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	Desc
`project_id`	A unique identifier for the proposed project. **Example:** `p0`
`project_title`	Title of the project. **Exa** • `Art Will Make You H` • `First Grad`
`project_grade_category`	Grade level of students for which the project is targeted. One of the fo enumerated v • `Grades P` • `Grade` • `Grade` • `Grades`

Feature	Desc
project_subject_categories	One or more (comma-separated) subject categories for the project fr following enumerated list of v • Applied Lea • Care & H • Health & S • History & C • Literacy & Lan • Math & Sc • Music & The • Special • W **Exar** • Music & The • Literacy & Language, Math & Sc
school_state	State where school is located (Two-letter U.S. post (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_c **Exampl**
project_subject_subcategories	One or more (comma-separated) subject subcategories for the p **Exar** • Lit • Literature & Writing, Social Sci
project_resource_summary	An explanation of the resources needed for the project. **Exa** • My students need hands on literacy materials to ma sensory needs!<
project_essay_1	First application
project_essay_2	Second application
project_essay_3	Third application
project_essay_4	Fourth application
project_submitted_datetime	Datetime when project application was submitted. **Example:** 2016-(12:43:5
teacher_id	A unique identifier for the teacher of the proposed project. **Ex** bdf8baa8fedef6bfeec7ae4ff1c
teacher_prefix	Teacher's title. One of the following enumerated v • • • • • • Tea
teacher_number_of_previously_posted_projects	Number of project applications previously submitted by the same te **Examp**

* 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`

Feature	Description
`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 [87]:

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

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [88]:

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

In [89]:

```
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' 's
chool_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 [90]:

```
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[90]:

	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 [91]:

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

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

PREPROCESSING OF PROJECT GRADE CATEGORY

In [92]:

```
grade_categories=list(project_data['project_grade_category'].values)
clean_grades=[]
for i in grade_categories:
    temp=""
    for j in i.split(','):
        j=j.replace(' ','_')
        j=j.replace('-','_')
        temp+=j
        clean_grades.append(temp)
project_data['clean_grades']=clean_grades
project_data.drop(['project_grade_category'],axis=1,inplace=True)
```

1.3 preprocessing of `project_subject_subcategories`

In [93]:

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

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", '
        if 'The' in j.split(): # this will split each of the catogory based on space "Math
            j=j.replace('The','') # if we have the words "The" we are going to replace it w
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    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 [94]:

```
# 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 [95]:

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

Out[95]:

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

In [96]:

```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [97]:

```python
# 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 seco
nd or third languages. We are a melting pot of refugees, immigrants, and n
ative-born Americans bringing the gift of language to our school. \r\n\r\n
We have over 24 languages represented in our English Learner program with 
students at every level of mastery.  We also have over 40 countries repres
ented with the families within our school.  Each student brings a wealth o
f knowledge and experiences to us that open our eyes to new cultures, beli
efs, and respect.\"The limits of your language are the limits of your worl
d.\"-Ludwig Wittgenstein  Our English learner's have a strong support syst
em at home that begs for more resources.  Many times our parents are learn
ing to read and speak English along side of their children.  Sometimes thi
s creates barriers for parents to be able to help their child learn phonet
ics, letter recognition, and other reading skills.\r\n\r\nBy providing the
se dvd's and players, students are able to continue their mastery of the E
nglish language even if no one at home is able to assist.  All families wi
th 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 English Learner Teacher and will be sent home regularly to watch.  
The videos are to help the child develop early reading skills.\r\n\r\nPare
nts that do not have access to a dvd player will have the opportunity to c
heck out a dvd player to use for the year.  The plan is to use these video
s and educational dvd's for the years to come for other EL students.\r\nna
nnan
==================================================
The 51 fifth grade students that will cycle through my classroom this year
all love learning, at least most of the time. At our school, 97.3% of the
students receive free or reduced price lunch. Of the 560 students, 97.3% a
re minority students. \r\nThe school has a vibrant community that loves to
get together and celebrate. Around Halloween there is a whole school parad
e to show off the beautiful costumes that students wear. On Cinco de Mayo 
we put on a big festival with crafts made by the students, dances, and gam
es. At the end of the year the school hosts a carnival to celebrate the ha
rd work put in during the school year, with a dunk tank being the most pop
ular activity.My students will use these five brightly colored Hokki stool
s 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 have an i
ndividual 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 used by th
e 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 mis
sing, 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 in group wi
th me on the Hokki Stools, they are always moving, but at the same time do
ing their work. Anytime the students get to pick where they can sit, the H
okki Stools are the first to be taken. 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 student
s to do desk work and move at the same time. These stools will help studen
ts 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, th
ese chairs will take away the barrier that exists in schools for a child w
ho can't sit still.nannan
==================================================
How do you remember your days of school? Was it in a sterile environment w
ith plain walls, rows of desks, and a teacher in front of the room? A typi
cal day in our room is nothing like that. I work hard to create a warm inv
iting 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 r
aces in Arkansas.\r\nThey attend a Title I school, which means there is a
high enough percentage of free and reduced-price lunch to qualify. Our sch
ool is an \"open classroom\" concept, which is very unique as there are no
walls separating the classrooms. These 9 and 10 year-old students are very
eager learners; they are like sponges, absorbing all the information and e
xperiences and keep on wanting more.With these resources such as the comfy
red throw pillows and the whimsical nautical hanging decor and the blue fi
sh nets, I will be able to help create the mood in our classroom setting t
o be one of a themed nautical environment. Creating a classroom environmen
t is very important in the success in each and every child's education. Th
e 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 pi
ctures 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 c
ards 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 m
e to help make our classroom a fun, inviting, learning environment from da
y 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 to make
our new school year a very successful one. Thank you!nannan
==================================================
My kindergarten students have varied disabilities ranging from speech and
language delays, cognitive delays, gross/fine motor delays, to autism. The
y are eager beavers and always strive to work their hardest working past t
heir limitations. \r\n\r\nThe materials we have are the ones I seek out fo
r my students. I teach in a Title I school where most of the students rece
ive free or reduced price lunch.  Despite their disabilities and limitatio
ns, 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 groov
e and move as you were in a meeting? This is how my kids feel all the tim
e. 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 develop their core, which enha
nces gross motor and in Turn fine motor skills. \r\nThey also want to lear
n through games, my kids don't want to sit and do worksheets. They want to
learn to count by jumping and playing. Physical engagement is the key to o
ur 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.nannan
==================================================
The mediocre teacher tells. The good teacher explains. The superior teache
r demonstrates. The great teacher inspires. -William A. Ward\r\n\r\nMy sch
ool has 803 students which is makeup is 97.6% African-American, making up
the largest segment of the student body. A typical school in Dallas is mad
e up of 23.2% African-American students. Most of the students are on free
or reduced lunch. We aren't receiving doctors, lawyers, or engineers child
ren from rich backgrounds or neighborhoods. As an educator I am inspiring
```

```
minds of young children and we focus not only on academics but one smart,
effective, efficient, and disciplined students with good character.In our
classroom we can utilize the Bluetooth for swift transitions during class.
I use a speaker which doesn't amplify the sound enough to receive the mess
age. Due to the volume of my speaker my students can't hear videos or book
s clearly and it isn't making the lessons as meaningful. But with the blue
tooth speaker my students will be able to hear and I can stop, pause and r
eplay it at any time.\r\nThe cart will allow me to have more room for stor
age of things that are needed for the day and has an extra part to it I ca
n 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 [98]:

```
# 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 [99]:

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

```
My kindergarten students have varied disabilities ranging from speech and la
nguage delays, cognitive delays, gross/fine motor delays, to autism. They ar
e eager beavers and always strive to work their hardest working past their l
imitations. \r\n\r\nThe materials we have are the ones I seek out for my stu
dents. I teach in a Title I school where most of the students receive free o
r reduced price lunch.  Despite their disabilities and limitations, my stude
nts 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 groove and move as yo
u were in a meeting? This is how my kids feel all the time. The want to be a
ble to move as they learn or so they say.Wobble chairs are the answer and I
love then because they develop their core, which enhances gross motor and in
Turn fine motor skills. \r\nThey also want to learn through games, my kids d
o 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 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 [100]:

```
# \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 la
nguage delays, cognitive delays, gross/fine motor delays, to autism. They ar
e eager beavers and always strive to work their hardest working past their l
imitations.     The materials we have are the ones I seek out for my student
s. I teach in a Title I school where most of the students receive free or re
duced price 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 groove and move as you wer
e in a meeting? This is how my kids feel all the time. The want to be able t
o move as they learn or so they say.Wobble chairs are the answer and I love
then because they develop their core, which enhances gross motor and in Turn
fine motor skills.   They also want to learn through games, my kids do not w
ant to sit and do worksheets. They want to learn to count by jumping and pla
ying. Physical engagement is the key to our success. The number toss and col
or and shape mats can make that happen. My students will forget they are doi
ng work and just have the fun a 6 year old deserves.nannan
```

In [101]:

```
#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 la
nguage delays cognitive delays gross fine motor delays to autism They are ea
ger beavers and always strive to work their hardest working past their limit
ations 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 eager to learn and explore Have you ever felt like you had a
nts in your pants and you needed to groove 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 le
arn or so they say Wobble chairs are the answer and I love then because they
develop their core 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 work
sheets They want to learn to count by jumping and playing Physical engagemen
t is the key to our success The number toss and color and shape mats can mak
e that happen My students will forget they are doing work and just have the
fun a 6 year old deserves nannan
```

In [102]:

```
# 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'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'c
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"]
```

In [103]:

```
# 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)
    sent = sent.lower()
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|███████████████████████████████████████████████████████████
█| 109248/109248 [02:03<00:00, 885.34it/s]
```

In [104]:

```
# after preprocesing
preprocessed_essays[20000]
```

Out[104]:

```
'kindergarten students varied disabilities ranging speech language delays co
gnitive delays gross fine motor delays autism eager beavers always strive wo
rk hardest working past limitations materials ones seek students teach title
school students receive free reduced price lunch despite disabilities limita
tions students love coming school come eager learn explore ever felt like an
ts pants needed groove move meeting kids feel time want able move learn say
wobble chairs answer love develop core enhances gross motor turn fine motor
skills also want learn games kids not want sit worksheets want learn count j
umping playing physical engagement key success number toss color shape mats
make happen students forget work fun 6 year old deserves nannan'
```

1.4 Preprocessing of `project_title`

In [105]:

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

```
100%|████████████████████████████████████████████████████████████████████
| 109248/109248 [00:04<00:00, 22129.42it/s]
```

1.5 Preparing data for models

In [106]:

```
project_data.columns
```

Out[106]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', '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_approve
d',
       'clean_categories', 'clean_grades', 'clean_subcategories', 'essay'],
      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
```

1.5.1 Vectorizing Categorical data

- [https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/ (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)

In [107]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
categories_one_hot = vectorizer.fit_transform(project_data['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

```
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'liter
acy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
Shape of matrix after one hot encodig  (109248, 9)
```

In [108]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer()
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subcategories'].value
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)
```

```
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_governmen
t', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economic
s', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy',
'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness',
'history_geography', 'literacy', 'literature_writing', 'mathematics', 'musi
c', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 's
ocialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
Shape of matrix after one hot encodig  (109248, 30)
```

In [109]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
```

In [110]:

```
#vectorizing student state
vectorizer=CountVectorizer()
school_state_one_hot=vectorizer.fit_transform(project_data['school_state'].values)
print(vectorizer.get_feature_names())
print('shape of matrix after one hot encoding',school_state_one_hot.shape)
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'i
a', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo',
'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'o
r', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv',
'wy']
shape of matrix after one hot encoding (109248, 51)
```

In [111]:

```
#vectorizing project grade category
vectorizer=CountVectorizer()
project_grade_one_hot=vectorizer.fit_transform(project_data['clean_grades'].values)
print(vectorizer.get_feature_names())
print('shape of matrix after one hot encoding',project_grade_one_hot.shape)
```

```
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
shape of matrix after one hot encoding (109248, 4)
```

In [112]:

```
#vectorizing teacher prefix
x=project_data['teacher_prefix'].fillna('')
vectorizer = CountVectorizer()

teacher_prefix_one_hot = vectorizer.fit_transform(x.values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",teacher_prefix_one_hot.shape)
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
Shape of matrix after one hot encodig  (109248, 5)
```

In [ ]:

```

```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [113]:

```
# We are considering only the words which appeared in at least 10 documents(rows or project
vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_bow.shape)
```

```
Shape of matrix after one hot encodig  (109248, 16512)
```

In [114]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer=CountVectorizer(min_df=10)
title_bow=vectorizer.fit_transform(preprocessed_title)
print('Shape of matrix after vectorizing',title_bow.shape)
```

```
Shape of matrix after vectorizing (109248, 3222)
```

1.5.2.2 TFIDF vectorizer

In [115]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidfvectorizer = TfidfVectorizer(min_df=10)
text_tfidf = tfidfvectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (109248, 16512)
```

In [116]:

```
vectorizer=TfidfVectorizer(min_df=10)
title_tfidf=vectorizer.fit_transform(preprocessed_title)
print('Shape of matrix after vectorizing',title_tfidf.shape)
```

```
Shape of matrix after vectorizing (109248, 3222)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [0]:

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

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

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

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

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

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

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

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


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickl

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


'''
```

Out[26]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/3823034
```

```
9/4084039\ndef (https://stackoverflow.com/a/38230349/4084039\ndef) loadGlo
veModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(glove
File,\'r\', encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n
splitLine = line.split()\n        word = splitLine[0]\n        embedding =
np.array([float(val) for val in splitLine[1:]])\n        model[word] = emb
edding\n    print ("Done.",len(model)," words loaded!")\n    return model
\nmodel = loadGloveModel(\'glove.42B.300d.txt\')\n\n# ====================
========\nOutput:\n    \nLoading Glove Model\n1917495it [06:32, 4879.69it/
s]\nDone. 1917495  words loaded!\n\n# ============================\n\nword
s = []\nfor i in preproced_texts:\n    words.extend(i.split(\' \'))\n\nfor
i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the w
ords in the coupus", len(words))\nwords = set(words)\nprint("the unique wo
rds in the coupus", len(words))\n\ninter_words = set(model.keys()).interse
ction(words)\nprint("The number of words that are present in both glove ve
ctors and our coupus",       len(inter_words),"(",np.round(len(inter_word
s)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove = set(model.
keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus
[i] = model[i]\nprint("word 2 vec length", len(words_courpus))\n\n\n# stro
nging variables into pickle files python: http://www.jessicayung.com/how-t
o-use-pickle-to-save-and-load-variables-in-python/\n\nimport (http://www.j
essicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\n
import) pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dum
p(words_courpus, f)\n\n\n'
```

In [31]:

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

In [32]:

```python
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in (preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors.append(vector)

print(len(avg_w2v_vectors))
print(len(avg_w2v_vectors[0]))
```

```
109248
300
```

In [33]:

```
# average Word2Vec for project_title
# compute average word2vec for each review.
title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_title): # 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
    title_avg_w2v_vectors.append(vector)

print(len(title_avg_w2v_vectors))
print(len(title_avg_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████
| 109248/109248 [00:02<00:00, 45325.25it/s]

109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [34]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [35]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

print(len(tfidf_w2v_vectors))
print(len(tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█| 109248/109248 [04:33<00:00, 399.02it/s]

109248
300
```

In [36]:

```
# Similarly you can vectorize for title also
# compute average word2vec for each review.
title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    title_tfidf_w2v_vectors.append(vector)

print(len(title_tfidf_w2v_vectors))
print(len(title_tfidf_w2v_vectors[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
| 109248/109248 [00:05<00:00, 20548.18it/s]

109248
300
```

1.5.3 Vectorizing Numerical features

In [117]:

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

In [118]:

```
data1=project_data.drop(['id','teacher_id','project_essay_1','project_essay_2','project_ess
data1.head(2)
data=data1[0:100000]
data[0:2]
```

Out[118]:

	Unnamed: 0	teacher_prefix	school_state	project_submitted_datetime	project_title	project_res
0	160221	Mrs.	IN	2016-12-05 13:43:57	Educational Support for English Learners at Home	I opportu
1	140945	Mr.	FL	2016-10-25 09:22:10	Wanted: Projector for Hungry Learners	My students tc

In [119]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.prepro
from sklearn.preprocessing import StandardScaler

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()
price_scalar.fit(project_data['price'].values.reshape(-1,1)) # finding the mean and standar
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])

# Now standardize the data with above maen and variance.
price_standardized = price_scalar.transform(project_data['price'].values.reshape(-1, 1))
```

```
Mean : 298.1193425966608, Standard deviation : 367.49634838483496
```

In [120]:

```
price_standardized
```

Out[120]:

```
array([[-0.3905327 ],
       [ 0.00239637],
       [ 0.59519138],
       ...,
       [-0.15825829],
       [-0.61243967],
       [-0.51216657]])
```

In [121]:

```
projects_scalar = StandardScaler()
projects_scalar.fit(project_data['teacher_number_of_previously_posted_projects'].values.res
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])

# Now standardize the data with above maen and variance.
projects_standardized = projects_scalar.transform(project_data['teacher_number_of_previousl
projects_standardized
```

```
Mean : 298.1193425966608, Standard deviation : 367.49634838483496
```

Out[121]:

```
array([[-0.40152481],
       [-0.14951799],
       [-0.36552384],
       ...,
       [-0.29352189],
       [-0.40152481],
       [-0.40152481]])
```

In [122]:

```
projects_scalar = StandardScaler()
projects_scalar.fit(project_data['quantity'].values.reshape(-1,1)) # finding the mean and s
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])

# Now standardize the data with above maen and variance.
quantity_standardized = projects_scalar.transform(project_data['quantity'].values.reshape(-
quantity_standardized
```

```
Mean : 298.1193425966608, Standard deviation : 367.49634838483496
```

Out[122]:

```
array([[ 0.23047132],
       [-0.60977424],
       [ 0.19227834],
       ...,
       [-0.4951953 ],
       [-0.03687954],
       [-0.45700232]])
```

In [ ]:

```

```

1.5.4 Merging all the above features

- we need to merge all the numerical vectors i.e catogorical, text, numerical vectors

In [123]:

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(teacher_prefix_one_hot.shape)
print(school_state_one_hot.shape)
print(project_grade_one_hot.shape)
print(title_bow.shape)
print(text_bow.shape)
print(price_standardized.shape)
print(projects_standardized.shape)
```

```
(109248, 9)
(109248, 30)
(109248, 5)
(109248, 51)
(109248, 4)
(109248, 3222)
(109248, 16512)
(109248, 1)
(109248, 1)
```

In [124]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
X.shape
```

Out[124]:

```
(109248, 16552)
```

In [125]:

```
y1=project_data['project_is_approved']
print(y1.shape)
y=y1[0:100000]
```

```
(109248,)
```

In [126]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

Computing Sentiment Scores

In [127]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

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

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest stu
for learning my students learn in many different ways using all of our senses and multiple
of techniques to help all my students succeed students in my class come from a variety of d
for wonderful sharing of experiences and cultures including native americans our school is
learners which can be seen through collaborative student project based learning in and out
in my class love to work with hands on materials and have many different opportunities to p
mastered having the social skills to work cooperatively with friends is a crucial aspect of
montana is the perfect place to learn about agriculture and nutrition my students love to r
in the early childhood classroom i have had several kids ask me can we try cooking with rea
and create common core cooking lessons where we learn important math and writing concepts w
food for snack time my students will have a grounded appreciation for the work that went in
of where the ingredients came from as well as how it is healthy for their bodies this proje
nutrition and agricultural cooking recipes by having us peel our own apples to make homemad
and mix up healthy plants from our classroom garden in the spring we will also create our c
shared with families students will gain math and literature skills as well as a life long e
nannan'
ss = sid.polarity_scores(for_sentiment)

for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

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

```
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

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 [4,6, 8, 9,10,12,14,17] , and the best `min_samples_split` in range [2,10,20,30,40,50])**

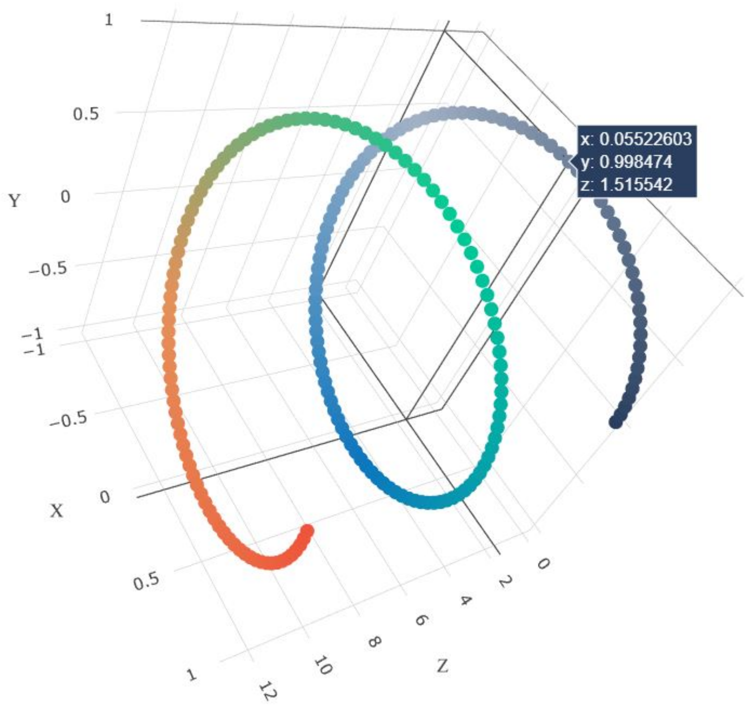
   - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - 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**

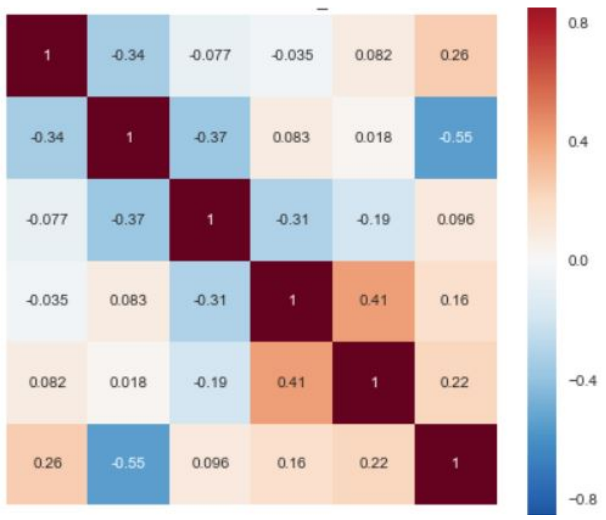
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



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

or

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



[seaborn heat maps (https://seaborn.pydata.org/generated/seaborn.heatmap.html)](https://seaborn.pydata.org/generated/seaborn.heatmap.html) with rows as **min_sample_split**, columns as **max_depth**, and values inside the cell representing **AUC Score**
- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
- Along with plotting ROC curve, you need to print the [confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points
- Once after you plot the confusion matrix with the test data, get all the `false positive data points`
  - Plot the WordCloud [WordCloud (https://www.geeksforgeeks.org/generating-word-cloud-python/)](https://www.geeksforgeeks.org/generating-word-cloud-python/)
  - 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_` (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html), 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 (http://zetcode.com/python/prettytable/)](http://zetcode.com/python/prettytable/)

2. Decision Tree

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

In [128]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your coc
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(data,y,test_size=0.30,stratify=y)
X_train,X_cv,y_train,y_cv=train_test_split(X_train,y_train,test_size=0.30,stratify=y_train)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [129]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your coc
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [130]:

```
statevectorizer=CountVectorizer()
statevectorizer.fit(X_train['school_state'].values)

X_train_state_encoded=statevectorizer.transform(X_train['school_state'].values)
X_cv_state_encoded=statevectorizer.transform(X_cv['school_state'].values)
X_test_state_encoded=statevectorizer.transform(X_test['school_state'].values)

print("AFTER VECTORIZATION")
print('='*50)
print(X_train_state_encoded.shape,y_train.shape)
print(X_cv_state_encoded.shape,y_cv.shape)
print(X_test_state_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
(49000, 51) (49000,)
(21000, 51) (21000,)
(30000, 51) (30000,)
```

In [131]:

```
X_train['teacher_prefix'].unique()
```

Out[131]:

```
array(['Mrs.', 'Mr.', 'Ms.', 'Teacher', 'Dr.', nan], dtype=object)
```

In [132]:

```
X_train['teacher_prefix'].fillna('',inplace=True)
X_cv['teacher_prefix'].fillna('',inplace=True)
X_test['teacher_prefix'].fillna('',inplace=True)
```

In [133]:

```
prefixvectorizer=CountVectorizer()
prefixvectorizer.fit(X_train['teacher_prefix'].values)

X_train_prefix_encoded=prefixvectorizer.transform(X_train['teacher_prefix'].values)
X_cv_prefix_encoded=prefixvectorizer.transform(X_cv['teacher_prefix'].values)
X_test_prefix_encoded=prefixvectorizer.transform(X_test['teacher_prefix'].values)
print('AFTER VECTORIZATION')
print('='*50)
print(prefixvectorizer.get_feature_names())

print(X_train_prefix_encoded.shape,y_train.shape)
print(X_cv_prefix_encoded.shape,y_cv.shape)
print(X_test_prefix_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
['dr', 'mr', 'mrs', 'ms', 'teacher']
(49000, 5) (49000,)
(21000, 5) (21000,)
(30000, 5) (30000,)
```

In [134]:

```
#encoding grade category
gradevectorizer=CountVectorizer()
gradevectorizer.fit(X_train['clean_grades'].values)

X_train_grade_encoded=gradevectorizer.transform(X_train['clean_grades'].values)
X_cv_grade_encoded=gradevectorizer.transform(X_cv['clean_grades'].values)
X_test_grade_encoded=gradevectorizer.transform(X_test['clean_grades'].values)

print("AFTER VECTORIZATION")
print('='*50)
print(gradevectorizer.get_feature_names())
print(X_train_grade_encoded.shape,y_train.shape)
print(X_cv_grade_encoded.shape,y_cv.shape)
print(X_test_grade_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
(49000, 4) (49000,)
(21000, 4) (21000,)
(30000, 4) (30000,)
```

In [135]:

```
#encoding clean category
categoryvectorizer=CountVectorizer()
categoryvectorizer.fit(X_train['clean_categories'].values)

X_train_category_encoded=categoryvectorizer.transform(X_train['clean_categories'].values)
X_cv_category_encoded=categoryvectorizer.transform(X_cv['clean_categories'].values)
X_test_category_encoded=categoryvectorizer.transform(X_test['clean_categories'].values)

print('AFTER VECTORIZATION')
print('='*50)
print(categoryvectorizer.get_feature_names())
print(X_train_category_encoded.shape,y_train.shape)
print(X_cv_category_encoded.shape,y_cv.shape)
print(X_test_category_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'liter
acy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
(49000, 9) (49000,)
(21000, 9) (21000,)
(30000, 9) (30000,)
```

In [136]:

```
#encoding subcategories
subcategoryvectorizer=CountVectorizer()
subcategoryvectorizer.fit(X_train['clean_subcategories'].values)
X_train_subcategories_encoded=subcategoryvectorizer.transform(X_train['clean_subcategories'
X_cv_subcategories_encoded=subcategoryvectorizer.transform(X_cv['clean_subcategories'].valu
X_test_subcategories_encoded=subcategoryvectorizer.transform(X_test['clean_subcategories'].
print("AFTER VECTORIZATION")
print('='*50)
print(subcategoryvectorizer.get_feature_names())
print(X_train_subcategories_encoded.shape,y_train.shape)
print(X_cv_subcategories_encoded.shape,y_cv.shape)
print(X_test_subcategories_encoded.shape,y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_governmen
t', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economic
s', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy',
'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness',
'history_geography', 'literacy', 'literature_writing', 'mathematics', 'musi
c', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 's
ocialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
(49000, 30) (49000,)
(21000, 30) (21000,)
(30000, 30) (30000,)
```

In [137]:

```
#encoding numerical categories---price
from sklearn.preprocessing import Normalizer
normalizer=Normalizer()
normalizer.fit(X_train['price'].values.reshape(1,-1))

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

print("after vectorization")
print(X_train_price_norm.shape,y_train.shape)
print(X_cv_price_norm.shape,y_cv.shape)
print(X_test_price_norm.shape,y_test.shape)
```

```
after vectorization
(1, 49000) (49000,)
(1, 21000) (21000,)
(1, 30000) (30000,)
```

In [138]:

```
price_train_norm=X_train_price_norm.reshape(49000,1)
price_cv_norm=X_cv_price_norm.reshape(21000,1)
price_test_norm=X_test_price_norm.reshape(30000,1)
print(price_train_norm.shape)
print(price_cv_norm.shape)
print(price_test_norm.shape)
```

```
(49000, 1)
(21000, 1)
(30000, 1)
```

In [139]:

```
#encoding numerical category quantity
normalizer=Normalizer()
normalizer.fit(X_train['quantity'].values.reshape(1,-1))

X_train_quantity_norm=normalizer.transform(X_train['quantity'].values.reshape(1,-1))
X_cv_quantity_norm=normalizer.transform(X_cv['quantity'].values.reshape(1,-1))
X_test_quantity_norm=normalizer.transform(X_test['quantity'].values.reshape(1,-1))

print('after vectorization')
print(X_train_quantity_norm.shape,y_train.shape)
print(X_cv_quantity_norm.shape,y_cv.shape)
print(X_test_quantity_norm.shape,y_test.shape)
```

```
after vectorization
(1, 49000) (49000,)
(1, 21000) (21000,)
(1, 30000) (30000,)
```

In [140]:

```
train_quantity_norm=X_train_quantity_norm.reshape(49000,1)
cv_quantity_norm=X_cv_quantity_norm.reshape(21000,1)
test_quantity_norm=X_test_quantity_norm.reshape(30000,1)
print(train_quantity_norm.shape)
print(cv_quantity_norm.shape)
print(test_quantity_norm.shape)
```

```
(49000, 1)
(21000, 1)
(30000, 1)
```

In [141]:

```
#encoding previous projects posted by teachers
normalizer=Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)

X_train_projects_norm=normalizer.transform(X_train['teacher_number_of_previously_posted_pro
X_cv_projects_norm=normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'
X_test_projects_norm=normalizer.transform(X_test['teacher_number_of_previously_posted_proje

print("after vectorization")
print(X_train_projects_norm.shape,y_train.shape)
print(X_cv_projects_norm.shape,y_cv.shape)
print(X_test_projects_norm.shape,y_test.shape)
```

```
after vectorization
(1, 49000) (49000,)
(1, 21000) (21000,)
(1, 30000) (30000,)
```

In [142]:

```
projects_train_norm=X_train_projects_norm.reshape(49000,1)
projects_cv_norm=X_cv_projects_norm.reshape(21000,1)
projects_test_norm=X_test_projects_norm.reshape(30000,1)
print(price_train_norm.shape)
print(price_cv_norm.shape)
print(price_test_norm.shape)
```

```
(49000, 1)
(21000, 1)
(30000, 1)
```

In [ ]:

```

```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [63]:

```
# please write all the code with proper documentation, and proper titles for each subsectio
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your cod
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [143]:

```
essaybowvectorizer=CountVectorizer(min_df=10,ngram_range=(1,1))
essaybowvectorizer.fit(X_train['essay'].values)
X_train_essay_bow=essaybowvectorizer.transform(X_train['essay'].values)
#print(X_train_essay_bow.shape)
X_cv_essay_bow=essaybowvectorizer.transform(X_cv["essay"].values)
X_test_essay_bow=essaybowvectorizer.transform(X_test['essay'].values)

print('AFTER VECTORIZATION')
print('='*50)
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
AFTER VECTORIZATION
==================================================
(49000, 12613) (49000,)
(21000, 12613) (21000,)
(30000, 12613) (30000,)
```

In [144]:

```
#encoding project title
titlebowvectorizer=CountVectorizer(min_df=10,ngram_range=(1,1))
titlebowvectorizer.fit(X_train['project_title'].values)
X_train_title_bow=titlebowvectorizer.transform(X_train['project_title'].values)
X_cv_title_bow=titlebowvectorizer.transform(X_cv['project_title'].values)
X_test_title_bow=titlebowvectorizer.transform(X_test['project_title'].values)
print("after vectorization")
print(X_train_title_bow.shape,y_train.shape)
print(X_cv_title_bow.shape,y_cv.shape)
print(X_test_title_bow.shape,y_test.shape)
```

```
after vectorization
(49000, 2101) (49000,)
(21000, 2101) (21000,)
(30000, 2101) (30000,)
```

In [ ]:

```

```

2.4 Appling Decision Tree on different kind of featurization as

mentioned in the instructions

Apply Decision Tree 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 [0]:

```
# please write all the code with proper documentation, and proper titles for each subsectio
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your cod
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2.4.1 Applying Decision Trees on BOW, SET 1

In [145]:

```
# Please write all the code with proper documentation
#getting final data matrix
from scipy.sparse import hstack
final_train_bow=hstack((X_train_state_encoded,X_train_prefix_encoded,X_train_grade_encoded,
final_cv_bow=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_categor
final_test_bow=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encoded,X_te
print(final_train_bow.shape,y_train.shape)
print(final_cv_bow.shape,y_cv.shape)
print(final_test_bow.shape,y_test.shape)
```

```
(49000, 14816) (49000,)
(21000, 14816) (21000,)
(30000, 14816) (30000,)
```

In [146]:

```
def enable_plotly_in_cell():
  import IPython
  from plotly.offline import init_notebook_mode
  display(IPython.core.display.HTML('''<script src="/static/components/requirejs/require.js
  init_notebook_mode(connected=False)
```

In [ ]:

```

```

In [149]:

```
#plotting error plots
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
import plotly.graph_objs as go
from sklearn.calibration import CalibratedClassifierCV


train_auc=[]
cv_auc=[]
c=[4,6, 8, 9,10,12,14,17]
min_samples=[2,10,20,30,40,50,60,70]
for i in tqdm(c):
    for j in tqdm(min_samples):
        Decision_tree=DecisionTreeClassifier(max_depth=i,class_weight='balanced',min_sample
        calib_cv=CalibratedClassifierCV(base_estimator=Decision_tree)
        calib_cv.fit(final_train_bow,y_train)

        y_tr_pred=calib_cv.predict_proba(final_train_bow)[:,1]
        y_cv_pred=calib_cv.predict_proba(final_cv_bow)[:,1]



        train_auc.append(roc_auc_score(y_train,y_tr_pred))
        cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

trace1=go.Scatter3d(x=min_samples,y=c,z=train_auc,name='TRAIN AUC')
trace2=go.Scatter3d(x=min_samples,y=c,z=cv_auc,name='CV AUC')
data=[trace1,trace2]
enable_plotly_in_cell()

layout=go.Layout(scene = dict(
        xaxis = dict(title='min_samples'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))
fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

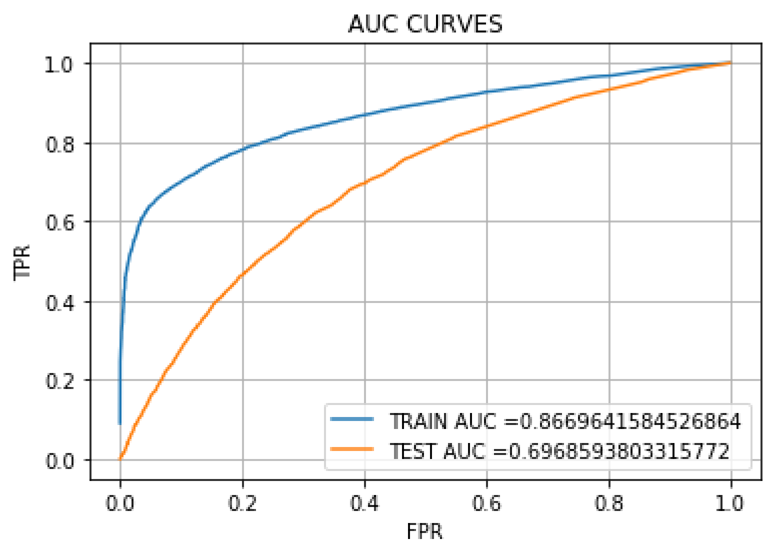
```
  0%|
| 0/8 [00:00<?, ?it/s]
  0%|
| 0/8 [00:00<?, ?it/s]
 12%|█████
| 1/8 [00:06<00:43,  6.23s/it]
 25%|██████████
| 2/8 [00:12<00:37,  6.26s/it]
 38%|███████████████
| 3/8 [00:18<00:31,  6.20s/it]
 50%|████████████████████
| 4/8 [00:24<00:24,  6.11s/it]
 62%|█████████████████████████
| 5/8 [00:31<00:18,  6.31s/it]
 75%|██████████████████████████████
| 6/8 [00:37<00:12,  6.25s/it]
 88%|█████████████████████████████████
██          | 7/8 [00:43<00:06,  6.19s/it]
```

```
12%|█████
```

OBSERVATIONS: We have plotted for different values of max depth and minimum samples split and we choose our best max depth and minimum samples split to be 9 and 10 respectively.

In [150]:

```
#plotting auc curves
from sklearn.metrics import roc_curve,auc
Decision_tree=DecisionTreeClassifier(max_depth=9,min_samples_split=10,class_weight='balance
calib_cv=CalibratedClassifierCV(base_estimator=Decision_tree)
calib_cv.fit(final_train_bow,y_train)


y_train_pred=calib_cv.predict_proba(final_train_bow)[:,1]
y_test_pred=calib_cv.predict_proba(final_test_bow)[:,1]


train_fpr,train_tpr,tr_threshold=roc_curve(y_train,y_tr_pred)
test_fpr,test_tpr,te_threshold=roc_curve(y_test,y_test_pred)

plt.plot(train_fpr,train_tpr,label='TRAIN AUC ='+str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr,label='TEST AUC ='+str(auc(test_fpr,test_tpr)))

plt.title('AUC CURVES')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.grid()
plt.show()
```



OBSERVATIONS: For max depth=9 and min samples split=10 we got train auc of 86.69 and test auc of 69.68

2.4.1.1 Graphviz visualization of Decision Tree on BOW, SET 1

In [151]:

```
feature_names=[]
feature_names.extend(statevectorizer.get_feature_names())
feature_names.extend(prefixvectorizer.get_feature_names())
feature_names.extend(gradevectorizer.get_feature_names())
feature_names.extend(categoryvectorizer.get_feature_names())
feature_names.extend(subcategoryvectorizer.get_feature_names())
feature_names.append(['price_standardized'])
feature_names.append(['projects_standardized'])
feature_names.append(['quantity_standardized'])
feature_names.extend(essaybowvectorizer.get_feature_names())
feature_names.extend(titlebowvectorizer.get_feature_names())
print(len(feature_names))
```

```
14816
```

In [152]:

```
!python -m pip install --upgrade pip
```

```
Requirement already up-to-date: pip in c:\users\hp\appdata\local\continuum\a
naconda3\lib\site-packages (19.2.3)
```

In [153]:

```
!pip install python-graphviz
```

```
Collecting python-graphviz

  ERROR: Could not find a version that satisfies the requirement python-grap
hviz (from versions: none)
ERROR: No matching distribution found for python-graphviz
```

In [157]:

```
# Please write all the code with proper documentation
from sklearn.tree import export_graphviz
import graphviz
from graphviz import Source
import pydotplus
import os
Decision_tree=DecisionTreeClassifier(max_depth=9,min_samples_split=10,class_weight='balance
Decision_tree.fit(final_train_bow,y_train)

dot_data=export_graphviz(Decision_tree,out_file=None,max_depth=3,feature_names=feature_name
graph=graphviz.Source(dot_data)
os.environ['PATH']+=os.pathsep+r'C:\Users\HP\Desktop\graphviz'
graph.render('tf-idf_tree')
graph
```

Out[157]:

```
<graphviz.files.Source at 0x247b3c39be0>
```

In [158]:

```
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t
    return t

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

In [159]:

```
#printing confusion matrix
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,train_fpr,train_tpr)
print('TRAIN CONFUSION MATRIX')
print(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
print('test confusion matrix')
print(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
```

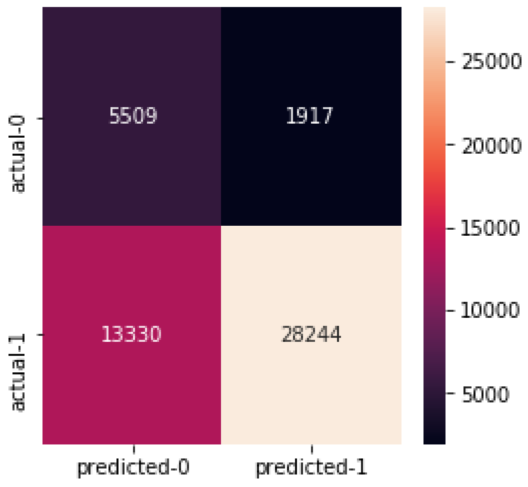
```
============================================================================
========================
the maximum value of tpr*(1-fpr) 0.6356544107340077 for threshold 0.83
TRAIN CONFUSION MATRIX
[[ 5754  1672]
 [14048 27526]]
test confusion matrix
[[ 3201  1345]
 [10413 15041]]
```

In [160]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[5509,1917],[13330,28244]]

train=pd.DataFrame(array,columns=['predicted-0','predicted-1'],index=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[160]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x247b40b75c0>
```



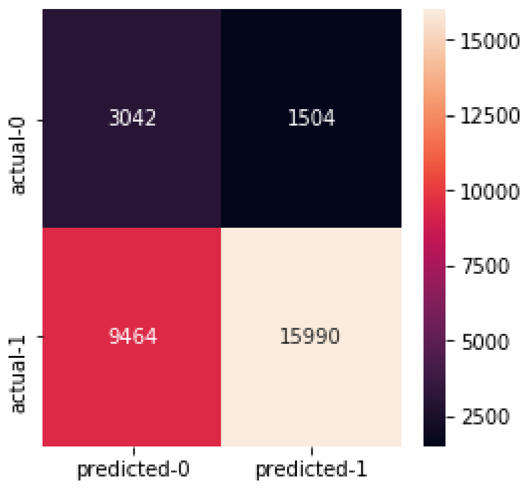
OBSERVATION: We got decent values for tnr and tpr. The tnr value is a bit high.

In [161]:

```
#printing heatmap for test confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[3042,1504],[9464,15990]]

train=pd.DataFrame(array,columns=['predicted-0','predicted-1'],index=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[161]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x247b4018780>
```



OBSERVATIONS: For test confusion matrix we got a decent tnr and tpr values.

In [162]:

```
np.round(y_test_pred[0:10])
```

Out[162]:

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [163]:

```
z['y_actual'].value_counts()
```

Out[163]:

```
1    25454
0     4546
Name: y_actual, dtype: int64
```

In [164]:

```
z=X_test
z.shape
z['y_actual']=y_test
z['y_hat']=np.round(y_test_pred)
z.shape
```

Out[164]:

```
(30000, 15)
```

In [165]:

```
fpr=z[(z.y_actual==0)&(z.y_hat==1)]
fpr.shape
```

Out[165]:

```
(4546, 15)
```

In [173]:

```
#taking only 1k points as it was taking long time to run
fpr1=fpr[0:1000]
fpr1.shape
```

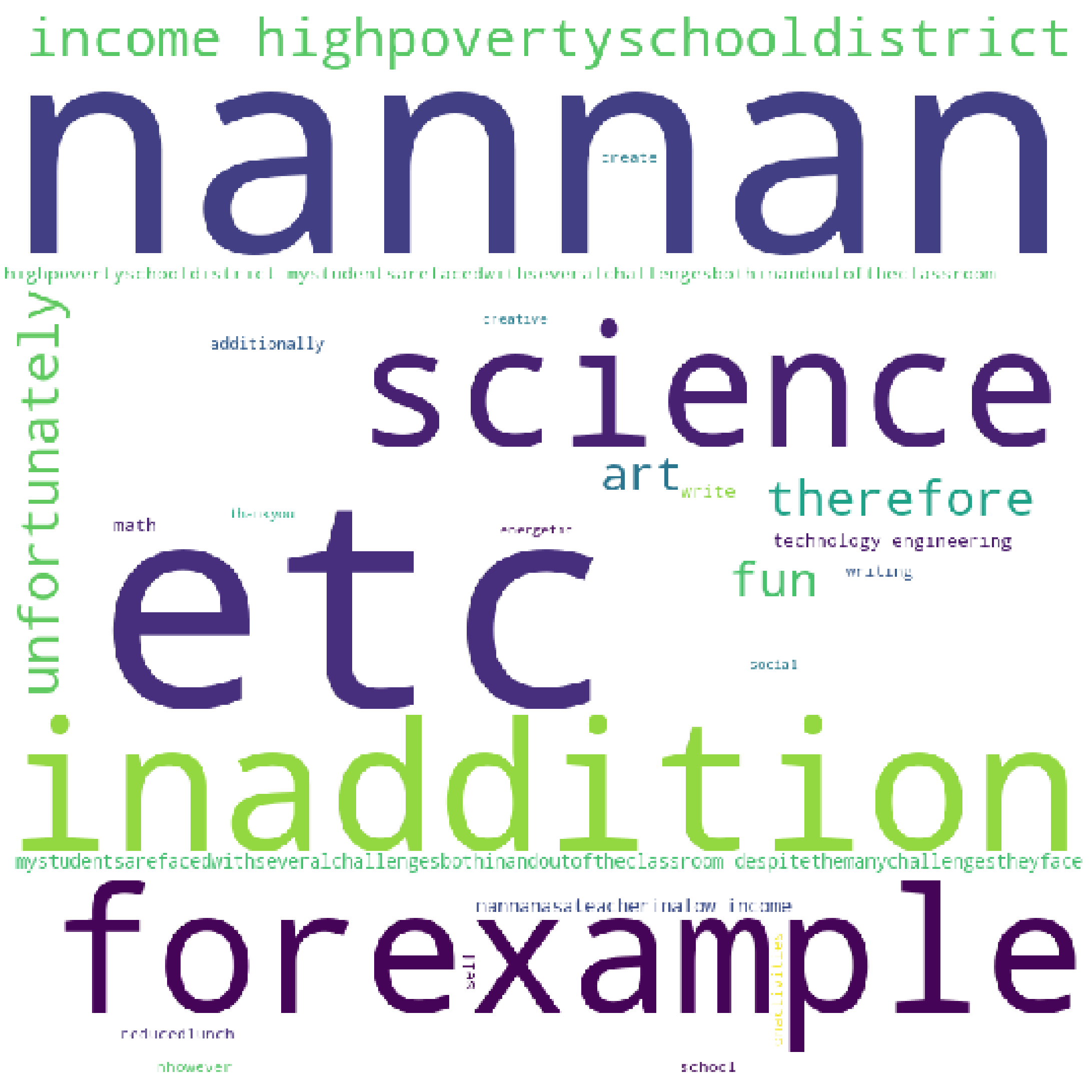
Out[173]:

```
(1000, 15)
```

WORD CLOUD FOR THE FALSE POSITIVE VALUES.

In [174]:

```
import matplotlib.pyplot as plt
import pandas as pd
import tqdm
from wordcloud import WordCloud,STOPWORDS

originalwords=''
stopwords=set(STOPWORDS)
for sent in fpr1['essay']:
    sent=str(sent)
    words=sent.split()
    for i in range(len(words)):
        words[i]=words[i].lower()
    for word in words:
        originalwords=originalwords+word+''
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(originalwords)
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



In [84]:

```
#box plot of price
fpr.columns
```

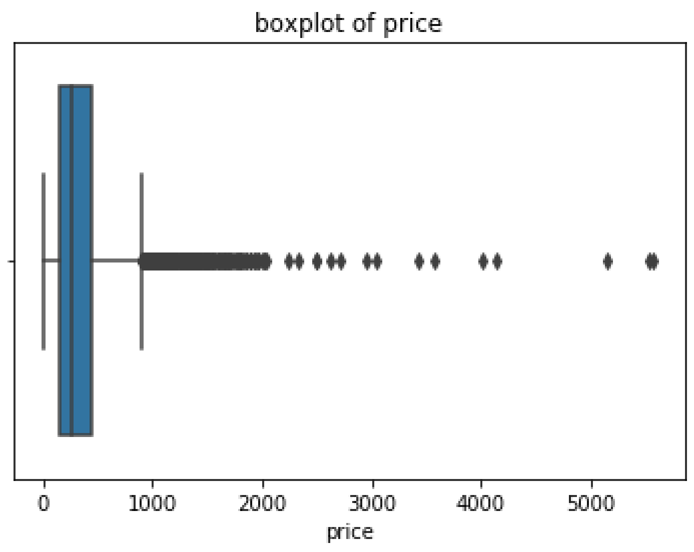
Out[84]:

```
Index(['Unnamed: 0', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_title',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_grades', 'clean_subcategories', 'essay', 'price', 'quantity',
       'y_actual', 'y_hat'],
      dtype='object')
```

BOX PLOT OF PRICE

In [175]:

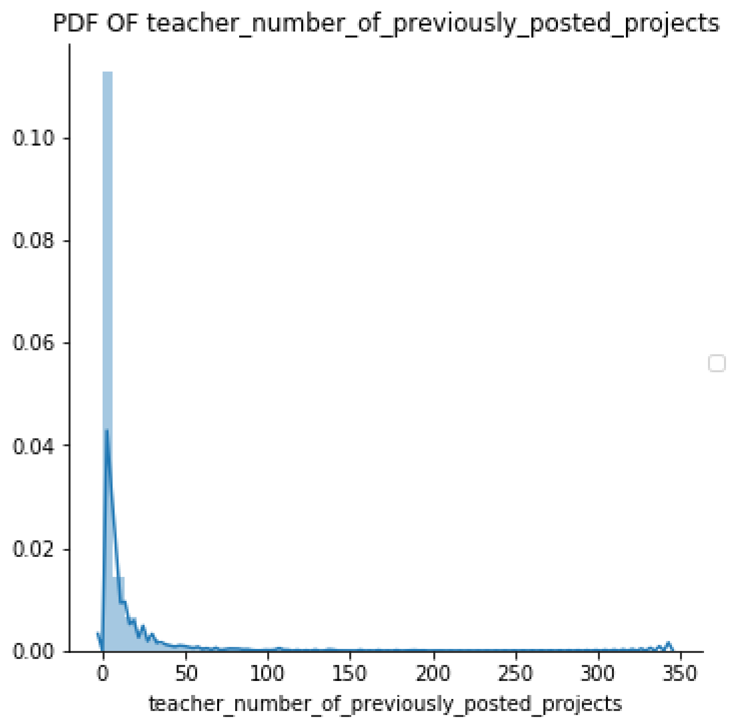
```
import seaborn as sns
sns.boxplot(x='price',data=fpr)
plt.title('boxplot of price')
plt.show()
```



PDF OF NUMBER OF PREVIOUSLY POSTED PROJECTS

In [176]:

```
#pdf of number of projects posted by teachers
sns.FacetGrid(fpr,size=5)\
   .map(sns.distplot,'teacher_number_of_previously_posted_projects')\
   .add_legend();
plt.title('PDF OF teacher_number_of_previously_posted_projects')
plt.show();
```



In [ ]:

```
```

In [ ]:

```
```

2.4.2 Applying Decision Trees on TFIDF, SET 2

In [0]:

```
# Please write all the code with proper documentation
```

In [177]:

```
#tfidf encoding  of text
from sklearn.feature_extraction.text import TfidfVectorizer
essaytfidfvectorizer = TfidfVectorizer(min_df=10)
essaytfidfvectorizer.fit(X_train['essay'].values)
train_tfidf=essaytfidfvectorizer.transform(X_train['essay'].values)
cv_tfidf=essaytfidfvectorizer.transform(X_cv['essay'].values)
test_tfidf=essaytfidfvectorizer.transform(X_test['essay'].values)
print("Shape of matrix after one hot encodig ",train_tfidf.shape)
print("Shape of matrix after one hot encodig ",cv_tfidf.shape)
print("Shape of matrix after one hot encodig ",test_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (49000, 12613)
Shape of matrix after one hot encodig  (21000, 12613)
Shape of matrix after one hot encodig  (30000, 12613)
```

In [178]:

```
#tfidf encoding of title
titletfidfvectorizer=TfidfVectorizer(min_df=10)
titletfidfvectorizer.fit(X_train['project_title'].values)
title_train_tfidf=titletfidfvectorizer.transform(X_train['project_title'].values)
title_cv_tfidf=titletfidfvectorizer.transform(X_cv['project_title'].values)
title_test_tfidf=titletfidfvectorizer.transform(X_test['project_title'].values)

print("Shape of matrix after one hot encodig ",title_train_tfidf.shape)
print("Shape of matrix after one hot encodig ",title_cv_tfidf.shape)
print("Shape of matrix after one hot encodig ",title_test_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (49000, 2101)
Shape of matrix after one hot encodig  (21000, 2101)
Shape of matrix after one hot encodig  (30000, 2101)
```

In [179]:

```
from scipy.sparse import hstack
final_train_tfidf=hstack((X_train_state_encoded,X_train_prefix_encoded,X_train_grade_encode
final_cv_tfidf=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_categ
final_test_tfidf=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encoded,X_
print(final_train_tfidf.shape,y_train.shape)
print(final_cv_tfidf.shape,y_cv.shape)
print(final_test_tfidf.shape,y_test.shape)
```

```
(49000, 14816) (49000,)
(21000, 14816) (21000,)
(30000, 14816) (30000,)
```

In [180]:

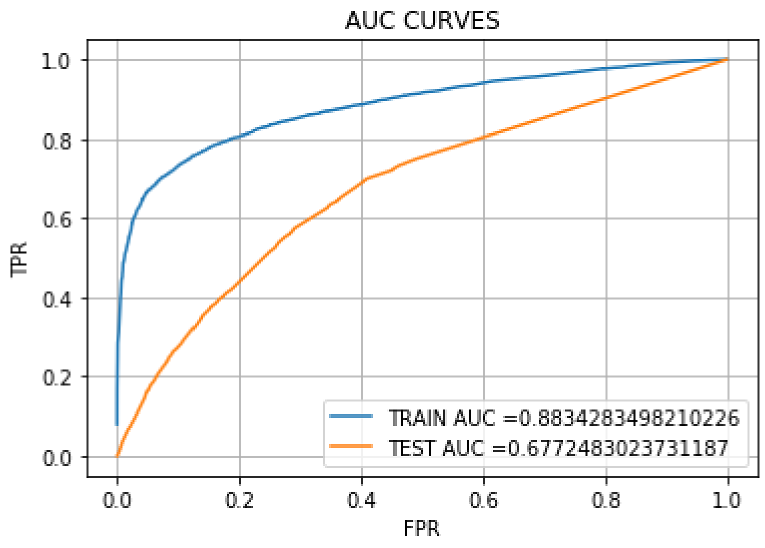
```
#plotting error plots
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
import plotly.graph_objs as go
from tqdm import tqdm

train_auc=[]
cv_auc=[]
depth=[4,6,8,9,10,12,14,17]
min_samples=[2,10,20,30,40,50,60,70]
for i in tqdm(depth):
    for j in tqdm(min_samples):
        tfidf_Decision_tree=DecisionTreeClassifier(max_depth=i,class_weight='balanced',min_
        calib_cv=CalibratedClassifierCV(base_estimator=tfidf_Decision_tree)
        calib_cv.fit(final_train_tfidf,y_train)

        y_tr_pred=calib_cv.predict_proba(final_train_tfidf)[:,1]
        y_cv_pred=calib_cv.predict_proba(final_cv_tfidf)[:,1]

        train_auc.append(roc_auc_score(y_train,y_tr_pred))
        cv_auc.append(roc_auc_score(y_cv,y_cv_pred))
        print('for {} max depth and {} min samples the train_auc={}'.format(i,j,roc_auc_sco
        print('for {} max depth and {} min samples the cv_auc={}'.format(i,j,roc_auc_score(


trace1=go.Scatter3d(x=min_samples,y=depth,z=train_auc,name='TRAIN AUC')
trace2=go.Scatter3d(x=min_samples,y=depth,z=cv_auc,name='CV AUC')
data=[trace1,trace2]
enable_plotly_in_cell()

layout=go.Layout(scene = dict(
        xaxis = dict(title='min_samples'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))
fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

```
  0%|
| 0/8 [00:00<?, ?it/s]
  0%|
| 0/8 [00:00<?, ?it/s]

for 4 max depth and 2 min samples the train_auc=0.6751862665595485
for 4 max depth and 2 min samples the cv_auc=0.6485219309085037


 12%|
| 1/8 [00:13<01:31, 13.01s/it]

for 4 max depth and 10 min samples the train_auc=0.6751862665595485
for 4 max depth and 10 min samples the cv_auc=0.6485219309085037


 25%|
| 2/8 [00:28<01:22, 13.72s/it]
```

OBSERVATIONS: We have plotted for different values of max depth and minimum samples split. We choose our max depth to be 6 and min samples split to be 10.

In [181]:

```
#plotting auc curves
from sklearn.metrics import roc_curve,auc
tfidf_Decision_tree=DecisionTreeClassifier(max_depth=6,min_samples_split=10,class_weight='b
calib_cv=CalibratedClassifierCV(base_estimator=tfidf_Decision_tree)
calib_cv.fit(final_train_tfidf,y_train)

y_train_pred=calib_cv.predict_proba(final_train_tfidf)[:,1]
y_test_pred=calib_cv.predict_proba(final_test_tfidf)[:,1]

train_fpr,train_tpr,tr_threshold=roc_curve(y_train,y_tr_pred)
test_fpr,test_tpr,te_threshold=roc_curve(y_test,y_test_pred)

plt.plot(train_fpr,train_tpr,label='TRAIN AUC ='+str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr,label='TEST AUC ='+str(auc(test_fpr,test_tpr)))

plt.title('AUC CURVES')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.grid()
plt.show()
```



OBSERVATIONS: For max depth=6 and min sample split=10 we got train auc of 88.34 and test auc of 67.72.

In [182]:

```
#printing confusion matrix
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,train_fpr,train_tpr)
print('TRAIN CONFUSION MATRIX')
print(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
print('test confusion matrix')
print(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
```

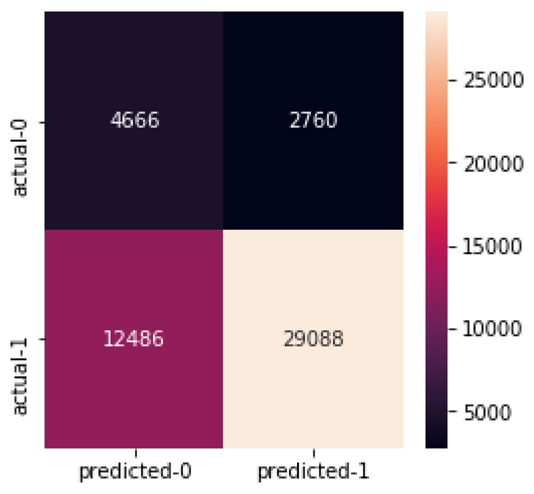
```
============================================================================
========================
the maximum value of tpr*(1-fpr) 0.6631577521486158 for threshold 0.829
TRAIN CONFUSION MATRIX
[[ 4666  2760]
 [12486 29088]]
test confusion matrix
[[ 2898  1648]
 [ 9014 16440]]
```

In [184]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[4666,2760],[12486,29088]]

train=pd.DataFrame(array,columns=['predicted-0','predicted-1'],index=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[184]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x247c2898b70>
```

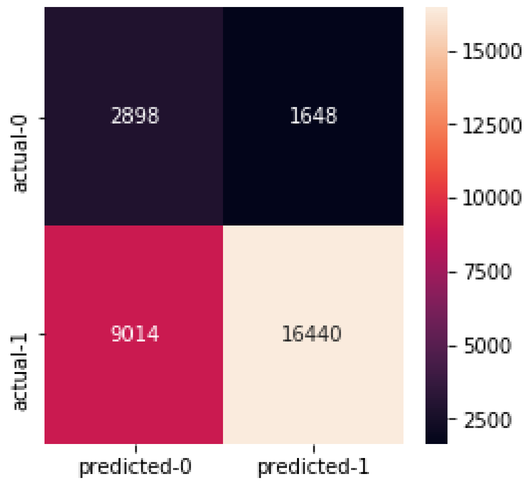


In [185]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[2898,1648],[9014,16440]]

train=pd.DataFrame(array,columns=['predicted-0','predicted-1'],index=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[185]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24782487c88>
```



2.4.2.1 Graphviz visualization of Decision Tree on TFIDF, SET 2

In [71]:

```
# Please write all the code with proper documentation
```

In [186]:

```
feature_names_tfidf=[]
feature_names_tfidf.extend(statevectorizer.get_feature_names())
feature_names_tfidf.extend(prefixvectorizer.get_feature_names())
feature_names_tfidf.extend(gradevectorizer.get_feature_names())
feature_names_tfidf.extend(categoryvectorizer.get_feature_names())
feature_names_tfidf.extend(subcategoryvectorizer.get_feature_names())
feature_names_tfidf.append(['price_standardized'])
feature_names_tfidf.append(['projects_standardized'])
feature_names_tfidf.append(['quantity_standardized'])
feature_names_tfidf.extend(essaytfidfvectorizer.get_feature_names())
feature_names_tfidf.extend(titletfidfvectorizer.get_feature_names())
print(len(feature_names_tfidf))
```

```
14816
```

In [187]:

```
# Please write all the code with proper documentation
from sklearn.tree import export_graphviz
import graphviz
from graphviz import Source
import pydotplus
import os
tfidf_Decision_tree=DecisionTreeClassifier(max_depth=9,min_samples_split=10,class_weight='b
tfidf_Decision_tree.fit(final_train_tfidf,y_train)

dot_data=export_graphviz(tfidf_Decision_tree,out_file=None,max_depth=3,feature_names=featur
graph=graphviz.Source(dot_data)
os.environ['PATH']+=os.pathsep+r'C:\Users\HP\Desktop\graphviz'
graph.render('tf-idf_tree_2')
graph
```

Out[187]:

```
<graphviz.files.Source at 0x2478300e240>
```

In [188]:

```
y_test_pred[0:10]
```

Out[188]:

```
array([0.9150087 , 0.9399745 , 0.74899667, 0.93749374, 0.83529741,
       0.93983735, 0.84992608, 0.74899667, 0.74899667, 0.83529741])
```

In [189]:

```
b=X_test
b.shape
b['y_actual']=y_test
b['y_hat']=np.round(y_test_pred)
b.shape
```

Out[189]:

```
(30000, 15)
```

In [190]:

```
b['y_hat'].value_counts()
```

Out[190]:

```
1.0    30000
Name: y_hat, dtype: int64
```

In [191]:

```
b['y_actual'].value_counts()
```

Out[191]:

```
1    25454
0     4546
Name: y_actual, dtype: int64
```

In [194]:

```
fpr_tfidf=b[(b.y_actual==0)&(b.y_hat==1)]
fpr1_tfidf=fpr_tfidf[0:1000]
fpr1_tfidf.shape
```

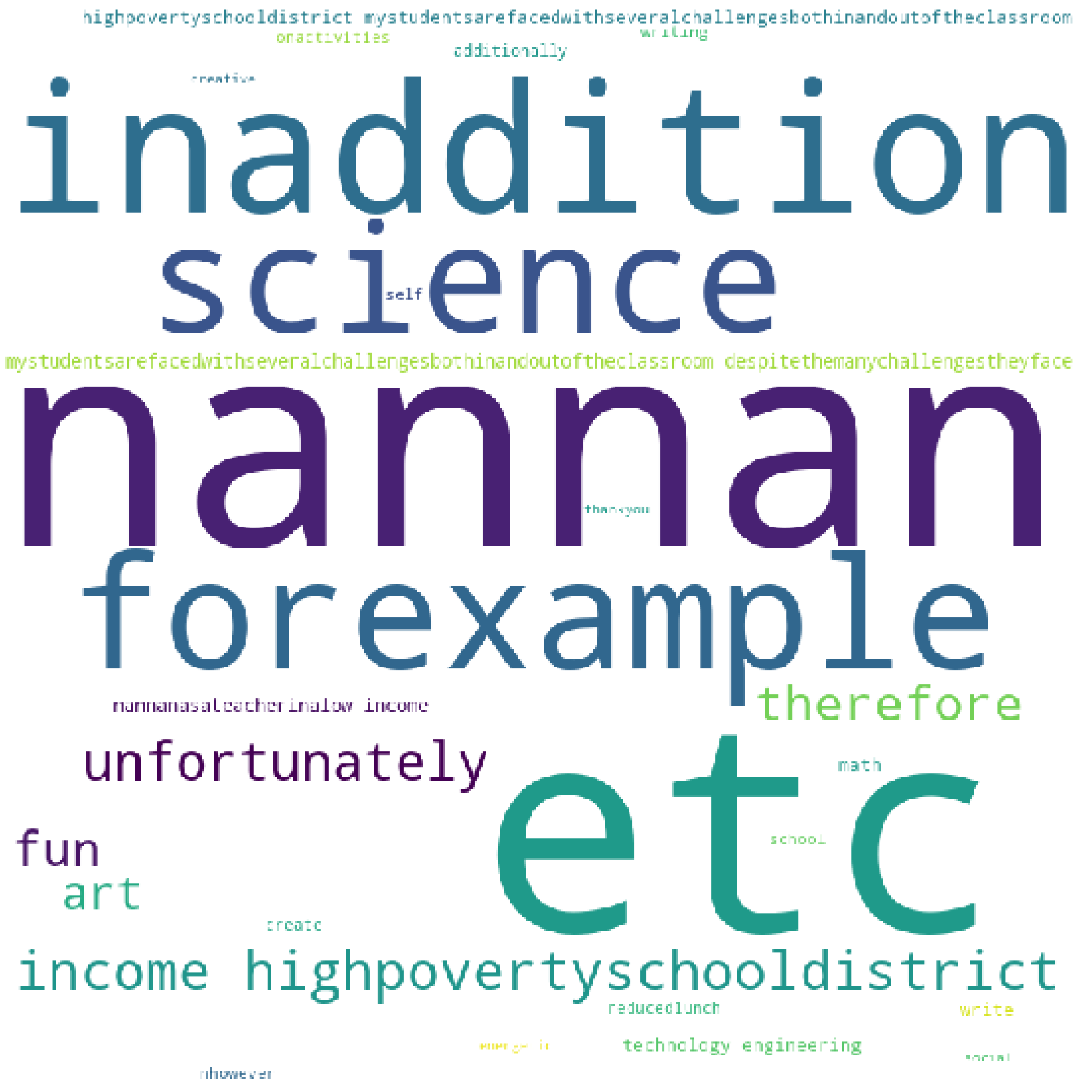
Out[194]:

```
(1000, 15)
```

WORD CLOUD FOR FALSE POSITIVES

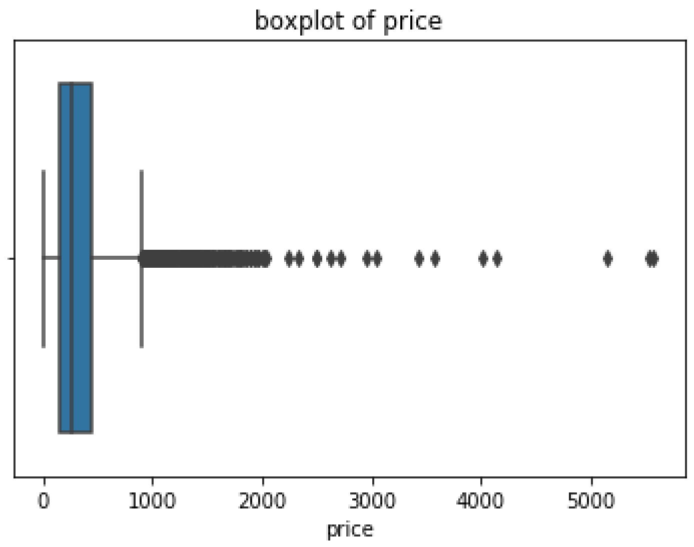
In [195]:

```
import matplotlib.pyplot as plt
import pandas as pd
import tqdm
from wordcloud import WordCloud,STOPWORDS

originalwords_tfidf=''
stopwords=set(STOPWORDS)
for sent in fpr1_tfidf['essay']:
    sent=str(sent)
    words=sent.split()
    for i in range(len(words)):
        words[i]=words[i].lower()
    for word in words:
        originalwords_tfidf=originalwords_tfidf+word+''
wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(originalwords_tfidf)
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



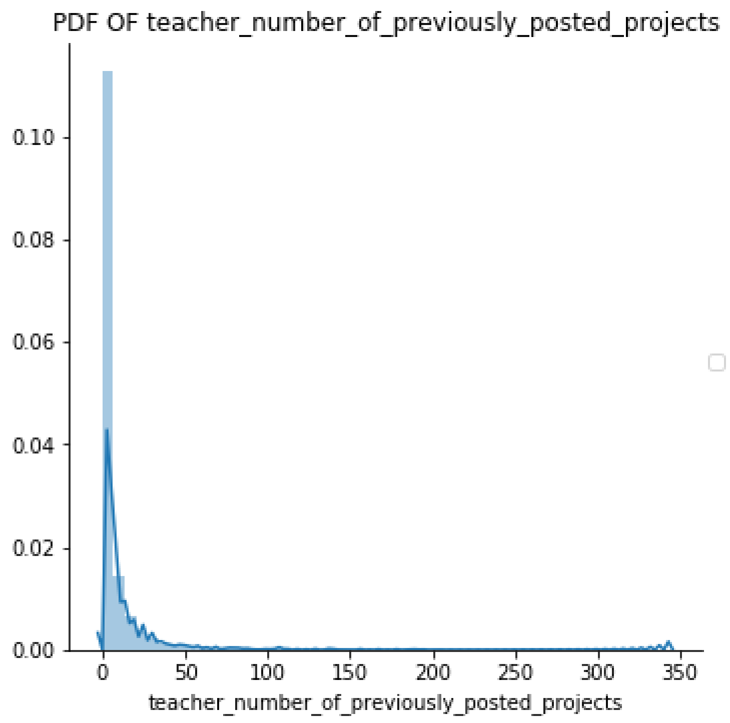
In [196]:

```
import seaborn as sns
sns.boxplot(x='price',data=fpr_tfidf)
plt.title('boxplot of price')
plt.show()
```



In [197]:

```
#pdf of number of projects posted by teachers
sns.FacetGrid(fpr_tfidf,size=5)\
   .map(sns.distplot,'teacher_number_of_previously_posted_projects')\
   .add_legend();
plt.title('PDF OF teacher_number_of_previously_posted_projects')
plt.show();
```



2.4.3 Applying Decision Trees on AVG W2V, SET 3

In [198]:

```
# Please write all the code with proper documentation
```

In [199]:

```
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [201]:

```
# average Word2Vec
# compute average word2vec for each review.
from tqdm import tqdm
avg_w2v_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    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
    avg_w2v_train.append(vector)

print(len(avg_w2v_train))
print(len(avg_w2v_train[0]))
#print(avg_w2v_train[0])
```

```
100%|████████████████████████████████████████████████████████████████████
███| 49000/49000 [00:17<00:00, 2862.12it/s]

49000
300
```

In [202]:

```
avg_w2v_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay'].values): # 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
    avg_w2v_cv.append(vector)

print(len(avg_w2v_cv))
print(len(avg_w2v_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
███| 21000/21000 [00:06<00:00, 3081.85it/s]

21000
300
```

In [203]:

```
avg_w2v_test =[];
for sentence in tqdm(X_test['essay'].values):
    vector=np.zeros(300)
    cnt_words = 0;
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words +=1
    if cnt_words != 0:
            vector /= cnt_words
    avg_w2v_test.append(vector)
print(len(avg_w2v_test))
```

```
100%|████████████████████████████████████████████████████████████████████████
███| 30000/30000 [00:09<00:00, 3136.93it/s]

30000
```

In [204]:

```
title_train_avgw2v=[]
for sentence in tqdm(X_train['project_title'].values):
    vector=np.zeros(300)
    cnt_words=0;
    for word in sentence.split():
        if word in glove_words:
            vector+=model[word]
            cnt_words+=1
    if cnt_words !=0:
        vector/=cnt_words
    title_train_avgw2v.append(vector)
print(len(title_train_avgw2v))
print(len(title_train_avgw2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████
█| 49000/49000 [00:00<00:00, 100674.28it/s]

49000
300
```

In [205]:

```
title_cv_avgw2v=[]
for sentence in tqdm(X_cv['project_title'].values):
    vector=np.zeros(300)
    cnt_words=0;
    for word in sentence.split():
        if word in glove_words:
            vector+=model[word]
            cnt_words+=1
    if cnt_words !=0:
        vector/=cnt_words
    title_cv_avgw2v.append(vector)

print(len(title_cv_avgw2v))
print(len(title_cv_avgw2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█| 21000/21000 [00:00<00:00, 100535.76it/s]

21000
300
```

In [206]:

```
title_test_avgw2v=[]
for sentence in tqdm(X_test['project_title'].values):
    vector=np.zeros(300)
    cnt_ords=0;
    for word in sentence.split():
        if word in glove_words:
            vector+=model[word]
            cnt_words+=1
    if cnt_words !=0:
        vector/=cnt_words
    title_test_avgw2v.append(vector)
print(len(title_test_avgw2v))
```

```
100%|████████████████████████████████████████████████████████████████████
██| 30000/30000 [00:00<00:00, 63596.09it/s]

30000
```

In [207]:

```
from scipy.sparse import hstack
final_train_avgw2v=hstack((X_train_state_encoded,X_train_prefix_encoded,X_train_grade_encod
final_cv_avgw2v=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_cate
final_test_avgw2v=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encoded,X
print(final_train_avgw2v.shape,y_train.shape)
print(final_cv_avgw2v.shape,y_cv.shape)
print(final_test_avgw2v.shape,y_test.shape)
```

```
(49000, 702) (49000,)
(21000, 702) (21000,)
(30000, 702) (30000,)
```

In [209]:

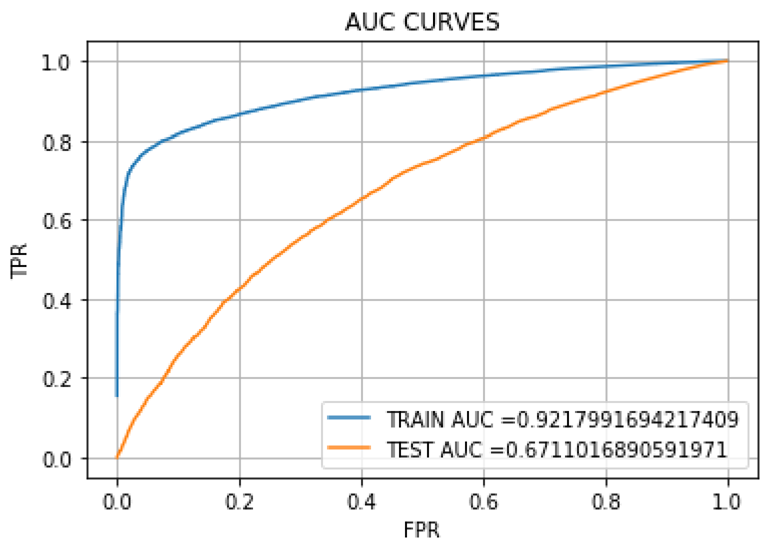
```
#plotting error plots
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
import plotly.graph_objs as go
from tqdm import tqdm

train_auc=[]
cv_auc=[]
depth=[4,6,8,9,10,12,14,17]
min_samples=[2,10,20,30,40,50,60,70]
for i in tqdm(depth):
    for j in tqdm(min_samples):
        Decision_tree=DecisionTreeClassifier(max_depth=i,class_weight='balanced',min_sample
        calib_cv=CalibratedClassifierCV(base_estimator=Decision_tree)
        calib_cv.fit(final_train_avgw2v,y_train)

        y_tr_pred=calib_cv.predict_proba(final_train_avgw2v)[:,1]
        y_cv_pred=calib_cv.predict_proba(final_cv_avgw2v)[:,1]



        train_auc.append(roc_auc_score(y_train,y_tr_pred))
        cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

        print('for {} max depth and {} min samples the train_auc={}'.format(i,j,roc_auc_sco
        print('for {} max depth and {} min samples the cv_auc={}'.format(i,j,roc_auc_score(

trace1=go.Scatter3d(x=min_samples,y=depth,z=train_auc,name='TRAIN AUC')
trace2=go.Scatter3d(x=min_samples,y=depth,z=cv_auc,name='CV AUC')
data=[trace1,trace2]
enable_plotly_in_cell()

layout=go.Layout(scene = dict(
        xaxis = dict(title='min_samples'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))
fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

```
  0%|
| 0/8 [00:00<?, ?it/s]
  0%|
| 0/8 [00:00<?, ?it/s]



for 4 max depth and 2 min samples the train_auc=0.6832056713360246
for 4 max depth and 2 min samples the cv_auc=0.6509459621725895


 12%|█████
| 1/8 [00:29<03:28, 29.84s/it]

for 4 max depth and 10 min samples the train_auc=0.6832056713360246
for 4 max depth and 10 min samples the cv_auc=0.6509459621725895
```

OBSERVATIONS: We have plotted for different values of max depth and min sample split. We choose max depth to be 6 and min samples split to be 10.

In [210]:

```
#plotting auc curves
from sklearn.metrics import roc_curve,auc
Decision_tree=DecisionTreeClassifier(max_depth=6,min_samples_split=10,class_weight='balance
calib_cv=CalibratedClassifierCV(base_estimator=Decision_tree)
calib_cv.fit(final_train_avgw2v,y_train)

y_train_pred=calib_cv.predict_proba(final_train_avgw2v)[:,1]
y_test_pred=calib_cv.predict_proba(final_test_avgw2v)[:,1]

train_fpr,train_tpr,tr_threshold=roc_curve(y_train,y_tr_pred)
test_fpr,test_tpr,te_threshold=roc_curve(y_test,y_test_pred)

plt.plot(train_fpr,train_tpr,label='TRAIN AUC ='+str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr,label='TEST AUC ='+str(auc(test_fpr,test_tpr)))

plt.title('AUC CURVES')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.grid()
plt.show()
```



OBSERVATIONS: For max depth=6 and min samples split=10 we got train auc of 92.17 and test auc of 67.11.

In [211]:

```
#printing confusion matrix
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,train_fpr,train_tpr)
print('TRAIN CONFUSION MATRIX')
print(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
print('test confusion matrix')
print(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
```

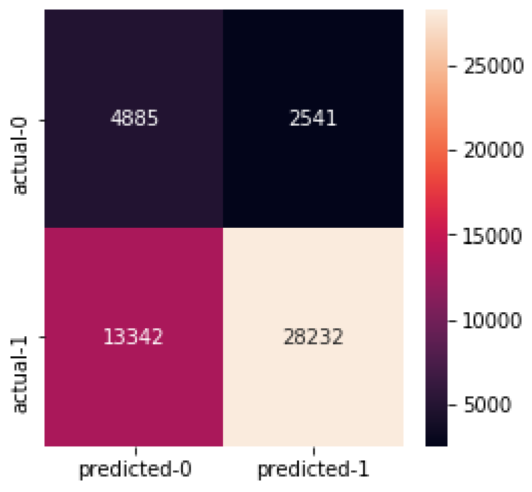
```
===========================================================================
========================
the maximum value of tpr*(1-fpr) 0.7397290248438463 for threshold 0.834
TRAIN CONFUSION MATRIX
[[ 4885  2541]
 [13342 28232]]
test confusion matrix
[[ 2989  1557]
 [10284 15170]]
```

In [212]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[4885,2541],[13342,28232]]

train=pd.DataFrame(array,columns=['predicted-0','predicted-1'],index=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[212]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x247b3cad128>
```



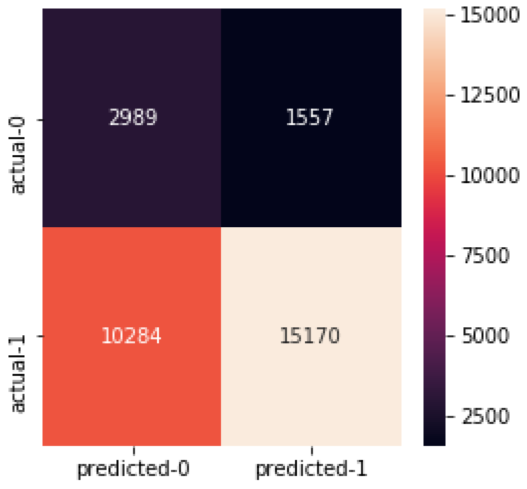
OBSERVATIONS: We got a decent tnr and tpr values.

In [213]:

```
#printing heatmap for test confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[2989,1557],[10284,15170]]

train=pd.DataFrame(array,columns=['predicted-0','predicted-1'],index=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[213]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x247c4433160>
```



OBSERVATIONS: We got a decent tpr and tnr values.The value of tnr is a bit high.

2.4.4 Applying Decision Trees on TFIDF W2V, SET 4

In [214]:

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

```
100%|████████████████████████████████████████████████████████████████████
████| 49000/49000 [01:08<00:00, 716.36it/s]
```

In [215]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays_train)
# 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 [216]:

```
# compute average word2vec for each review.
tfidf_w2v_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in (X_train['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((sentenc
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # gettir
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_train.append(vector)

print(len(tfidf_w2v_train))
print(len(tfidf_w2v_train[0]))
```

```
49000
300
```

In [217]:

```
from tqdm import tqdm
```

In [218]:

```
tfidf_w2v_cv=[]
for sentence in tqdm(X_cv['essay']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector +=(vec * tf_idf)
            tf_idf_weight+= tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    tfidf_w2v_cv.append(vector)

print(len(tfidf_w2v_cv))
print(len(tfidf_w2v_cv[0]))
```

```
100%|████████████████████████████████████████
████| 21000/21000 [01:34<00:00, 223.19it/s]

21000
300
```

In [219]:

```
tfidf_w2v_test=[]
for sentence in tqdm(X_test['essay']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if (word in glove_words)and(word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector+=(vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    tfidf_w2v_test.append(vector)
print(len(tfidf_w2v_test))
print(len(tfidf_w2v_test[0]))
```

```
100%|████████████████████████████████████████
████| 30000/30000 [01:55<00:00, 260.66it/s]

30000
300
```

In [220]:

```
#tfidf weighted w2v vectorizing project title
title_train_tfidfw2v=[]
for sentence in tqdm(X_train['project_title']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if(word in glove_words)and(word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector+=(vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    title_train_tfidfw2v.append(vector)
print(len(title_train_tfidfw2v))
print(len(title_train_tfidfw2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█| 49000/49000 [00:01<00:00, 31395.63it/s]

49000
300
```

In [221]:

```
title_cv_tfidfw2v=[]
for sentence in tqdm(X_cv['project_title']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if (word in glove_words) and (word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector+=(vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    title_cv_tfidfw2v.append(vector)
print(len(title_cv_tfidfw2v))
print(len(title_cv_tfidfw2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█| 21000/21000 [00:00<00:00, 39605.95it/s]

21000
300
```

In [222]:

```
title_test_tfidfw2v=[]
for sentence in tqdm(X_test['project_title']):
    vector=np.zeros(300)
    tf_idf_weight=0;
    for word in sentence.split():
        if (word in glove_words)and(word in tfidf_words):
            vec=model[word]
            tf_idf=dictionary[word]*(sentence.count(word)/len(sentence.split()))
            vector+=(vec*tf_idf)
            tf_idf_weight+=tf_idf
    if tf_idf_weight !=0:
        vector /=tf_idf_weight
    title_test_tfidfw2v.append(vector)

print(len(title_test_tfidfw2v))
print(len(title_test_tfidfw2v[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
█| 30000/30000 [00:00<00:00, 42361.03it/s]

30000
300
```

In [223]:

```
#creating data matrix
from scipy.sparse import hstack
final_train_tfidfw2v=hstack((X_train_state_encoded,X_train_prefix_encoded,X_train_grade_enc
final_cv_tfidfw2v=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_ca
final_test_tfidfw2v=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encodec
print(final_train_tfidfw2v.shape,y_train.shape)
print(final_cv_tfidfw2v.shape,y_cv.shape)
print(final_test_tfidfw2v.shape,y_test.shape)
```

```
(49000, 702) (49000,)
(21000, 702) (21000,)
(30000, 702) (30000,)
```

In [224]:

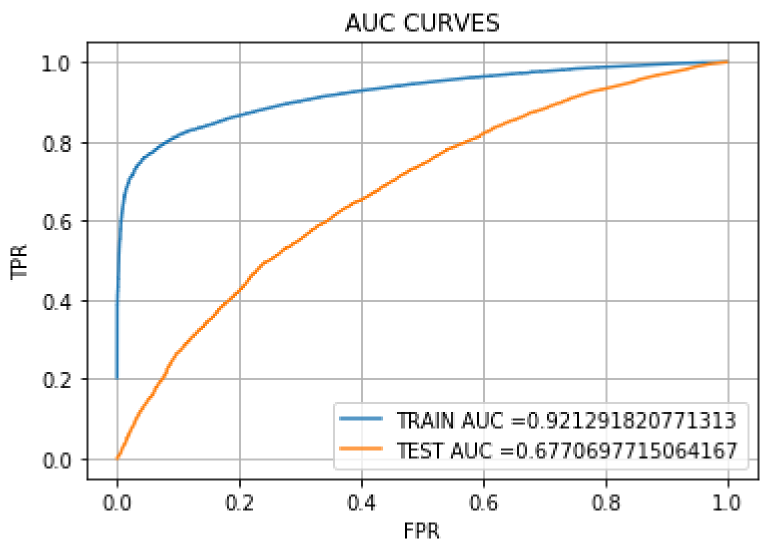
```
#plotting error plots
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
import plotly.graph_objs as go
from tqdm import tqdm

train_auc=[]
cv_auc=[]
depth=[4,6,8,9,10,12,14,17]
min_samples=[2,10,20,30,40,50,60,70]
for i in tqdm(depth):
    for j in tqdm(min_samples):
        Decision_tree=DecisionTreeClassifier(max_depth=i,class_weight='balanced',min_sample
        calib_cv=CalibratedClassifierCV(base_estimator=Decision_tree)
        calib_cv.fit(final_train_tfidfw2v,y_train)

        y_tr_pred=calib_cv.predict_proba(final_train_tfidfw2v)[:,1]
        y_cv_pred=calib_cv.predict_proba(final_cv_tfidfw2v)[:,1]


        train_auc.append(roc_auc_score(y_train,y_tr_pred))
        cv_auc.append(roc_auc_score(y_cv,y_cv_pred))

        print('for {} max depth and {} min samples the train_auc={}'.format(i,j,roc_auc_scc
        print('for {} max depth and {} min samples the cv_auc={}'.format(i,j,roc_auc_score(

trace1=go.Scatter3d(x=min_samples,y=depth,z=train_auc,name='TRAIN AUC')
trace2=go.Scatter3d(x=min_samples,y=depth,z=cv_auc,name='CV AUC')
data=[trace1,trace2]
enable_plotly_in_cell()

layout=go.Layout(scene = dict(
        xaxis = dict(title='min_samples'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))
fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

```
  0%|
| 0/8 [00:00<?, ?it/s]
  0%|
| 0/8 [00:00<?, ?it/s]

for 4 max depth and 2 min samples the train_auc=0.6860884467545991
for 4 max depth and 2 min samples the cv_auc=0.6566151281421572


 12%|████
| 1/8 [00:38<04:30, 38.70s/it]

for 4 max depth and 10 min samples the train_auc=0.6860884467545991
for 4 max depth and 10 min samples the cv_auc=0.6566151281421572


 25%|████████
| 2/8 [01:14<03:47, 37.86s/it]
```

OBSERVATIONS: Here we plotted for different values of max depth and min samples split. We choose our best max depth to be 8 and min samples split to be 10.

In [225]:

```
#plotting auc curves
from sklearn.metrics import roc_curve,auc
Decision_tree=DecisionTreeClassifier(max_depth=8,min_samples_split=10,class_weight='balance
calib_cv=CalibratedClassifierCV(base_estimator=Decision_tree)
calib_cv.fit(final_train_tfidfw2v,y_train)

y_train_pred=calib_cv.predict_proba(final_train_tfidfw2v)[:,1]
y_test_pred=calib_cv.predict_proba(final_test_tfidfw2v)[:,1]

train_fpr,train_tpr,tr_threshold=roc_curve(y_train,y_tr_pred)
test_fpr,test_tpr,te_threshold=roc_curve(y_test,y_test_pred)

plt.plot(train_fpr,train_tpr,label='TRAIN AUC ='+str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr,label='TEST AUC ='+str(auc(test_fpr,test_tpr)))

plt.title('AUC CURVES')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.grid()
plt.show()
```



OBSERVATIONS: For max depth=8 and min samples split=10 we got train auc of 92.12 and test auc of 67.70.

In [226]:

```
#printing confusion matrix
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,train_fpr,train_tpr)
print('TRAIN CONFUSION MATRIX')
print(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
print('test confusion matrix')
print(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
```

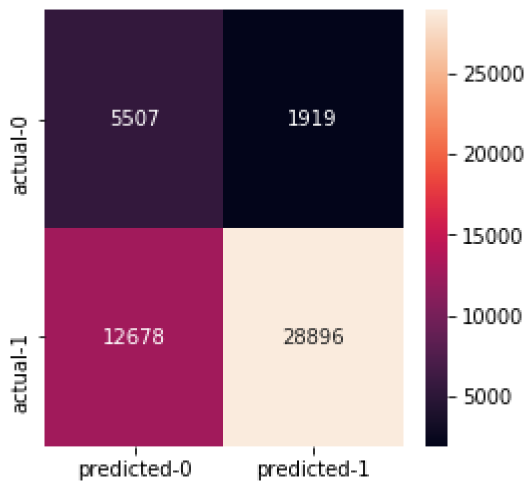
```
============================================================================
========================
the maximum value of tpr*(1-fpr) 0.7346660070839455 for threshold 0.834
TRAIN CONFUSION MATRIX
[[ 5507  1919]
 [12678 28896]]
test confusion matrix
[[ 2939  1607]
 [ 9936 15518]]
```

In [227]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[5507,1919],[12678,28896]]

train=pd.DataFrame(array,columns=['predicted-0','predicted-1'],index=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[227]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x247b3158898>
```



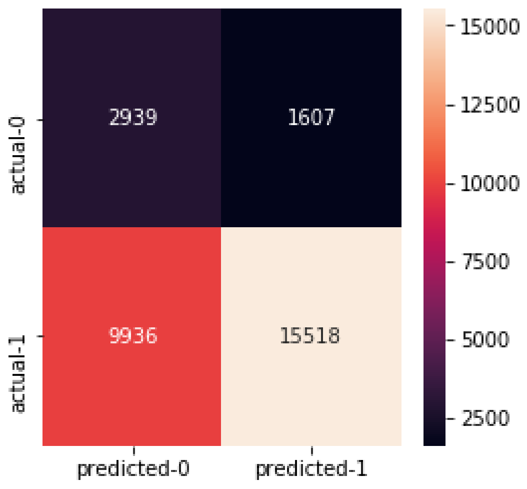
OBSERVATIONS: For training data we got good tnr and tpr values. And the value of tnr is a bit high.

In [228]:

```
#printing heatmap for test confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[2939,1607],[9936,15518]]

train=pd.DataFrame(array,columns=['predicted-0','predicted-1'],index=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[228]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x247b34de128>
```



OBSERVATIONS: For test data we got good tnr and tpr values. The fnr value is a bit high.

2.5 [Task-2]Getting top 5k features using `feature_importances_`

In [0]:

```
# please write all the code with proper documentation, and proper titles for each subsectic
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your coc
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [229]:

```
best_k=tfidf_Decision_tree.feature_importances_
best_k.shape
```

Out[229]:

```
(14816,)
```

In [230]:

```
def best5000important(model, X, k):
 return X[:,model.feature_importances_.argsort()[::-1][:k]]
```

In [231]:

```
# for tf-idf set 2
best_5k_train = best5000important(tfidf_Decision_tree,final_train_tfidf ,5000)
best_5k_cv = best5000important(tfidf_Decision_tree,final_cv_tfidf ,5000)
best_5k_test= best5000important(tfidf_Decision_tree, final_test_tfidf, 5000)
print(best_5k_train.shape)
print(best_5k_cv.shape)
print(best_5k_test.shape)
```

```
(49000, 5000)
(21000, 5000)
(30000, 5000)
```

In [233]:

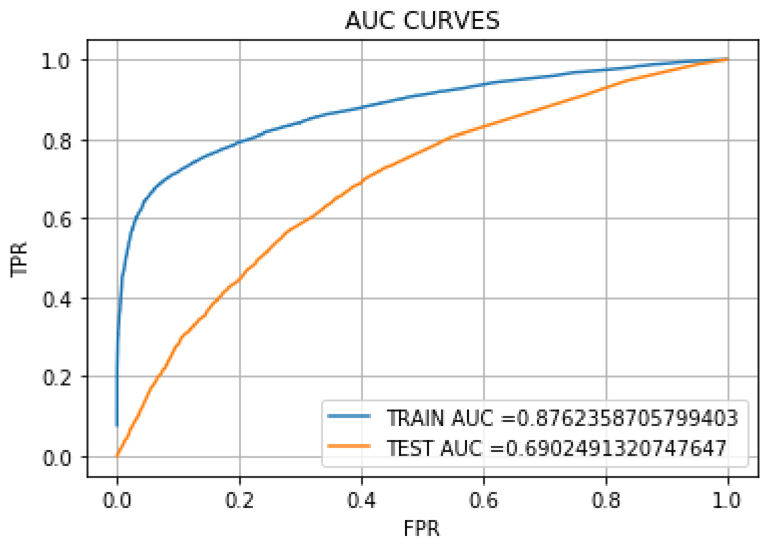
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
import plotly.graph_objs as go
from tqdm import tqdm

train_auc=[]
cv_auc=[]
depth=[4,6,8,9,10,12,14,17]
min_samples=[2,10,20,30,40,50,60,70]
for i in tqdm(depth):
    for j in tqdm(min_samples):
        Decision_tree=DecisionTreeClassifier(max_depth=i,class_weight='balanced',min_sample
        calib_cv=CalibratedClassifierCV(base_estimator=Decision_tree)
        calib_cv.fit(best_5k_train,y_train)

        y_tr_pred=calib_cv.predict_proba(best_5k_train)[:,1]
        y_cv_pred=calib_cv.predict_proba(best_5k_cv)[:,1]

        train_auc.append(roc_auc_score(y_train,y_tr_pred))
        cv_auc.append(roc_auc_score(y_cv,y_cv_pred))
        print('for {} max depth and {} min samples the train_auc={}'.format(i,j,roc_auc_sco
        print('for {} max depth and {} min samples the cv_auc={}'.format(i,j,roc_auc_score(


trace1=go.Scatter3d(x=min_samples,y=depth,z=train_auc,name='TRAIN AUC')
trace2=go.Scatter3d(x=min_samples,y=depth,z=cv_auc,name='CV AUC')
data=[trace1,trace2]
enable_plotly_in_cell()

layout=go.Layout(scene = dict(
        xaxis = dict(title='min_samples'),
        yaxis = dict(title='max_depth'),
        zaxis = dict(title='AUC'),))
fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

```
  0%|
| 0/8 [00:00<?, ?it/s]
  0%|
| 0/8 [00:00<?, ?it/s]

for 4 max depth and 2 min samples the train_auc=0.6750697353769618
for 4 max depth and 2 min samples the cv_auc=0.6484488316428582


 12%|██████████
| 1/8 [00:07<00:53,  7.63s/it]

for 4 max depth and 10 min samples the train_auc=0.6750697353769618
for 4 max depth and 10 min samples the cv_auc=0.6484488316428582


 25%|█████████████████████
| 2/8 [00:15<00:45,  7.57s/it]

for 4 max depth and 20 min samples the train_auc=0.6750697353769618
for 4 max depth and 20 min samples the cv_auc=0.6484488316428582
```

OBSERVATIONS: We selected the top 5000 features of the data and we plotted for different values of max depth and min samples split. We choose our max depth to be 8 and min samples split to be 10.

In [234]:

```
#plotting auc curves
from sklearn.metrics import roc_curve,auc
Decision_tree=DecisionTreeClassifier(max_depth=8,min_samples_split=10,class_weight='balance
calib_cv=CalibratedClassifierCV(base_estimator=Decision_tree)
calib_cv.fit(best_5k_train,y_train)

y_train_pred=calib_cv.predict_proba(best_5k_train)[:,1]
y_test_pred=calib_cv.predict_proba(best_5k_test)[:,1]

train_fpr,train_tpr,tr_threshold=roc_curve(y_train,y_tr_pred)
test_fpr,test_tpr,te_threshold=roc_curve(y_test,y_test_pred)

plt.plot(train_fpr,train_tpr,label='TRAIN AUC ='+str(auc(train_fpr,train_tpr)))
plt.plot(test_fpr,test_tpr,label='TEST AUC ='+str(auc(test_fpr,test_tpr)))

plt.title('AUC CURVES')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.legend()
plt.grid()
plt.show()
```



OBSERVATIONS: For max depth=8 and min samples split=10 we got train auc of 87.62 and test auc of 69.02.

In [235]:

```
#printing confusion matrix
print('='*100)
from sklearn.metrics import confusion_matrix
best_t=find_best_threshold(tr_threshold,train_fpr,train_tpr)
print('TRAIN CONFUSION MATRIX')
print(confusion_matrix(y_train,predict_with_best_t(y_train_pred,best_t)))
print('test confusion matrix')
print(confusion_matrix(y_test,predict_with_best_t(y_test_pred,best_t)))
```

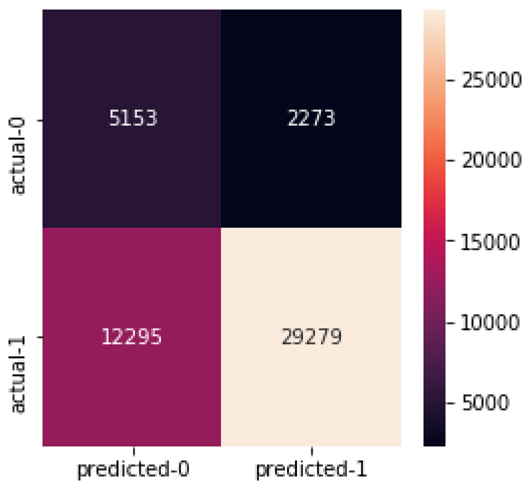
```
============================================================================
========================
the maximum value of tpr*(1-fpr) 0.6487601385351746 for threshold 0.83
TRAIN CONFUSION MATRIX
[[ 5153  2273]
 [12295 29279]]
test confusion matrix
[[ 2991  1555]
 [ 9388 16066]]
```

In [236]:

```
#printing heatmap for train confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[5153,2273],[12295,29279]]

train=pd.DataFrame(array,columns=['predicted-0','predicted-1'],index=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[236]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x247b32b16d8>
```



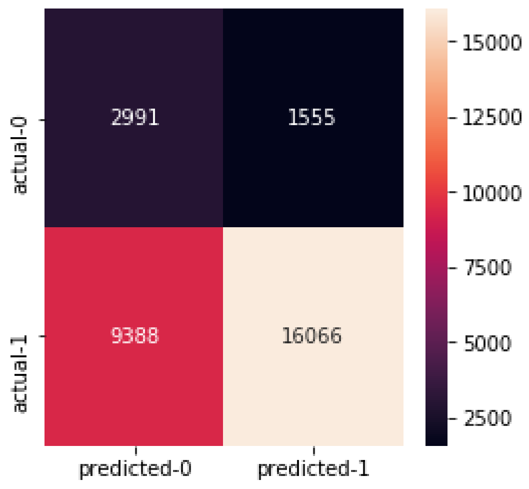
OBSERVATIONS: For test set we got a decent tnr and tpr values.

In [237]:

```
#printing heatmap for test confusion matrix
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

array=[[2991,1555],[9388,16066]]

train=pd.DataFrame(array,columns=['predicted-0','predicted-1'],index=['actual-0','actual-1'
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[237]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x247b3bf70f0>
```



OBSERVATIONS: For test set we got a decent tnr and tpr values.

3. Conclusion

In [0]:

```
# Please compare all your models using Prettytable library
```

In [238]:

```
# Please compare all your models using Prettytable library
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x=PrettyTable(['vectorizer','max depth','min samples split','train_auc','test_auc'])
x.add_row(["bag of words",9,10,0.866964,0.696859])
x.add_row(["TFIDF",6,10,0.883428,0.677248])
x.add_row(["avgw2v",6,10,0.921799,0.671101])
x.add_row(["TFIDFW2V",8,10,0.921291,0.677069])
x.add_row(["Best 5k",8,10,0.876235,0.690249])

print(x.get_string(start=0,end=7))
```

```
+--------------+-----------+-------------------+-----------+----------+
|  vectorizer  | max depth | min samples split | train_auc | test_auc |
+--------------+-----------+-------------------+-----------+----------+
| bag of words |     9     |         10        |  0.866964 | 0.696859 |
|    TFIDF     |     6     |         10        |  0.883428 | 0.677248 |
|    avgw2v    |     6     |         10        |  0.921799 | 0.671101 |
|   TFIDFW2V   |     8     |         10        |  0.921291 | 0.677069 |
|   Best 5k    |     8     |         10        |  0.876235 | 0.690249 |
+--------------+-----------+-------------------+-----------+----------+
```

OBSERVATIONS: We have plotted for different models and best max depth varied between 6 to 9 and min samples split is 10 for all models. The train auc and test auc is almost same for all the models.