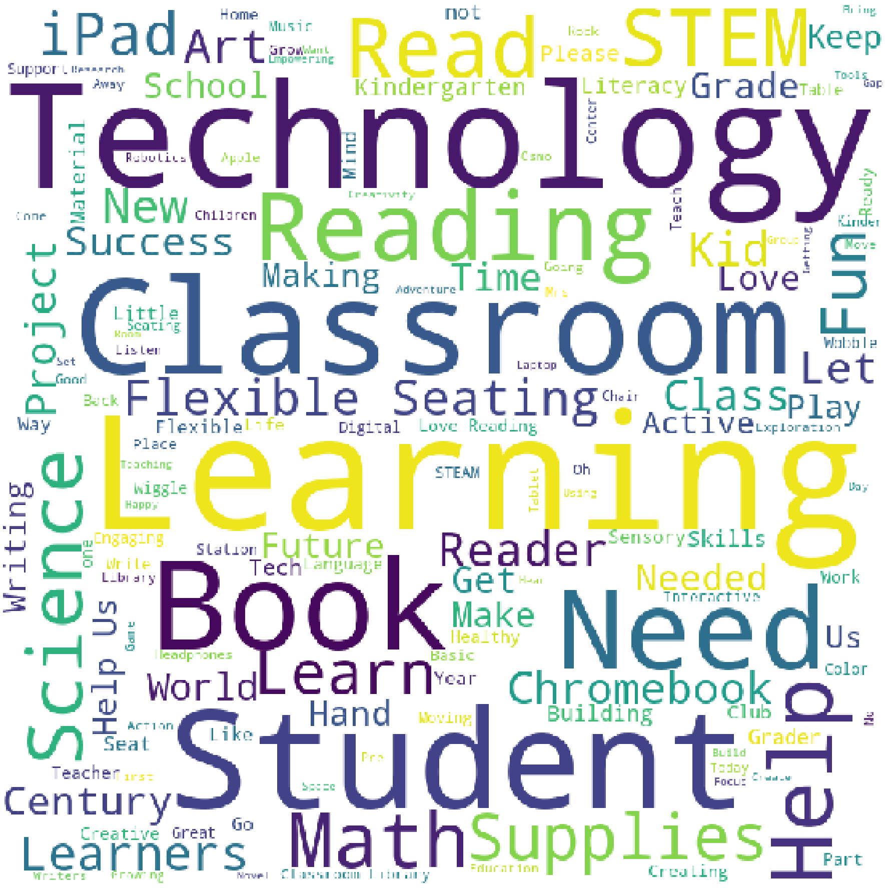
```
fp_indices = get_fp_indices(pred_output_test, y_test)
print(len(fp_indices))

print(fp_indices[:20])
```

```
fp_essay = X_test['processed_titles'].iloc[fp_indices]

essay_str = ' ';
for e in fp_essay:
    essay_str = essay_str + e + ' ';
```

```
2008
[3, 17, 29, 56, 60, 102, 136, 137, 156, 193, 238, 255, 275, 326, 330, 331, 339, 394, 416, 479]
```

In [140]:

```
#generating word cloud in python
from wordcloud import WordCloud

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

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

plt.show()
```



plotting box plot for price and pdf for teacher_number_of_previously_posted_projects present in false positive data points

In [141]:

```
fp_price = X_test['price'].iloc[fp_indices]
fp_teacher_no = X_test['teacher_number_of_previously_posted_projects'].iloc[fp_indices]
fp_output = y_test.iloc[fp_indices];

df = pd.DataFrame({'price':fp_price, 'teacher_no':fp_teacher_no, 'output':fp_output}).reset_index(
);
```

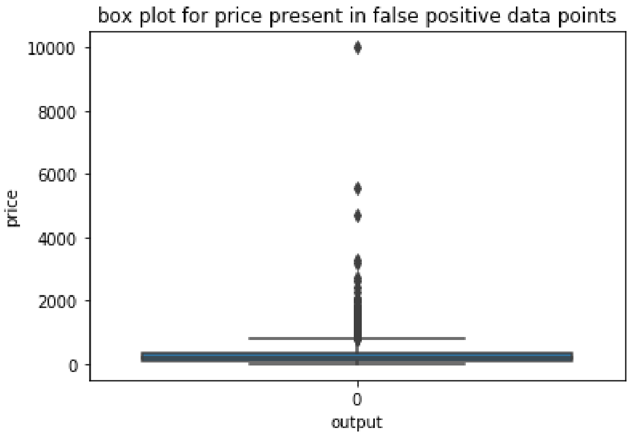
```
df.drop(['index'], axis=1, inplace=True)
df.head()
```

Out[141]:

	price	teacher_no	output
0	359.98	12	0
1	195.98	2	0
2	124.11	0	0
3	129.97	14	0
4	149.97	4	0

In [142]:

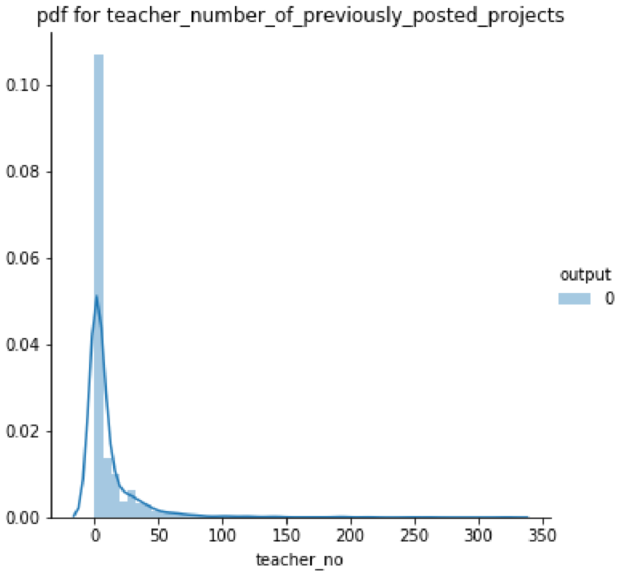
```
sns.boxplot(x='output',y='price', data=df)
plt.title('box plot for price present in false positive data points')
plt.show()
```



box plot for price present in false positive data points

In [143]:

```
sns.FacetGrid(df, hue="output", size=5) \
   .map(sns.distplot, "teacher_no") \
   .add_legend();
plt.title('pdf for teacher_number_of_previously_posted_projects')
plt.show();
```



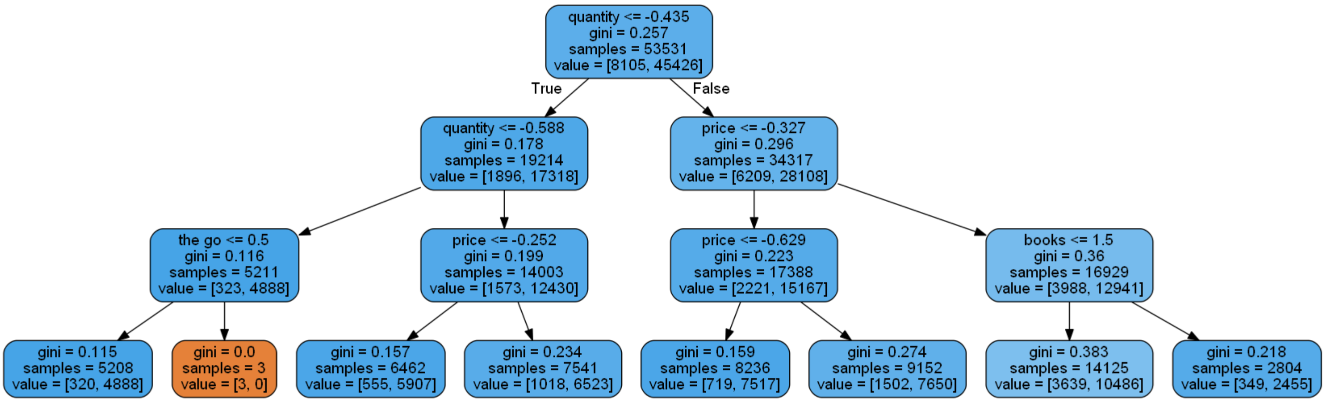
pdf for teacher_number_of_previously_posted_projects

visualizing decision tree

In [144]:

```
#genearting visualization of trained DT using graphviz: https://medium.com/@rnbrown/creating-and-v
isualizing-decision-trees-with-python-f8e8fa394176
from sklearn import tree
import pydotplus
from sklearn.externals.six import StringIO
from IPython.display import Image

dot_data = StringIO()

#visualizing using depth = 3
DT = DecisionTreeClassifier(max_depth = 3, min_samples_split = set1_min_samples_split);
DT.fit(train_set_1, y_train);

tree.export_graphviz(DT, out_file=dot_data,
                filled=True, rounded=True,
                feature_names=final_bow_featues_name)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

Image(graph.create_png())
```

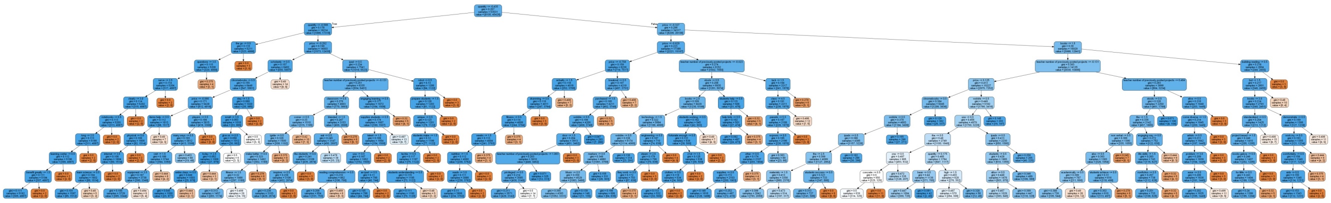
Out[144]:



In [145]:

```
dot_data = StringIO()

#visualizing using depth as the optimal depth we obtained using grid search
DT = DecisionTreeClassifier(max_depth = set1_max_depth, min_samples_split = set1_min_samples_split
);
DT.fit(train_set_1, y_train);

tree.export_graphviz(DT, out_file=dot_data,
                filled=True, rounded=True,
                feature_names=final_bow_featues_name)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

Image(graph.create_png())
```

Out[145]:



finding top 5 important features

In [146]:

```
imp_features_value = DT.feature_importances_;
```

```
type(imp_features_value)
imp_features_indices = imp_features_value.argsort()[::-1]

print('top 5 features')
impo_features = [(final_bow_featues_name[i], imp_features_value[i]) for i in
imp_features_indices[:5]]

print(impo_features)
```

```
top 5 features
[('price', 0.23034862554936666), ('quantity', 0.13500884669761232), ('teacher number of previously
posted projects', 0.08809167869507346), ('books', 0.07084548483434334), ('wobble',
0.043891002439080444)]
```

Applying DT on set2

In [149]:

```
from sklearn.model_selection import GridSearchCV

#defining parameters: alpha and penalty, whose different values we want to try.
tuned_parameters=[{'max_depth': [1, 5, 10, 50, 100, 500], 'min_samples_split': [5, 10, 100, 500]}]

#configuring DecisionTreeClassifier with hinge loss, to create linear SVC.
DT = DecisionTreeClassifier(class_weight="balanced");

#here GridSearchCV with metric score as auc, which is used for hyperparameter tuning, which gives
us optimal value of max_depth and min_samples_split
model = GridSearchCV(DT, tuned_parameters, scoring = 'roc_auc', cv=8, n_jobs=-1)

model.fit(train_set_2, y_train);

print(model.best_estimator_)
print(model.score(test_set_2, y_test))
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=10,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
0.6691939705177439
```

In [134]:

```
#creating list for holding auc value for train, cv
train_auc = [];
cv_auc = [];

#defining list of alpha's
max_depth = [1, 5, 10, 50, 100, 500];
min_samples_split = [5, 10, 100, 500];

for sample in min_samples_split:
    train_auc_temp = [];
    cv_auc_temp = [];
    for d in max_depth:
        DT = DecisionTreeClassifier(max_depth = d, min_samples_split = sample, class_weight="balanc
ed");
        DT.fit(train_set_2, y_train);

        y_train_pred = DT.predict_proba(train_set_2)[:, 1];
        y_cv_pred = DT.predict_proba(cv_set_2)[:, 1];

        train_auc_temp.append(roc_auc_score(y_train, y_train_pred));
        cv_auc_temp.append(roc_auc_score(y_cv, y_cv_pred));

    train_auc.append(train_auc_temp)
    cv_auc.append(cv_auc_temp)
```

In [135]:

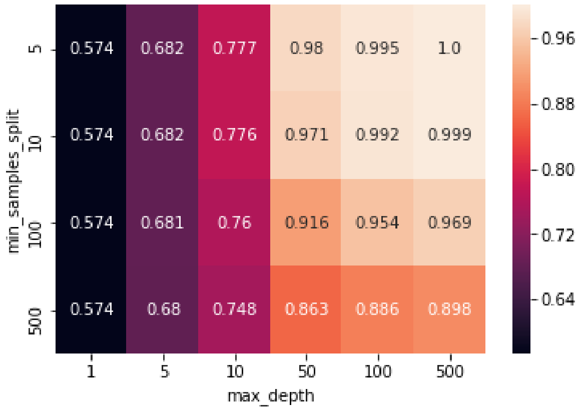
```
train_auc = [[round(train_auc[i][j], 3) for j in range(len(train_auc[1]))] for i in range(len(train
_auc))];
cv_auc = [[round(cv_auc[i][j], 3) for j in range(len(cv_auc[1]))] for i in range(len(cv_auc))];
```

In [136]:

```
sns.heatmap(train_auc,  annot=True, fmt='', yticklabels=min_samples_split, xticklabels=max_depth);
plt.xlabel("max_depth");
plt.ylabel("min_samples_split")
```

Out[136]:

```
Text(33.0, 0.5, 'min_samples_split')
```

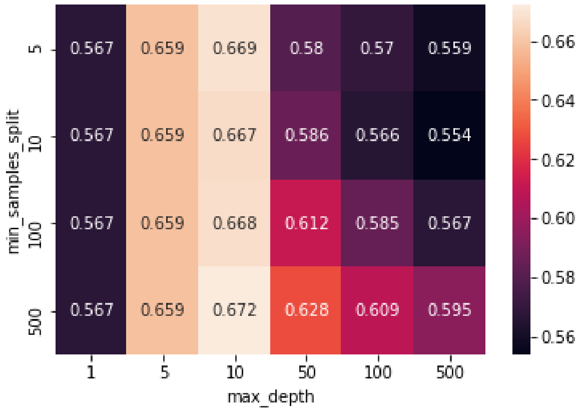


In [137]:

```
sns.heatmap(cv_auc,  annot=True, fmt='', yticklabels=min_samples_split, xticklabels=max_depth);
plt.xlabel("max_depth");
plt.ylabel("min_samples_split")
```

Out[137]:

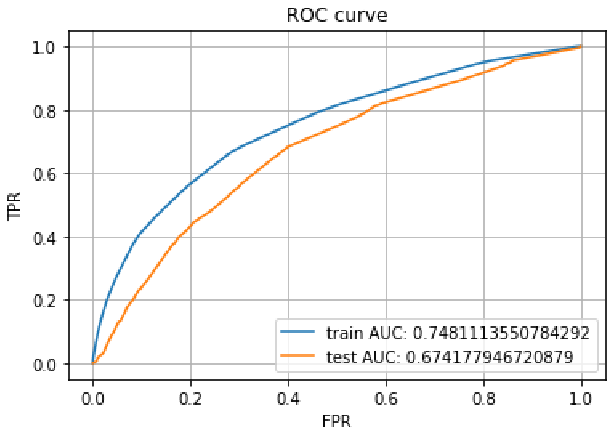
```
Text(33.0, 0.5, 'min_samples_split')
```



In [156]:

```
from sklearn.metrics import roc_auc_score;

#obtained optimal values
set2_max_depth = 10;
set2_min_samples_split = 500;


DT = DecisionTreeClassifier(max_depth = set2_max_depth, min_samples_split = set2_min_samples_split
, class_weight="balanced");
DT.fit(train_set_2, y_train);
```

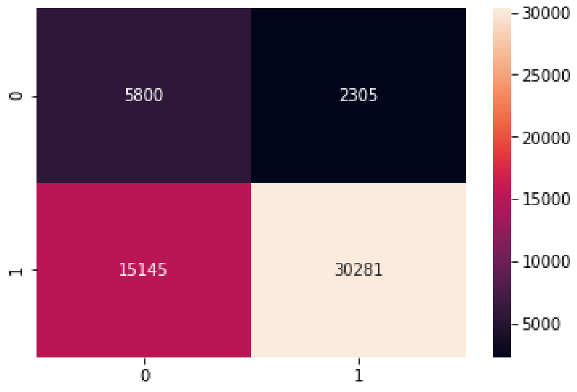
```
y_train_pred = DT.predict_proba(train_set_2)[:, 1];
y_test_pred = DT.predict_proba(test_set_2)[:, 1];

#obtaining auc value
train_auc = roc_auc_score(y_train, y_train_pred);
test_auc = roc_auc_score(y_test, y_test_pred);
set2_auc = test_auc;

#obtaining fpr, tpr and thresholds
train_fpr, train_tpr, train_thresholds = metrics.roc_curve(y_train, y_train_pred);
test_fpr, test_tpr, test_thresholds = metrics.roc_curve(y_test, y_test_pred);

#plotting ROC curve
plt.plot(train_fpr, train_tpr, label="train AUC: "+str(train_auc))
plt.plot(test_fpr, test_tpr, label="test AUC: "+str(test_auc))


plt.grid();
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend();
plt.show()
```



In [157]:

```
print("Train confusion matrix")
pred_output_train = predict(y_train_pred, train_thresholds, train_fpr, train_tpr);
train_cm = confusion_matrix(y_train, pred_output_train)
sns.heatmap(train_cm, annot=True, fmt="d");
```

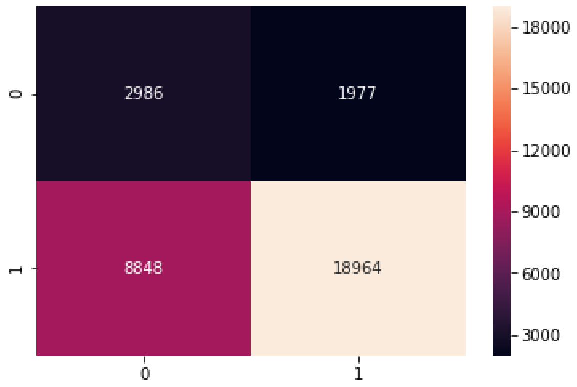
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.47702450661532403 for threshold 0.512
```



In [158]:

```
print("Test confusion matrix")
pred_output_test = predict(y_test_pred, train_thresholds, test_fpr, test_tpr);
test_cm = confusion_matrix(y_test, pred_output_test)
sns.heatmap(test_cm, annot=True, fmt="d");
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.4102449598335029 for threshold 0.416
```



Plotting word cloud of false positive data point's essay feature

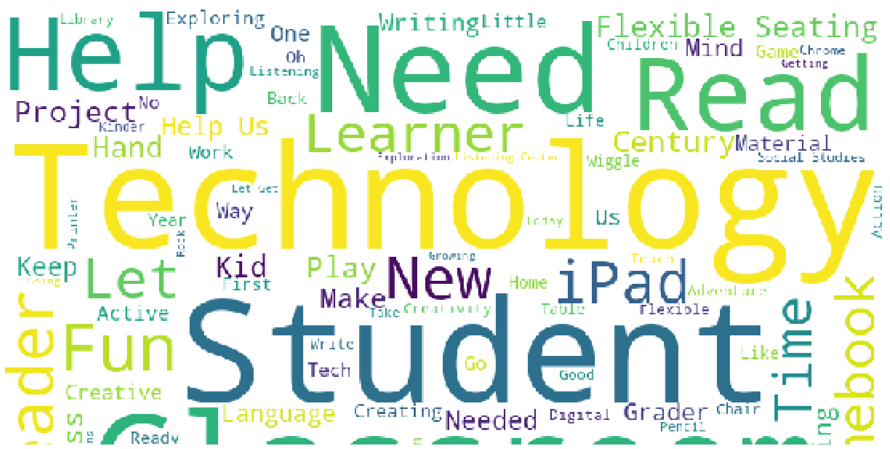
In [251]:

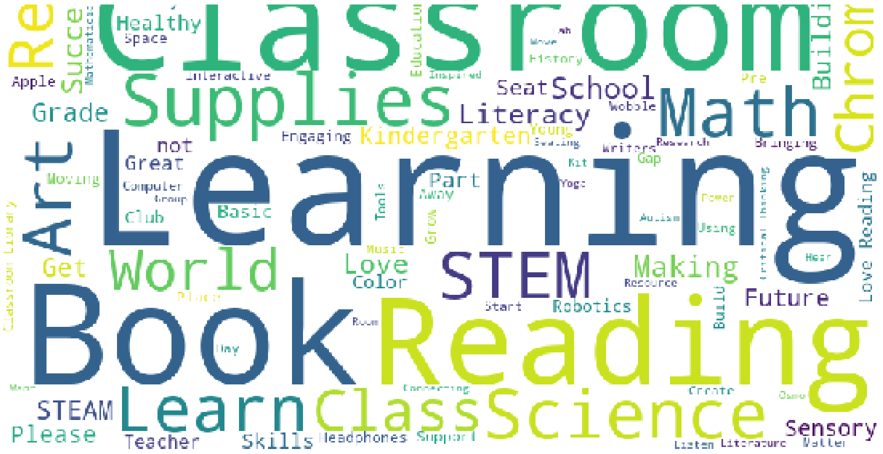
```python
fp_indices = get_fp_indices(pred_output_test, y_test)
print(len(fp_indices))

print(fp_indices[:20])

fp_essay = X_test['processed_titles'].iloc[fp_indices]

essay_str = ' ';
for e in fp_essay:
    essay_str = essay_str + e + ' ';
```

```
1925
[17, 29, 50, 102, 136, 137, 156, 179, 193, 232, 255, 266, 275, 330, 331, 394, 416, 418, 479, 487]
```

In [252]:

```python
#generating word cloud in python
from wordcloud import WordCloud

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

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

plt.show()
```





plotting box plot for price and pdf for teacher_number_of_previously_posted_projects present in false positive data points

In [253]:

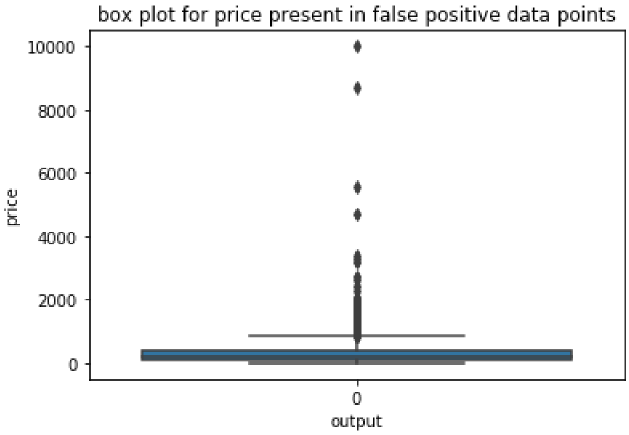
```
fp_price = X_test['price'].iloc[fp_indices]
fp_teacher_no = X_test['teacher_number_of_previously_posted_projects'].iloc[fp_indices]
fp_output = y_test.iloc[fp_indices];

df = pd.DataFrame({'price':fp_price, 'teacher_no':fp_teacher_no, 'output':fp_output}).reset_index(
);
df.drop(['index'], axis=1, inplace=True)
df.head()
```

Out[253]:

	price	teacher_no	output
0	195.98	2	0
1	124.11	0	0
2	127.43	8	0
3	349.95	4	0
4	1299.99	5	0

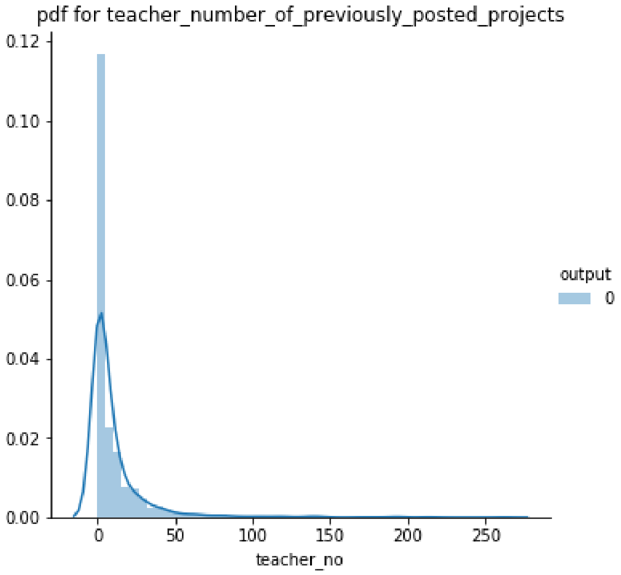
In [254]:

```
sns.boxplot(x='output',y='price', data=df)
plt.title('box plot for price present in false positive data points')
plt.show()
```



In [255]:

```
sns.FacetGrid(df, hue="output", size=5) \
    .map(sns.distplot, "teacher_no") \
    .add_legend();
plt.title('pdf for teacher_number_of_previously_posted_projects')
plt.show();
```

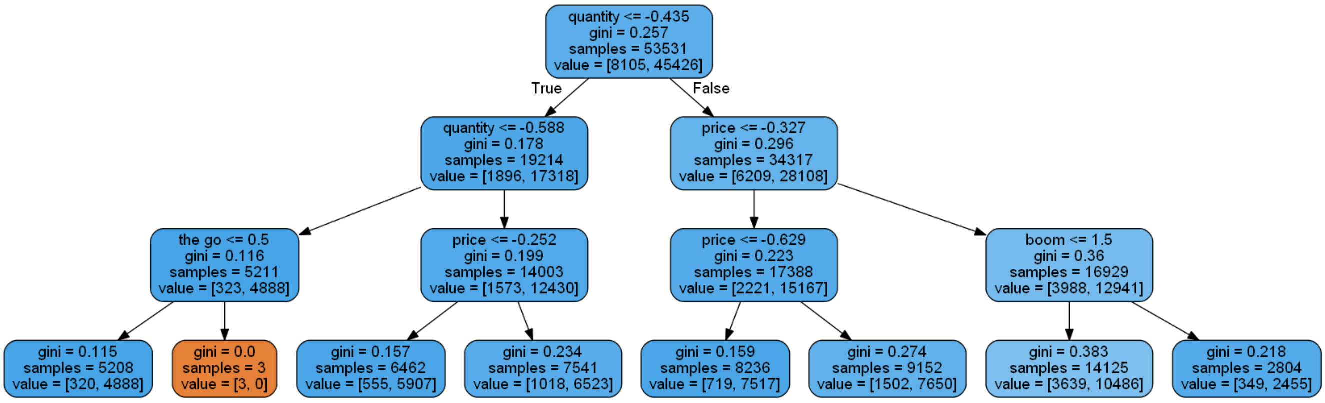


visualizing decision tree

In [259]:

```
#genearting visualization of trained DT using graphviz: https://medium.com/@rnbrown/creating-and-v
isualizing-decision-trees-with-python-f8e8fa394176
from sklearn import tree
import pydotplus
from sklearn.externals.six import StringIO
from IPython.display import Image

dot_data = StringIO()

#visualizing using depth = 3
DT = DecisionTreeClassifier(max_depth = 3, min_samples_split = set1_min_samples_split);
DT.fit(train_set_1, y_train);

tree.export_graphviz(DT, out_file=dot_data,
                filled=True, rounded=True,
                feature_names=final_tfidf_featues_name)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

Image(graph.create_png())
```

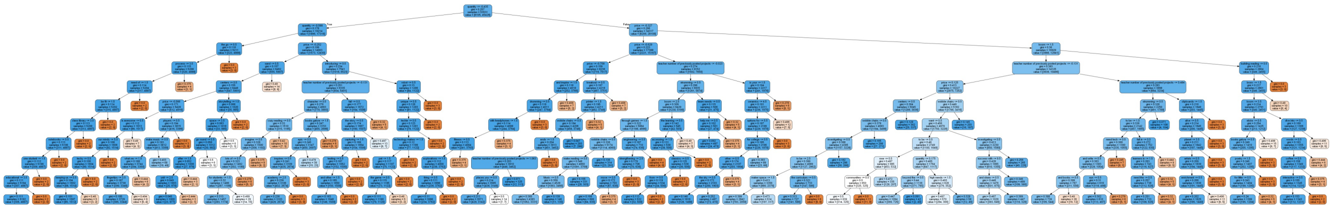
Out[259]:



In [260]:

```
dot_data = StringIO()
dot_data = StringIO()

#visualizing using depth as the optimal depth we obtained using grid search
DT = DecisionTreeClassifier(max_depth = set1_max_depth, min_samples_split = set1_min_samples_split
);
DT.fit(train_set_1, y_train);

tree.export_graphviz(DT, out_file=dot_data,
                filled=True, rounded=True,
                feature_names=final_tfidf_featues_name)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

Image(graph.create_png())
```

Out[260]:



selecting top 5000 features

In [82]:

```
print(train_set_1.shape)
```

```
(53531, 10103)
```

In [73]:

```
imp_features_value = DT.feature_importances_;
type(imp_features_value)
imp_features_indices = imp_features_value.argsort()[::-1][:5000]

print('top 5 features')
impo_features = [(final_bow_featues_name[i], imp_features_value[i]) for i in
imp_features_indices[:5]]

print(impo_features)
```

```
top 5 features
[('price', 0.195176668797981), ('quantity', 0.13689819934989994), ('nannan', 0.0946571681890948),
('teacher number of previously posted projects', 0.06702360314793329), ('books',
0.06102745971945269)]
```

In [75]:

```
train_set_5 = train_set_2[:, imp_features_indices]
test_set_5 = test_set_2[:, imp_features_indices]
cv_set_5 = cv_set_2[:, imp_features_indices]
```

In [76]:

```
print(train_set_5.shape, test_set_5.shape, cv_set_5.shape)
```

```
(53531, 5000) (32775, 5000) (22942, 5000)
```

Applying DT on set3

In [ ]:

```
#defining parameters: alpha and penalty, whose different values we want to try.
```

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

#configuring DecisionTreeClassifier with hinge loss, to create linear SVC.
DT = DecisionTreeClassifier(class_weight="balanced");

#here GridSearchCV with metric score as auc, which is used for hyperparameter tuning, which gives us optimal value of max_depth and min_samples_split
model = GridSearchCV(DT, tuned_parameters, scoring = 'roc_auc', cv=8, n_jobs=-1)

model.fit(train_set_3, y_train);

print(model.best_estimator_)
print(model.score(test_set_3, y_test))
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=10,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
0.6580969271849425
```

In [138]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score;

#creating list for holding auc value for train, cv
train_auc = [];
cv_auc = [];

#defining list of alpha's
max_depth = [1, 5, 10, 50, 100, 500];
min_samples_split = [5, 10, 100, 500];

for sample in min_samples_split:
    train_auc_temp = [];
    cv_auc_temp = [];
    for d in max_depth:
        DT = DecisionTreeClassifier(max_depth = d, min_samples_split = sample, class_weight="balanced");
        DT.fit(train_set_3, y_train);

        y_train_pred = DT.predict_proba(train_set_3)[:, 1];
        y_cv_pred = DT.predict_proba(cv_set_3)[:, 1];

        train_auc_temp.append(roc_auc_score(y_train, y_train_pred));
        cv_auc_temp.append(roc_auc_score(y_cv, y_cv_pred));

    train_auc.append(train_auc_temp)
    cv_auc.append(cv_auc_temp)
```

In [139]:

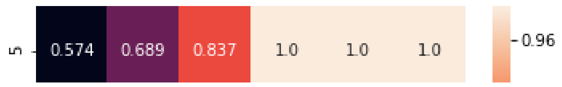
```
train_auc = [[round(train_auc[i][j], 3) for j in range(len(train_auc[1]))] for i in range(len(train_auc))];
cv_auc = [[round(cv_auc[i][j], 3) for j in range(len(cv_auc[1]))] for i in range(len(cv_auc))];
```

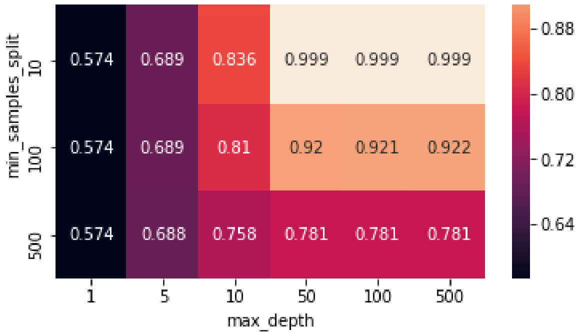
In [140]:

```
sns.heatmap(train_auc,  annot=True, fmt='', yticklabels=min_samples_split, xticklabels=max_depth);
plt.xlabel("max_depth");
plt.ylabel("min_samples_split")
```

Out[140]:

```
Text(33.0, 0.5, 'min_samples_split')
```



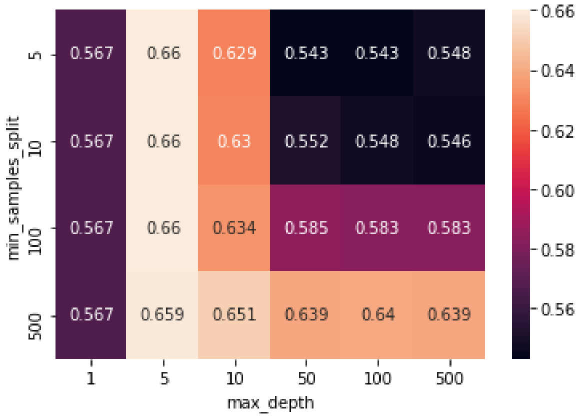


In [141]:

```
sns.heatmap(cv_auc,  annot=True, fmt='', yticklabels=min_samples_split, xticklabels=max_depth);
plt.xlabel("max_depth");
plt.ylabel("min_samples_split")
```

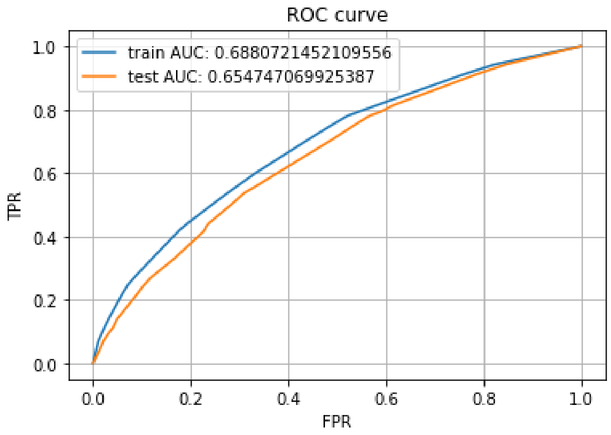
Out[141]:

```
Text(33.0, 0.5, 'min_samples_split')
```



In [159]:

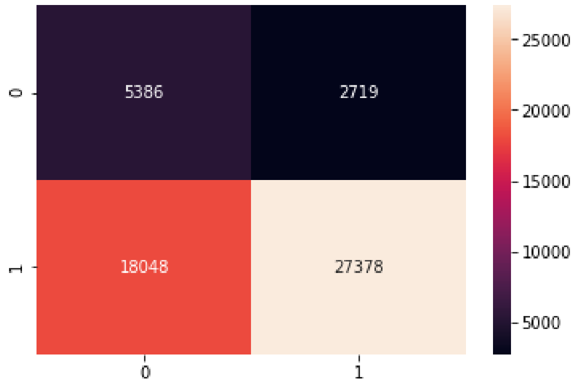
```
from sklearn.metrics import roc_auc_score;

#obtained optimal values
set3_max_depth = 5
set3_min_samples_split = 500;


DT = DecisionTreeClassifier(max_depth= set3_max_depth, min_samples_split = set3_min_samples_split,
class_weight="balanced");
DT.fit(train_set_3, y_train);

y_train_pred = DT.predict_proba(train_set_3)[:, 1];
y_test_pred = DT.predict_proba(test_set_3)[:, 1];

#obtaining auc value
train_auc = roc_auc_score(y_train, y_train_pred);
test_auc = roc_auc_score(y_test, y_test_pred);
set3_auc = test_auc;

#obtaining fpr, tpr and thresholds
train_fpr, train_tpr, train_thresholds = metrics.roc_curve(y_train, y_train_pred);
test_fpr, test_tpr, test_thresholds = metrics.roc_curve(y_test, y_test_pred);

#plotting ROC curve
plt.plot(train_fpr, train_tpr, label="train AUC: "+str(train_auc))
plt.plot(test_fpr, test_tpr, label="test AUC: "+str(test_auc))


plt.grid();
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend();
plt.show()
```



In [160]:

```
print("Train confusion matrix")
pred_output_train = predict(y_train_pred, train_thresholds, train_fpr, train_tpr);
train_cm = confusion_matrix(y_train, pred_output_train)
sns.heatmap(train_cm, annot=True, fmt="d");
```

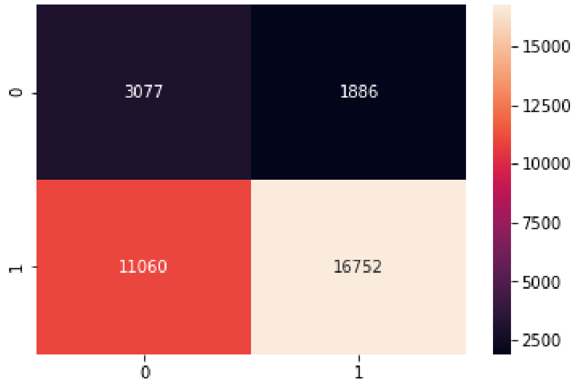
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.40050740711557975 for threshold 0.523
```



In [161]:

```
print("Test confusion matrix")
pred_output_test = predict(y_test_pred, train_thresholds, test_fpr, test_tpr);
test_cm = confusion_matrix(y_test, pred_output_test)
sns.heatmap(test_cm, annot=True, fmt="d");
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.37343727446182434 for threshold 0.523
```



Plotting word cloud of false positive data point's essay feature

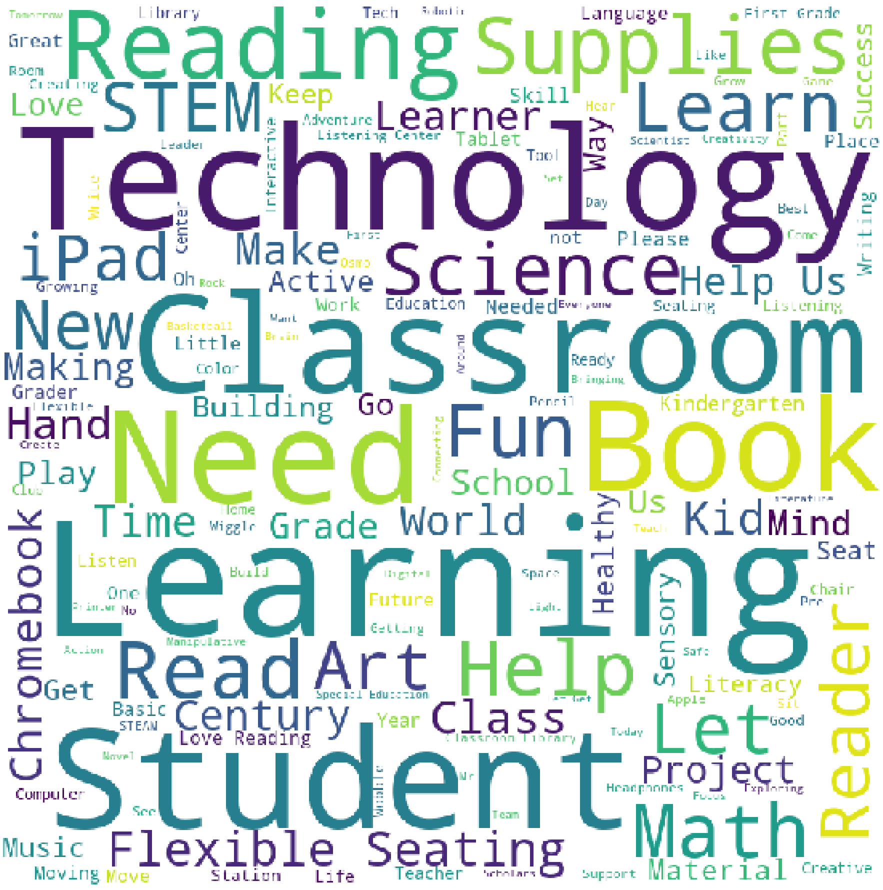
In [264]:

```
fp_indices = get_fp_indices(pred_output_test, y_test)
print(len(fp_indices))

print(fp_indices[:20])

fp_essay = X_test['processed_titles'].iloc[fp_indices]

essay_str = ' ';
for e in fp_essay:
    essay_str = essay_str + e + ' ';
```

```
2084
[6, 17, 29, 39, 56, 60, 90, 102, 136, 137, 142, 156, 193, 218, 232, 238, 255, 321, 330, 331]
```

In [265]:

```
#generating word cloud in python
from wordcloud import WordCloud

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

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

plt.show()
```



plotting box plot for price and pdf for teacher_number_of_previously_posted_projects present in false positive data points

In [266]:

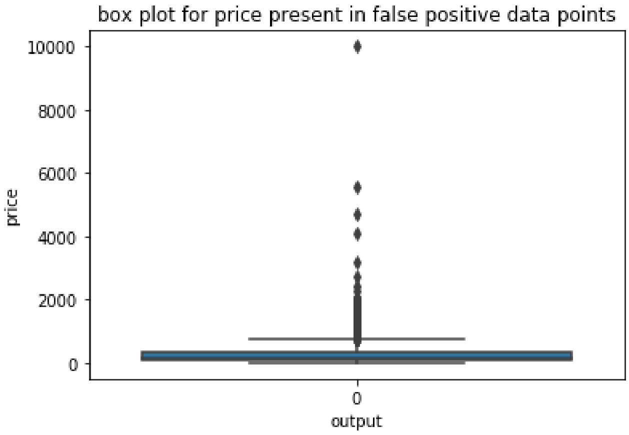
```
fp_price = X_test['price'].iloc[fp_indices]
fp_teacher_no = X_test['teacher_number_of_previously_posted_projects'].iloc[fp_indices]
fp_output = y_test.iloc[fp_indices];

df = pd.DataFrame({'price':fp_price, 'teacher_no':fp_teacher_no, 'output':fp_output}).reset_index(
);
df.drop(['index'], axis=1, inplace=True)
df.head()
```

Out[266]:

	price	teacher_no	output
0	161.57	0	0
1	195.98	2	0
2	124.11	0	0
3	250.70	12	0
4	129.97	14	0

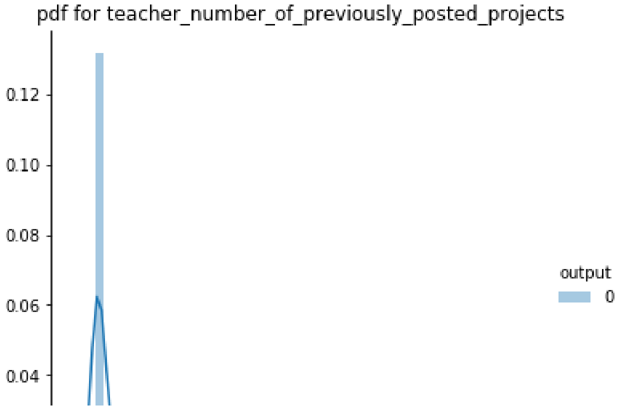
In [267]:

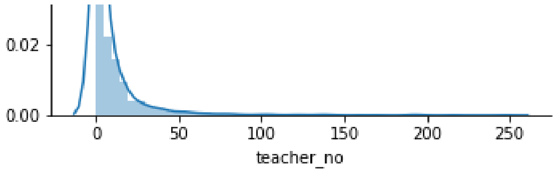
```
sns.boxplot(x='output',y='price', data=df)
plt.title('box plot for price present in false positive data points')
plt.show()
```



In [268]:

```
sns.FacetGrid(df, hue="output", size=5) \
   .map(sns.distplot, "teacher_no") \
   .add_legend();
plt.title('pdf for teacher_number_of_previously_posted_projects')
plt.show();
```





Applying DT on set4

In [86]:

```
from sklearn.model_selection import GridSearchCV

#defining parameters: alpha and penalty, whose different values we want to try.
tuned_parameters=[{'max_depth': [1, 5, 10, 50, 100, 500], 'min_samples_split': [5, 10, 100, 500]}]

#configuring DecisionTreeClassifier with hinge loss, to create linear SVC.
DT = DecisionTreeClassifier(class_weight="balanced");

#here GridSearchCV with metric score as auc, which is used for hyperparameter tuning, which gives
us optimal value of max_depth and min_samples_split
model = GridSearchCV(DT, tuned_parameters, scoring = 'roc_auc', cv=8, n_jobs = -1)

model.fit(train_set_4, y_train);

print(model.best_estimator_)
print(model.score(test_set_4, y_test))
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=10,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
0.6553265595001747
```

In [142]:

```
#creating list for holding auc value for train, cv
train_auc = [];
cv_auc = [];

#defining list of alpha's
max_depth = [1, 5, 10, 50, 100, 500];
min_samples_split = [5, 10, 100, 500];

for sample in min_samples_split:
    train_auc_temp = [];
    cv_auc_temp = [];
    for d in max_depth:
        DT = DecisionTreeClassifier(max_depth = d, min_samples_split = sample, class_weight="balanc
ed");
        DT.fit(train_set_4, y_train);

        y_train_pred = DT.predict_proba(train_set_4)[:, 1];
        y_cv_pred = DT.predict_proba(cv_set_4)[:, 1];

        train_auc_temp.append(roc_auc_score(y_train, y_train_pred));
        cv_auc_temp.append(roc_auc_score(y_cv, y_cv_pred));

    train_auc.append(train_auc_temp);
    cv_auc.append(cv_auc_temp);
```

In [143]:

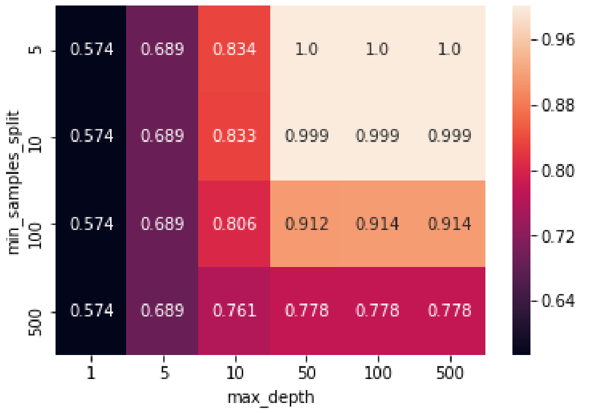
```
train_auc = [[round(train_auc[i][j], 3) for j in range(len(train_auc[1]))] for i in range(len(train
_auc))];
cv_auc = [[round(cv_auc[i][j], 3) for j in range(len(cv_auc[1]))] for i in range(len(cv_auc))];
```

In [144]:

```
sns.heatmap(train_auc,  annot=True, fmt='', yticklabels=min_samples_split, xticklabels=max_depth);
plt.xlabel("max_depth");
plt.ylabel("min_samples_split")
```

Out[144]:

```
Text(33.0, 0.5, 'min_samples_split')
```

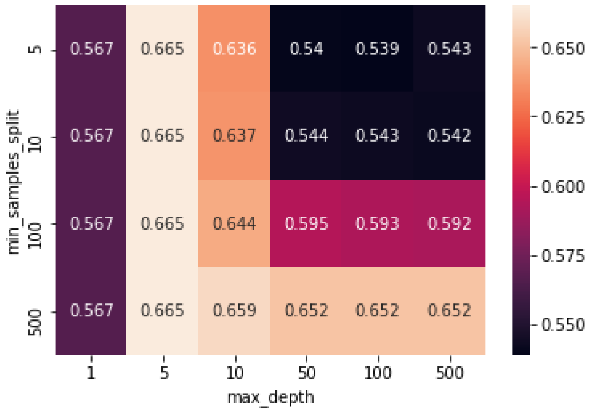


In [145]:

```
sns.heatmap(cv_auc,  annot=True, fmt='', yticklabels=min_samples_split, xticklabels=max_depth);
plt.xlabel("max_depth");
plt.ylabel("min_samples_split")
```

Out[145]:

```
Text(33.0, 0.5, 'min_samples_split')
```

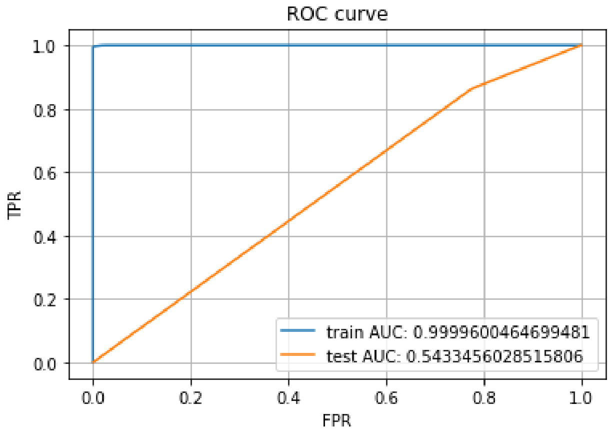


- As we can see from above plots, we are getting best AUC with min_samples_split as 500 and max_depth as 5

In [162]:

```
from sklearn.metrics import roc_auc_score;

#obtained optimal values
set4_max_depth = 500;
set4_min_samples_split = 5;


DT = DecisionTreeClassifier(max_depth = set4_max_depth, min_samples_split = set4_min_samples_split
, class_weight="balanced");
DT.fit(train_set_4, y_train);

y_train_pred = DT.predict_proba(train_set_4)[:, 1];
y_test_pred = DT.predict_proba(test_set_4)[:, 1];

#obtaining auc value
```

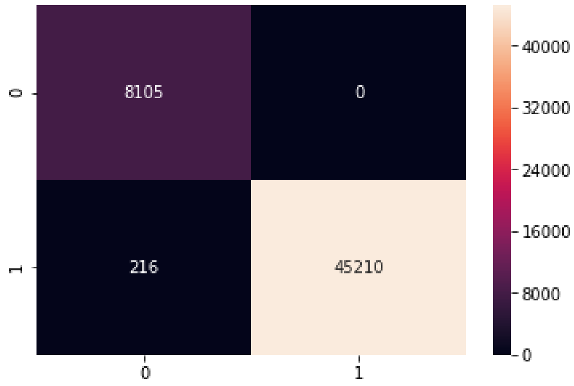
```
train_auc = roc_auc_score(y_train, y_train_pred);
test_auc = roc_auc_score(y_test, y_test_pred);
set4_auc = test_auc;

#obtaining fpr, tpr and thresholds
train_fpr, train_tpr, train_thresholds = metrics.roc_curve(y_train, y_train_pred);
test_fpr, test_tpr, test_thresholds = metrics.roc_curve(y_test, y_test_pred);

#plotting ROC curve
plt.plot(train_fpr, train_tpr, label="train AUC: "+str(train_auc))
plt.plot(test_fpr, test_tpr, label="test AUC: "+str(test_auc))


plt.grid();
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend();
plt.show()
```



In [163]:

```
print("Train confusion matrix")
pred_output_train = predict(y_train_pred, train_thresholds, train_fpr, train_tpr);
train_cm = confusion_matrix(y_train, pred_output_train)
sns.heatmap(train_cm, annot=True, fmt="d");
```

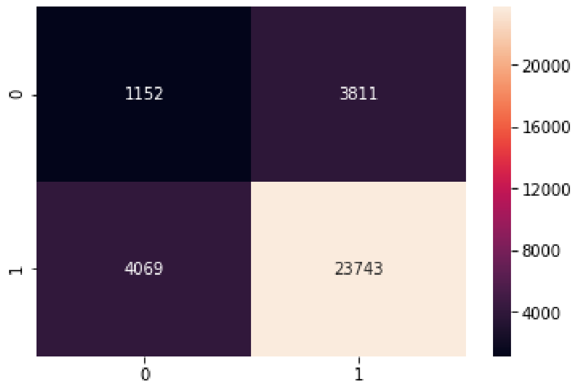
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.9952450138687096 for threshold 1.0
```



In [164]:

```
print("Test confusion matrix")
pred_output_test = predict(y_test_pred, train_thresholds, test_fpr, test_tpr);
test_cm = confusion_matrix(y_test, pred_output_test)
sns.heatmap(test_cm, annot=True, fmt="d");
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.1981579842133383 for threshold 1.0
```



Plotting word cloud of false positive data point's essay feature

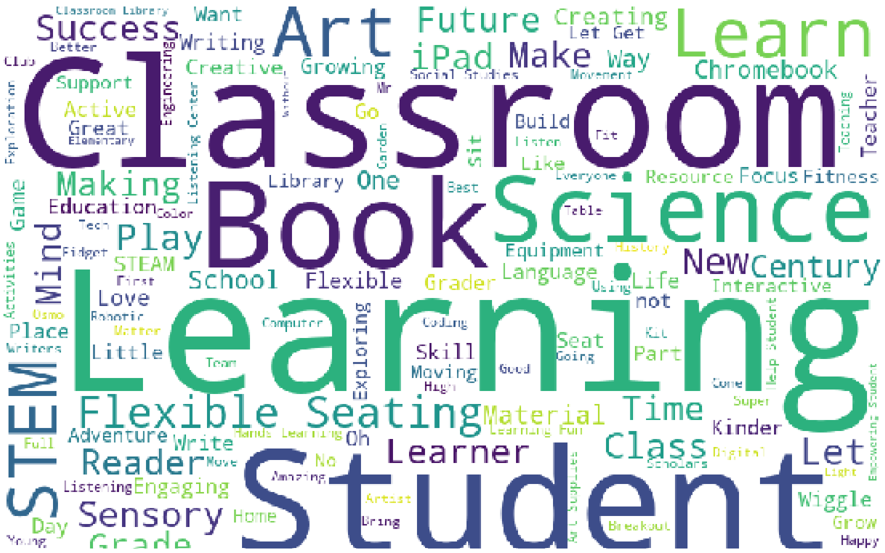
In [165]:

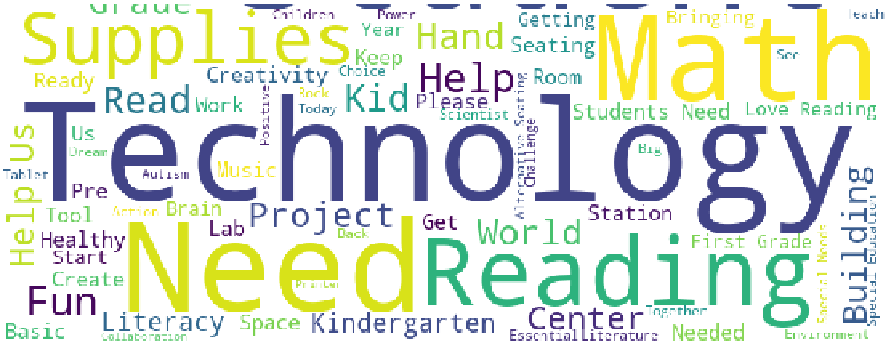
```
fp_indices = get_fp_indices(pred_output_test, y_test)
print(len(fp_indices))

print(fp_indices[:20])

fp_essay = X_test['processed_titles'].iloc[fp_indices]

essay_str = ' ';
for e in fp_essay:
    essay_str = essay_str + e + ' ';
```

```
3811
[3, 4, 6, 17, 29, 33, 36, 39, 50, 56, 76, 84, 90, 102, 104, 136, 141, 142, 156, 172]
```

In [166]:

```
#generating word cloud in python
from wordcloud import WordCloud

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

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

plt.show()
```





plotting box plot for price and pdf for teacher_number_of_previously_posted_projects present in false positive data points

In [167]:

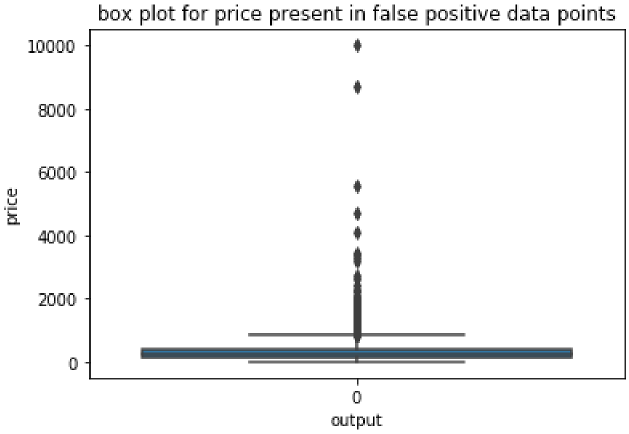
```
fp_price = X_test['price'].iloc[fp_indices]
fp_teacher_no = X_test['teacher_number_of_previously_posted_projects'].iloc[fp_indices]
fp_output = y_test.iloc[fp_indices];

df = pd.DataFrame({'price':fp_price, 'teacher_no':fp_teacher_no, 'output':fp_output}).reset_index(
);
df.drop(['index'], axis=1, inplace=True)
df.head()
```

Out[167]:

	price	teacher_no	output
0	359.98	12	0
1	417.23	3	0
2	161.57	0	0
3	195.98	2	0
4	124.11	0	0

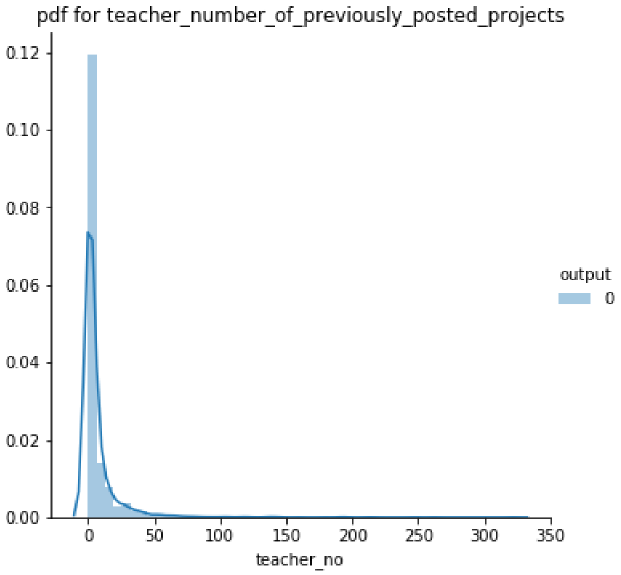
In [168]:

```
sns.boxplot(x='output',y='price', data=df)
plt.title('box plot for price present in false positive data points')
plt.show()
```



In [169]:

```
sns.FacetGrid(df, hue="output", size=5) \
   .map(sns.distplot, "teacher_no") \
   .add_legend();
plt.title('pdf for teacher_number_of_previously_posted_projects')
plt.show();
```



2.5 Support Vector Machines with added Features `Set 5`

In [146]:

```
#creating list for holding auc value for train, cv
train_auc = [];
cv_auc = [];

#defining list of alpha's
max_depth = [1, 5, 10, 50, 100, 500];
min_samples_split = [5, 10, 100, 500];

for sample in min_samples_split:
    train_auc_temp = [];
    cv_auc_temp = [];
    for d in max_depth:
        DT = DecisionTreeClassifier(max_depth = d, min_samples_split = sample, class_weight="balanc
ed");
        DT.fit(train_set_5, y_train);

        y_train_pred = DT.predict_proba(train_set_5)[:, 1];
        y_cv_pred = DT.predict_proba(cv_set_5)[:, 1];

        train_auc_temp.append(roc_auc_score(y_train, y_train_pred));
        cv_auc_temp.append(roc_auc_score(y_cv, y_cv_pred));

    train_auc.append(train_auc_temp);
    cv_auc.append(cv_auc_temp);
```

In [147]:

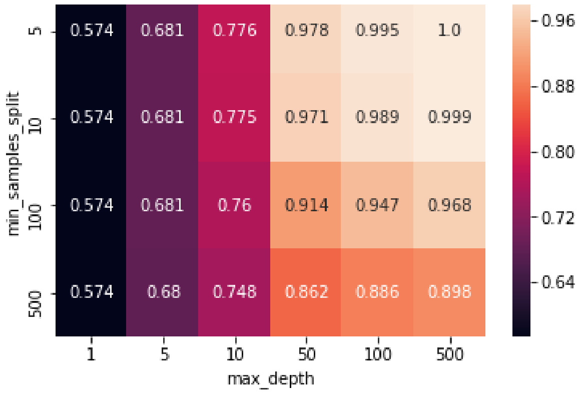
```
train_auc = [[round(train_auc[i][j], 3) for j in range(len(train_auc[1]))] for i in range(len(train
_auc))];
cv_auc = [[round(cv_auc[i][j], 3) for j in range(len(cv_auc[1]))] for i in range(len(cv_auc))];
```

In [148]:

```
sns.heatmap(train_auc,  annot=True, fmt='', yticklabels=min_samples_split, xticklabels=max_depth);
plt.xlabel("max_depth");
plt.ylabel("min_samples_split")
```

Out[148]:

```
Text(33.0, 0.5, 'min_samples_split')
```

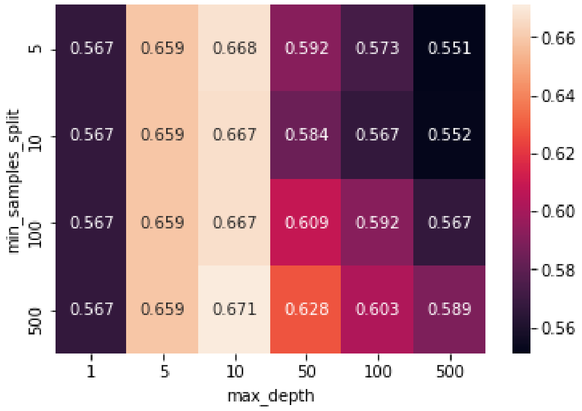


In [149]:

```
sns.heatmap(cv_auc,  annot=True, fmt='', yticklabels=min_samples_split, xticklabels=max_depth);
plt.xlabel("max_depth");
plt.ylabel("min_samples_split")
```

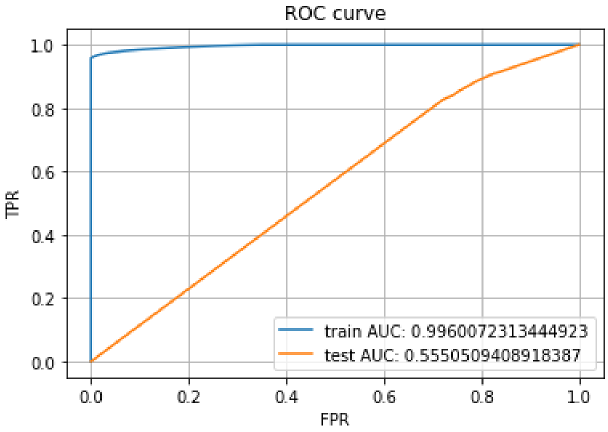
Out[149]:

```
Text(33.0, 0.5, 'min_samples_split')
```



In [170]:

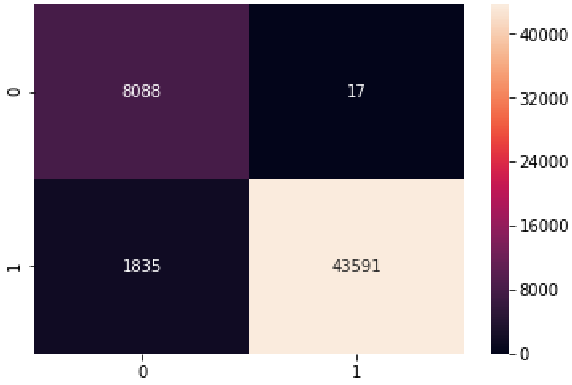
```
from sklearn.metrics import roc_auc_score;

#obtained optimal values
set5_max_depth = 500;
set5_min_samples_split = 10;


DT = DecisionTreeClassifier(max_depth = set5_max_depth, min_samples_split = set5_min_samples_split
);
DT.fit(train_set_5, y_train);

y_train_pred = DT.predict_proba(train_set_5)[:, 1];
y_test_pred = DT.predict_proba(test_set_5)[:, 1];

#obtaining auc value
train_auc = roc_auc_score(y_train, y_train_pred);
test_auc = roc_auc_score(y_test, y_test_pred);
set5_auc = test_auc;

#obtaining fpr, tpr and thresholds
train_fpr, train_tpr, train_thresholds = metrics.roc_curve(y_train, y_train_pred);
test_fpr, test_tpr, test_thresholds = metrics.roc_curve(y_test, y_test_pred);

#plotting ROC curve
plt.plot(train_fpr, train_tpr, label="train AUC: "+str(train_auc))
plt.plot(test_fpr, test_tpr, label="test AUC: "+str(test_auc))


plt.grid();
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend();
plt.show()
```



In [171]:

```
print("Train confusion matrix")
train_cm = confusion_matrix(y_train, predict(y_train_pred, train_thresholds, train_fpr,
train_tpr))
sns.heatmap(train_cm, annot=True, fmt="d");
```

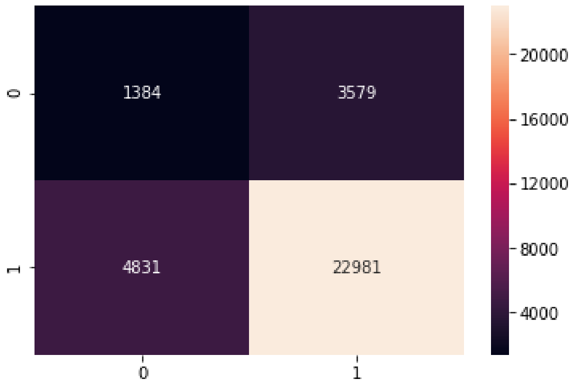
```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.957591889112902 for threshold 0.857
```



In [172]:

```
print("Test confusion matrix")
test_cm = confusion_matrix(y_test, predict(y_test_pred, train_thresholds, test_fpr, test_tpr))
sns.heatmap(test_cm, annot=True, fmt="d");
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.23042442740163296 for threshold 1.0
```



3. Conclusion

In [173]:

```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable

table = PrettyTable();
table.field_names = ['Vectorizer', 'Model', 'Hyper parameter(max_depth)', 'Hyper Parameter(min_samp
les_split)', 'AUC'];

table.add_row(['BOW', 'Brute', set1_max_depth, set1_min_samples_split, set1_auc]);
table.add_row(['TFIDF', 'Brute', set2_max_depth, set2_min_samples_split, set2_auc]);
table.add_row(['W2V', 'Brute', set3_max_depth, set3_min_samples_split, set3_auc]);
table.add_row(['TFIDFW2V', 'Brute', set4_max_depth, set4_min_samples_split, set4_auc]);
table.add_row(['Data with reduced dimensions', 'Brute', set5_max_depth, set5_min_samples_split, se
t5_auc]);

print(table)
```

```
+------------------------------+-------+----------------------------+---------------------------------
--+--------------------+
|          Vectorizer          | Model | Hyper parameter(max_depth) | Hyper
Parameter(min_samples_split) |        AUC         |
+------------------------------+-------+----------------------------+---------------------------------
--+--------------------+
|             BOW              | Brute |             10             |                500
| 0.6803555283642315 |
|            TFIDF             | Brute |             10             |                500
| 0.674177946720879  |
|             W2V              | Brute |             5              |                500
| 0.654747069925387  |
|           TFIDFW2V           | Brute |            500             |                 5
| 0.5433456028515806 |
| Data with reduced dimensions | Brute |            500             |                 10
| 0.5550509408918387 |
+------------------------------+-------+----------------------------+---------------------------------
--+--------------------+
```

In [ ]: