DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example**
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targete enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories f following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal ((https://en.wikipedia.org/wiki/List_of_U.S._state_abbr **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategori **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the projec • `My students need hands on literacy mater sensory needs!`

Feature	Description
project_essay_1	First application essay*
project_essay_2	Second application essay*
project_essay_3	Third application essay*
project_essay_4	Fourth application essay*
project_submitted_datetime	Datetime when project application was submitted. **Ex** 12:43:56.245
teacher_id	A unique identifier for the teacher of the proposed pro bdf8baa8fedef6bfeec7ae4ff1c15c56
teacher_prefix	Teacher's title. One of the following enumerated value • nan • Dr. • Mr. • Mrs. • Ms. • Teacher.
teacher_number_of_previously_posted_projects	Number of project applications previously submitted b **Example:** 2

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [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'
'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [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/47
301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-stri
ng
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-pyth
on
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", "Warmt
h", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "M
ath & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace
 it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"M
ath & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spa
ces
        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/47
301924/4084039

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

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", "Warmt
h", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "M
ath & Science"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace
 it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"M
ath & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spa
ces
        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_s
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]:

```
# 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 school 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 experiences and keep on wanting more.With these resources such as the comfy red throw pillows and the whimsical nautical hanging decor and the blue fish nets, I will be able to help create the mood in our classroom setting to be one of a themed nautical environment. Creating a classroom environment is very important in the success in each and every child's education. The nautical photo props will be used with each child as they step foot into our classroom for the first time on Meet the Teacher evening. I'll take pictures of each child with them, have them developed, and then hung in our classroom ready for their first day of 4th grade. This kind gesture will set the tone before even the first day of school! The nautical thank you cards will be used throughout the year by the students as they create thank you cards to their team groups.\r\n\r\nYour generous donations will help me to help make our classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of my own pocket on resources to get our classroom ready. Please consider helping with this project 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. They are eager beavers and always strive to work their hardest working past their limitations. \r\n\r\nThe materials we have are the ones I seek out for my students. I teach in a Title I school where most of the students receive free or reduced 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 were in a meeting? This is how my kids feel all the time. The want to be able to move as they learn or so they say.Wobble chairs are the answer and I love then because they develop their core, which enhances gross motor and in Turn fine motor skills. \r\nThey also want to learn 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 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

==

The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates. The great teacher inspires. -William A. Ward\r\n\r\nMy school 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 made 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 children from rich backgrounds or neighborhoods. As an educator I am inspiring minds of young children and we focus not only on academics but one 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 message. Due to the volume of my speaker my students can't hear videos or books 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 replay it at any time.\r\nThe cart will allow me to have more room for storage of things that are needed for the day and has an extra part to it I can use. The table top chart has all of the letter, words and pictures for students to learn about different letters and it is more accessible.nannan

==

In [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
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 do not want to sit and do worksheets. They want t
o 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 happe
n. My students will forget they are doing work and just have the fun a 6 y
ear 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
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.     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 f
ree or reduced price lunch.  Despite their disabilities and limitations, m
y 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 were in a meeting? This is how my kids feel all the time. The
want to be able to move as they learn or so they say.Wobble chairs are the
answer and I love then because they develop their core, which enhances gro
ss motor and in Turn fine motor skills.   They also want to learn through
games, my kids do not want to sit and do worksheets. They want to learn to
count by jumping and playing. Physical engagement is the key to our succes
s. The number toss and color and shape mats can make that happen. My stude
nts will forget they are doing work and just have the fun a 6 year old des
erves.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
language delays cognitive delays gross fine motor delays to autism They ar
e eager beavers and always strive to work their hardest working past their
limitations The materials we have are the ones I seek out for my students
I teach in a Title I school where most of the students receive free or red
uced price lunch Despite their disabilities and limitations my students lo
ve 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 w
ere in a meeting This is how my kids feel all the time The want to be able
to move as they learn or so they say Wobble chairs are the answer and I lo
ve 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 no
t 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 [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've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [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
cognitive delays gross fine motor delays autism eager beavers always striv
e work hardest working past limitations materials ones seek students teach
title school students receive free reduced price lunch despite disabilitie
s limitations students love coming school come eager learn explore ever fe
lt like ants pants needed groove move meeting kids feel time want able mov
e learn say wobble chairs answer love develop core enhances gross motor tu
rn fine motor skills also want learn games kids not want sit worksheets wa
nt learn count jumping 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', 'essa
y'],
      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/)

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', 'lit
eracy_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'].v
alues)
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', 'economi
cs', 'environmentalscience', 'esl', 'extracurricular', 'financialliterac
y', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_welln
ess', 'history_geography', 'literacy', 'literature_writing', 'mathematic
s', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performi
ngarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'wa
rmth']
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 al
so
```

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',
'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'm
o', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'o
k', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'w
i', '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)
```

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 pro
jects).
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-p
ickle-to-save-and-load-variables-in-python/

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


'''
```

Out[0]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/3823034
9/4084039\ndef loadGloveModel(gloveFile):\n    print ("Loading Glove Mode
l")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    model = {}\n    f
or line in tqdm(f):\n        splitLine = line.split()\n        word = spli
tLine[0]\n        embedding = np.array([float(val) for val in splitLine
[1:]])\n        model[word] = embedding\n    print ("Done.",len(model)," w
ords loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.
txt\')\n\n# ============================\nOutput:\n    \nLoading Glove Mod
el\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n# ====
========================\n\nwords = []\nfor i in preproced_texts:\n    wor
ds.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n    words.extend
(i.split(\' \'))\nprint("all the words in the coupus", len(words))\nwords
= set(words)\nprint("the unique words in the coupus", len(words))\n\ninter
_words = set(model.keys()).intersection(words)\nprint("The number of words
that are present in both glove vectors and our coupus",       len(inter_wo
rds),"(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpu
s = {}\nwords_glove = set(model.keys())\nfor i in words:\n    if i in word
s_glove:\n        words_courpus[i] = model[i]\nprint("word 2 vec length",
len(words_courpus))\n\n\n# stronging variables into pickle files python: h
ttp://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-
python/\n\nimport pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n
pickle.dump(words_courpus, f)\n\n\n'
```

In [31]:

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

In [32]:

```
# 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 li
st
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((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

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((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    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_i
ndex()
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
_essay_3','project_essay_4','project_is_approved'],axis=1)
data1.head(2)
data=data1[0:100000]
data[0:2]
```

Out[118]:

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

In [119]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
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.   287.73   5.5 ].
# 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 sta
ndard deviation of this data
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
.reshape(-1,1)) # finding the mean and standard deviation of this data
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_previ
ously_posted_projects'].values.reshape(-1,1))
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 a
nd standard deviation of this data
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.resha
pe(-1,1))
quantity_standardized
```

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

Out[122]:

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

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 subse
ction
# 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
students with the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multi
ple intelligences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety
 of different backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school
is a caring community of successful \
learners which can be seen through collaborative student project based learning in and
 out of the classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities
 to practice a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspec
t of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love
 to role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with
real food i will take their idea \
and create common core cooking lessons where we learn important math and writing concep
ts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that wen
t into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this p
roject would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make hom
emade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create o
ur own cookbooks to be printed and \
shared with families students will gain math and literature skills as well as a life lo
ng enjoyment for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

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

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

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

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])**

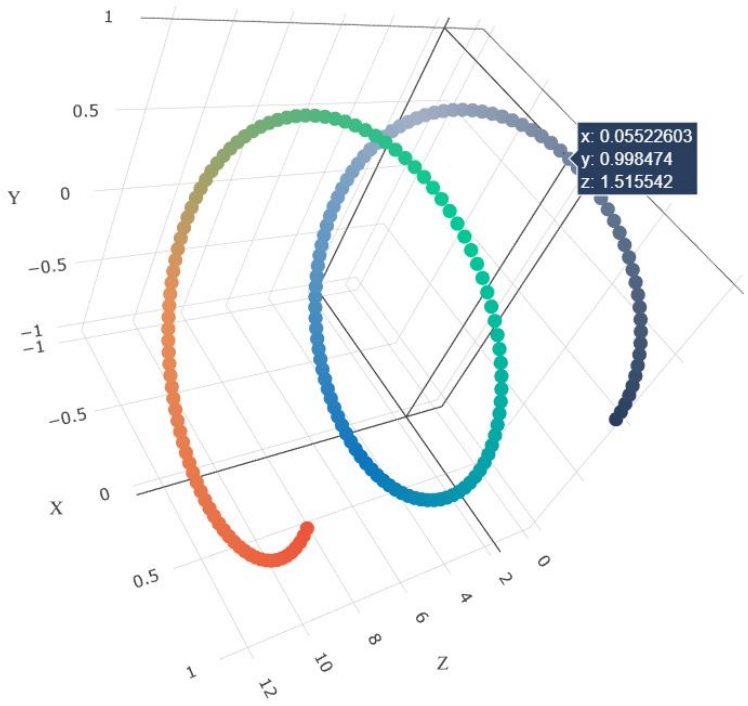
    - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data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

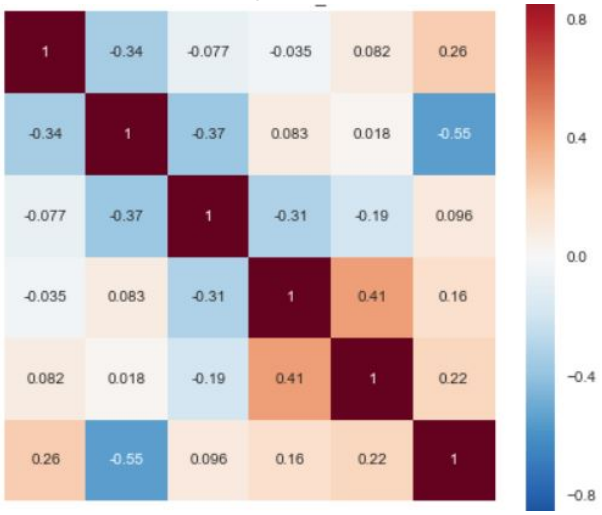
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



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) 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/) 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/)

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 subse
ction
# 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
code
# 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_tr
ain)
```

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 subse
ction
# 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
code
# 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'].value
s)
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', 'lit
eracy_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_subcategor
ies'].values)
X_cv_subcategories_encoded=subcategoryvectorizer.transform(X_cv['clean_subcategories'].
values)
X_test_subcategories_encoded=subcategoryvectorizer.transform(X_test['clean_subcategorie
s'].values)
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', 'economi
cs', 'environmentalscience', 'esl', 'extracurricular', 'financialliterac
y', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_welln
ess', 'history_geography', 'literacy', 'literature_writing', 'mathematic
s', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performi
ngarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'wa
rmth']
(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
_projects'].values.reshape(1,-1))
X_cv_projects_norm=normalizer.transform(X_cv['teacher_number_of_previously_posted_proje
cts'].values.reshape(1,-1))
X_test_projects_norm=normalizer.transform(X_test['teacher_number_of_previously_posted_p
rojects'].values.reshape(1,-1))

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)
```

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 subse
ction
# 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
code
# 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,)
```

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 subse
ction
# 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
code
# 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_enco
ded,X_train_category_encoded,X_train_subcategories_encoded,price_train_norm,projects_tr
ain_norm,train_quantity_norm,X_train_essay_bow,X_train_title_bow)).tocsr()
final_cv_bow=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_cat
egory_encoded,X_cv_subcategories_encoded,price_cv_norm,projects_cv_norm,cv_quantity_nor
m,X_cv_essay_bow,X_cv_title_bow)).tocsr()
final_test_bow=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encoded,
X_test_category_encoded,X_test_subcategories_encoded,price_test_norm,projects_test_norm
,test_quantity_norm,X_test_essay_bow,X_test_title_bow)).tocsr()
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/requir
e.js"></script>'''))
  init_notebook_mode(connected=False)
```

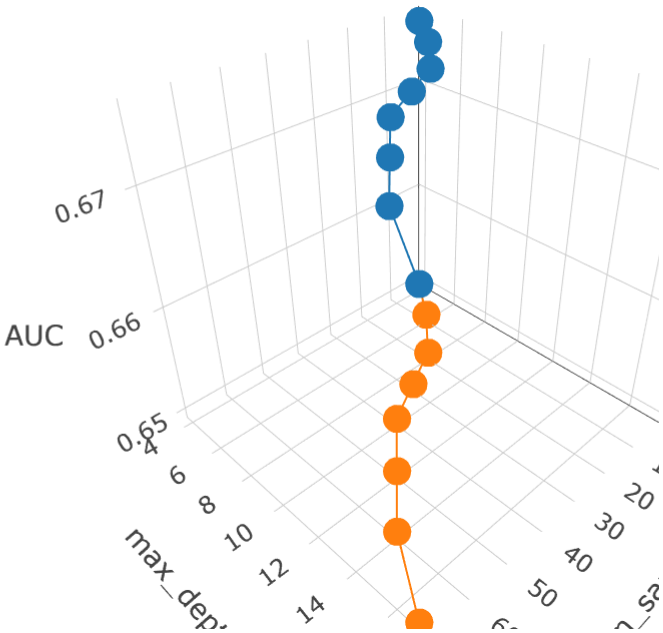
In [149]:

```
#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_sa
mples_split=j)
        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%|
| 1/8 [00:49<05:46, 49.54s/it]
  0%|
| 0/8 [00:00<?, ?it/s]
 12%|
| 1/8 [00:11<01:19, 11.35s/it]
 25%|
| 2/8 [00:22<01:07, 11.24s/it]
 38%|
| 3/8 [00:33<00:55, 11.15s/it]
 50%|
| 4/8 [00:44<00:45, 11.26s/it]
 62%|
| 5/8 [00:55<00:33, 11.16s/it]
 75%|
| 6/8 [01:06<00:22, 11.03s/it]
 88%|
| 7/8 [01:17<00:11, 11.08s/it]
 25%|
| 2/8 [02:17<06:07, 61.20s/it]
  0%|
| 0/8 [00:00<?, ?it/s]
 12%|
| 1/8 [00:20<02:22, 20.40s/it]
 25%|
| 2/8 [00:39<02:00, 20.12s/it]
 38%|
| 3/8 [00:58<01:39, 19.83s/it]
 50%|
| 4/8 [01:18<01:18, 19.63s/it]
 62%|
| 5/8 [01:37<00:58, 19.47s/it]
 75%|
| 6/8 [01:55<00:38, 19.12s/it]
 88%|
| 7/8 [02:14<00:18, 18.99s/it]
 38%|
| 3/8 [04:50<07:23, 88.62s/it]
  0%|
| 0/8 [00:00<?, ?it/s]
 12%|
| 1/8 [00:25<02:59, 25.70s/it]
 25%|
```

```
| 2/8 [00:50<02:33, 25.51s/it]
 38%|
| 3/8 [01:15<02:06, 25.20s/it]
 50%|
| 4/8 [01:39<01:39, 24.99s/it]
 62%|
| 5/8 [02:04<01:14, 24.95s/it]
 75%|
| 6/8 [02:28<00:49, 24.54s/it]
 88%|
          | 7/8 [02:51<00:24, 24.07s/it]
 50%|
| 4/8 [08:04<08:01, 120.28s/it]
  0%|
| 0/8 [00:00<?, ?it/s]
 12%|
| 1/8 [00:32<03:49, 32.83s/it]
 25%|
| 2/8 [01:04<03:14, 32.42s/it]
 38%|
| 3/8 [01:34<02:39, 31.80s/it]
 50%|
| 4/8 [02:04<02:04, 31.16s/it]
 62%|
| 5/8 [02:34<01:32, 30.77s/it]
 75%|
| 6/8 [03:02<01:00, 30.05s/it]
 88%|
          | 7/8 [03:30<00:29, 29.53s/it]
 62%|
| 5/8 [12:03<07:47, 155.77s/it]
  0%|
| 0/8 [00:00<?, ?it/s]
 12%|
| 1/8 [00:46<05:27, 46.84s/it]
 25%|
| 2/8 [01:32<04:38, 46.39s/it]
 38%|
| 3/8 [02:15<03:47, 45.43s/it]
 50%|
| 4/8 [02:58<02:58, 44.59s/it]
 62%|
| 5/8 [03:38<02:10, 43.36s/it]
 75%|
| 6/8 [04:18<01:24, 42.22s/it]
 88%|
          | 7/8 [04:58<00:41, 41.58s/it]
 75%|
| 6/8 [17:39<07:00, 210.02s/it]
  0%|
| 0/8 [00:00<?, ?it/s]
 12%|
| 1/8 [01:00<07:06, 60.94s/it]
 25%|
| 2/8 [01:58<05:59, 59.93s/it]
 38%|
| 3/8 [02:54<04:53, 58.74s/it]
 50%|
| 4/8 [03:49<03:50, 57.57s/it]
 62%|
| 5/8 [04:41<02:47, 56.00s/it]
```

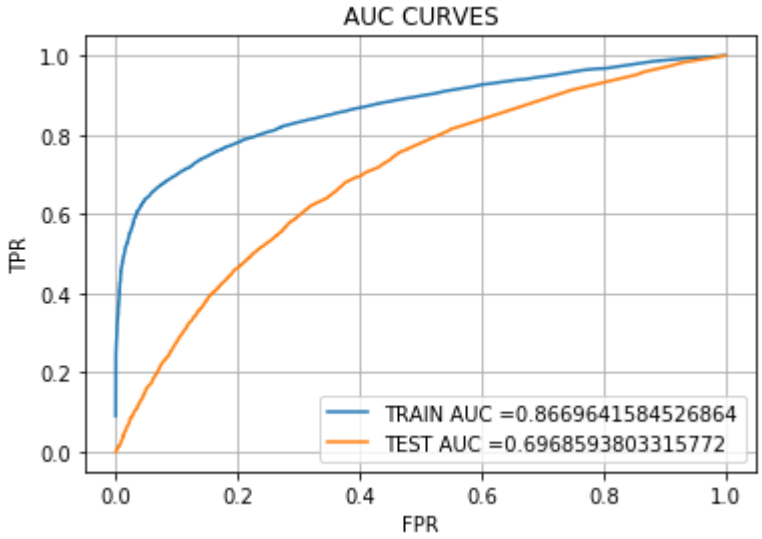
```
 75%|
| 6/8 [05:33<01:49, 54.62s/it]
 88%|
| 7/8 [06:22<00:53, 53.15s/it]
 88%|
| 7/8 [24:51<04:36, 276.57s/it]
  0%|
| 0/8 [00:00<?, ?it/s]
 12%|
| 1/8 [01:20<09:20, 80.08s/it]
 25%|
| 2/8 [02:36<07:53, 78.91s/it]
 38%|
| 3/8 [03:48<06:24, 76.92s/it]
 50%|
| 4/8 [04:57<04:58, 74.55s/it]
 62%|
| 5/8 [06:05<03:37, 72.50s/it]
 75%|
| 6/8 [07:11<02:21, 70.67s/it]
 88%|
| 7/8 [08:14<01:08, 68.43s/it]
100%|
| 8/8 [34:08<00:00, 360.75s/it]
```



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='bal
anced')
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
\anaconda3\lib\site-packages (19.2.3)
```

In [153]:

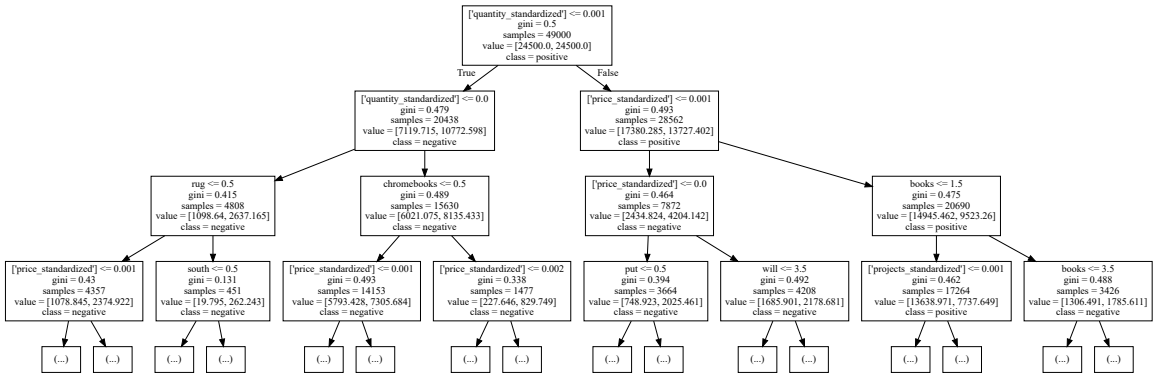
```
!pip install python-graphviz
```

```
Collecting python-graphviz

  ERROR: Could not find a version that satisfies the requirement python-gr
aphviz (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='bal
anced')
Decision_tree.fit(final_train_bow,y_train)

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

Out[157]:



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.rou
nd(t,3))
    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)))
```

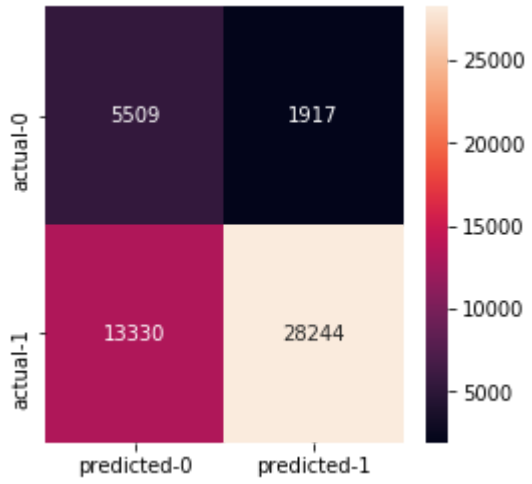
```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.6356544107340077 for threshold 0.83
TRAIN CONFUSION MATRIX
[[ 5754  1672]
 [14048 27526]]
test confusion matrix
[[ 3201  1345]
 [10413 15041]]
```

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','actua
l-1'])
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[160]:

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



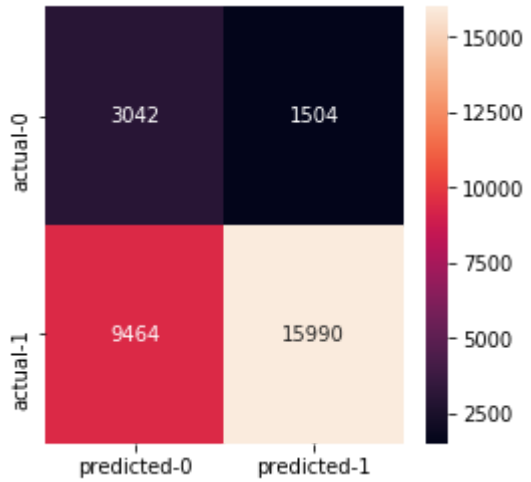
OBSERVATION: We got decent values for tnr and tpr. The tnr value is a bit high.

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','actua
l-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
```

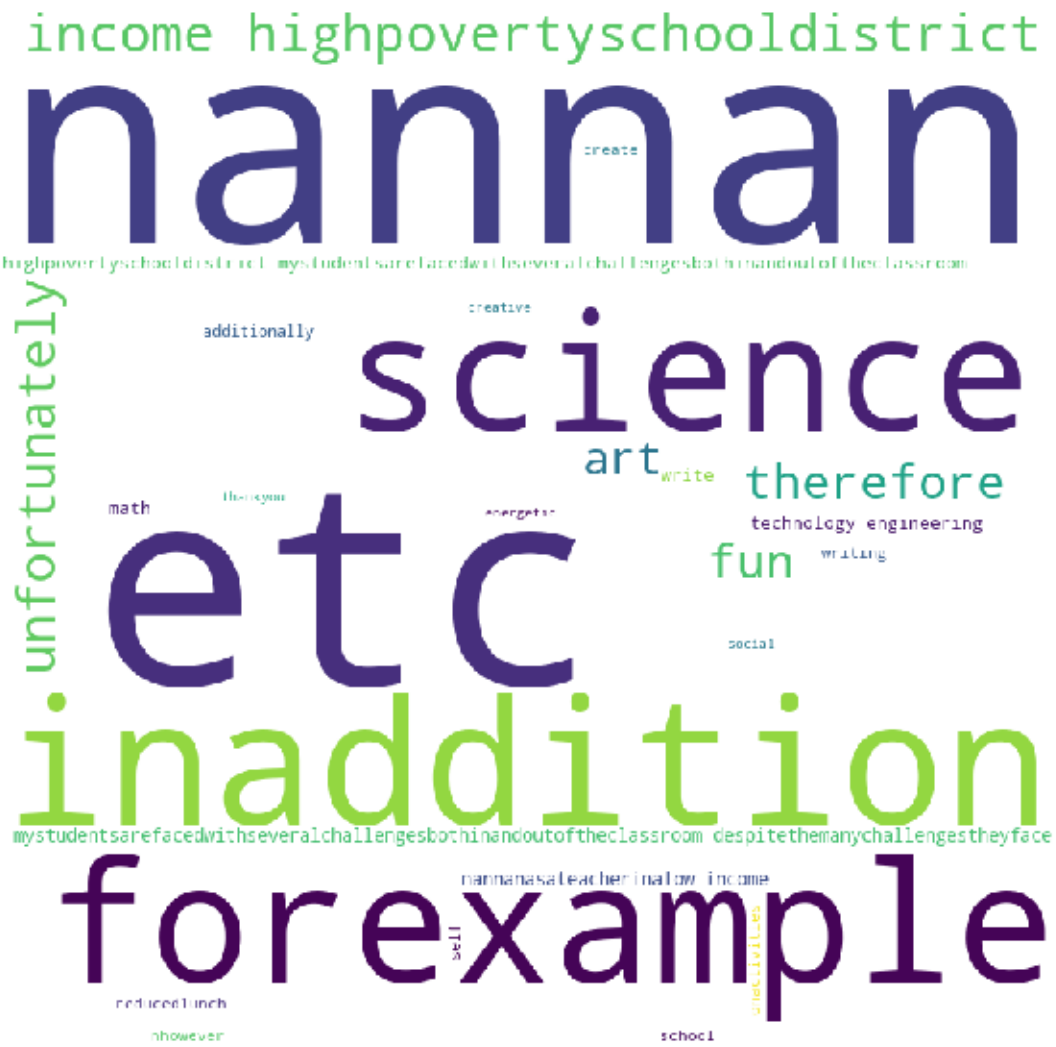
Out[173]:

(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
```

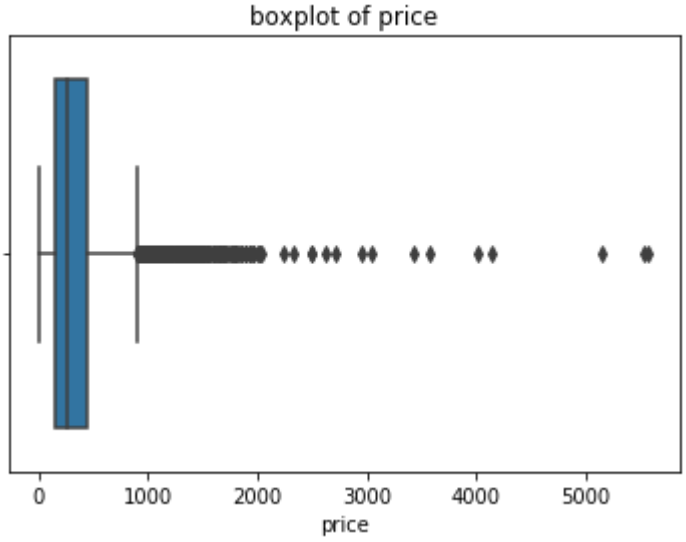
Out[84]:

```
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', 'quantit
y',
       'y_actual', 'y_hat'],
      dtype='object')
```

BOX PLOT OF PRICE

In [175]:

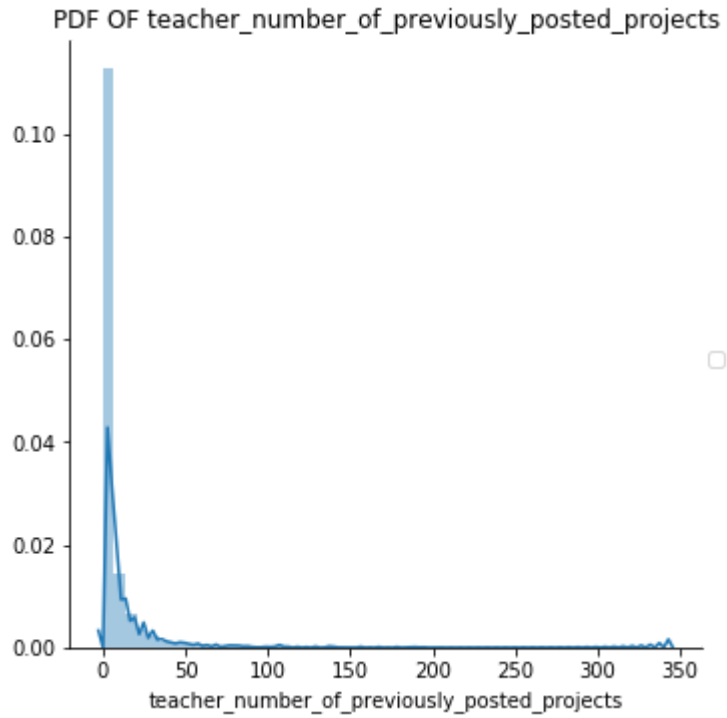
```
import seaborn as sns
sns.boxplot(x='price',data=fpr)
plt.title('boxplot of price')
plt.show()
```



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();
```



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_en
coded,X_train_category_encoded,X_train_subcategories_encoded,price_train_norm,projects_
train_norm,train_quantity_norm,train_tfidf,title_train_tfidf)).tocsr()
final_cv_tfidf=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_c
ategory_encoded,X_cv_subcategories_encoded,price_cv_norm,projects_cv_norm,cv_quantity_n
orm,cv_tfidf,title_cv_tfidf)).tocsr()
final_test_tfidf=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encode
d,X_test_category_encoded,X_test_subcategories_encoded,price_test_norm,projects_test_no
rm,test_quantity_norm,test_tfidf,title_test_tfidf)).tocsr()
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_samples_split=j)
        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
_score(y_train,y_tr_pred)))
        print('for {} max depth and {} min samples the cv_auc={}'.format(i,j,roc_auc_sc
ore(y_cv,y_cv_pred)))


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]

for 4 max depth and 20 min samples the train_auc=0.6751862665595485
for 4 max depth and 20 min samples the cv_auc=0.6485219309085037

 38%|
| 3/8 [00:44<01:11, 14.30s/it]

for 4 max depth and 30 min samples the train_auc=0.6751862665595485
for 4 max depth and 30 min samples the cv_auc=0.6485219309085037

 50%|
| 4/8 [00:59<00:58, 14.63s/it]

for 4 max depth and 40 min samples the train_auc=0.6751862665595485
for 4 max depth and 40 min samples the cv_auc=0.6485219309085037

 62%|
| 5/8 [01:15<00:44, 14.96s/it]

for 4 max depth and 50 min samples the train_auc=0.6751862665595485
for 4 max depth and 50 min samples the cv_auc=0.6485219309085037

 75%|
| 6/8 [01:31<00:30, 15.26s/it]

for 4 max depth and 60 min samples the train_auc=0.6751862665595485
for 4 max depth and 60 min samples the cv_auc=0.6485219309085037

 88%|
          | 7/8 [01:47<00:15, 15.72s/it]

for 4 max depth and 70 min samples the train_auc=0.6751862665595485
for 4 max depth and 70 min samples the cv_auc=0.6485219309085037

 12%|
| 1/8 [02:03<14:24, 123.50s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 6 max depth and 2 min samples the train_auc=0.7180212152991734
for 6 max depth and 2 min samples the cv_auc=0.6584226527754368

 12%|
| 1/8 [00:26<03:04, 26.35s/it]

for 6 max depth and 10 min samples the train_auc=0.7175716552837856
for 6 max depth and 10 min samples the cv_auc=0.6585189455588346

 25%|
| 2/8 [00:53<02:39, 26.59s/it]
```

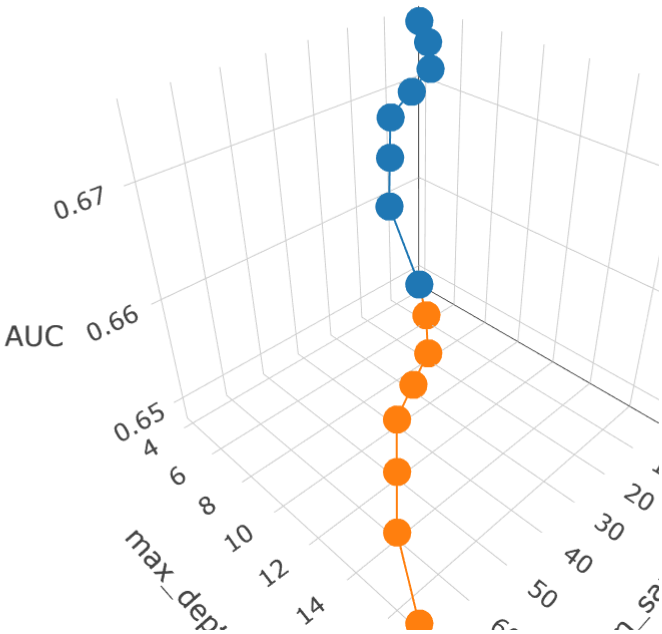
```
for 6 max depth and 20 min samples the train_auc=0.7171234281546335
for 6 max depth and 20 min samples the cv_auc=0.6580060072445614

 38%|████████████████████████████
| 3/8 [01:20<02:13, 26.78s/it]

for 6 max depth and 30 min samples the train_auc=0.7164054122190537
for 6 max depth and 30 min samples the cv_auc=0.6580719897159766

 50%|██████████████████████████████████████
| 4/8 [01:47<01:46, 26.68s/it]

for 6 max depth and 40 min samples the train_auc=0.715653437646079
for 6 max depth and 40 min samples the cv_auc=0.658698249970598

 62%|████████████████████████████████████████████████
| 5/8 [02:11<01:18, 26.11s/it]

for 6 max depth and 50 min samples the train_auc=0.71526727961165
for 6 max depth and 50 min samples the cv_auc=0.6581381838392648

 75%|██████████████████████████████████████████████████████████
| 6/8 [02:36<00:51, 25.72s/it]

for 6 max depth and 60 min samples the train_auc=0.7151145094711105
for 6 max depth and 60 min samples the cv_auc=0.6585294311453774

 88%|████████████████████████████████████████████████████████████████
████           | 7/8 [03:01<00:25, 25.45s/it]

for 6 max depth and 70 min samples the train_auc=0.7147694441735485
for 6 max depth and 70 min samples the cv_auc=0.6583168180200969

 25%|███████████████████▌
| 2/8 [05:27<14:45, 147.59s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 8 max depth and 2 min samples the train_auc=0.7701416601207863
for 8 max depth and 2 min samples the cv_auc=0.6737317061349201

 12%|██████████
| 1/8 [00:34<04:01, 34.54s/it]

for 8 max depth and 10 min samples the train_auc=0.7686774400540974
for 8 max depth and 10 min samples the cv_auc=0.6732332483362928

 25%|████████████████████
| 2/8 [01:13<03:35, 35.87s/it]

for 8 max depth and 20 min samples the train_auc=0.7654400083874336
for 8 max depth and 20 min samples the cv_auc=0.6728726729141125

 38%|██████████████████████████████
| 3/8 [01:52<03:04, 36.80s/it]

for 8 max depth and 30 min samples the train_auc=0.763529592102089
for 8 max depth and 30 min samples the cv_auc=0.6728088245990839

 50%|████████████████████████████████████████
| 4/8 [02:28<02:26, 36.70s/it]

for 8 max depth and 40 min samples the train_auc=0.7620803593127017
for 8 max depth and 40 min samples the cv_auc=0.6728898255346555

 62%|██████████████████████████████████████████████████
| 5/8 [03:04<01:49, 36.40s/it]
```

```
for 8 max depth and 50 min samples the train_auc=0.7605819587308362
for 8 max depth and 50 min samples the cv_auc=0.6732597930087013

 75%|███████████████████████████████████████████████████████████
| 6/8 [03:41<01:13, 36.61s/it]

for 8 max depth and 60 min samples the train_auc=0.7592268863372016
for 8 max depth and 60 min samples the cv_auc=0.6730880110572582

 88%|████████████████████████████████████████████████████████████
████           | 7/8 [04:20<00:37, 37.17s/it]

for 8 max depth and 70 min samples the train_auc=0.758503903254498
for 8 max depth and 70 min samples the cv_auc=0.6733501154455142

 38%|████████████████████████████
| 3/8 [10:24<16:02, 192.44s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 9 max depth and 2 min samples the train_auc=0.7957857693771113
for 9 max depth and 2 min samples the cv_auc=0.6768747011740117

 12%|█████████
| 1/8 [00:46<05:25, 46.57s/it]

for 9 max depth and 10 min samples the train_auc=0.7936532599106392
for 9 max depth and 10 min samples the cv_auc=0.6757615798796391

 25%|██████████████████
| 2/8 [01:34<04:41, 46.84s/it]

for 9 max depth and 20 min samples the train_auc=0.7896529184974176
for 9 max depth and 20 min samples the cv_auc=0.6758424308951343

 38%|████████████████████████████
| 3/8 [02:17<03:49, 45.89s/it]

for 9 max depth and 30 min samples the train_auc=0.7861702730130631
for 9 max depth and 30 min samples the cv_auc=0.675792137118807

 50%|████████████████████████████████████
| 4/8 [03:01<03:00, 45.23s/it]

for 9 max depth and 40 min samples the train_auc=0.7835395345588476
for 9 max depth and 40 min samples the cv_auc=0.6764145435455738

 62%|█████████████████████████████████████████████
| 5/8 [03:39<02:09, 43.12s/it]


for 9 max depth and 50 min samples the train_auc=0.7815272067313093
for 9 max depth and 50 min samples the cv_auc=0.6757184117163704

 75%|███████████████████████████████████████████████████████████
| 6/8 [04:31<01:31, 45.76s/it]

for 9 max depth and 60 min samples the train_auc=0.7801919397638813
for 9 max depth and 60 min samples the cv_auc=0.6761003904342102

 88%|████████████████████████████████████████████████████████████
████           | 7/8 [05:19<00:46, 46.34s/it]

for 9 max depth and 70 min samples the train_auc=0.7791420659271511
for 9 max depth and 70 min samples the cv_auc=0.6767340761420434
```

```
 50%|
| 4/8 [16:29<16:17, 244.33s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 10 max depth and 2 min samples the train_auc=0.8187504647286818
for 10 max depth and 2 min samples the cv_auc=0.6810601081442299

 12%|
| 1/8 [00:56<06:37, 56.76s/it]

for 10 max depth and 10 min samples the train_auc=0.8158812125179596
for 10 max depth and 10 min samples the cv_auc=0.680660482951477

 25%|
| 2/8 [01:54<05:42, 57.09s/it]

for 10 max depth and 20 min samples the train_auc=0.8100559004389241
for 10 max depth and 20 min samples the cv_auc=0.6797475137783606

 38%|
| 3/8 [02:49<04:41, 56.34s/it]

for 10 max depth and 30 min samples the train_auc=0.8051466860898153
for 10 max depth and 30 min samples the cv_auc=0.6809527124563266

 50%|
| 4/8 [03:44<03:43, 55.98s/it]

for 10 max depth and 40 min samples the train_auc=0.8020724333848724
for 10 max depth and 40 min samples the cv_auc=0.6805498860289938

 62%|
| 5/8 [04:37<02:45, 55.24s/it]

for 10 max depth and 50 min samples the train_auc=0.8000159227269845
for 10 max depth and 50 min samples the cv_auc=0.6806336825330553

 75%|
| 6/8 [05:31<01:49, 54.66s/it]

for 10 max depth and 60 min samples the train_auc=0.7981785366226801
for 10 max depth and 60 min samples the cv_auc=0.6813671356425353

 88%|
          | 7/8 [06:12<00:50, 50.75s/it]

for 10 max depth and 70 min samples the train_auc=0.7962002322143709
for 10 max depth and 70 min samples the cv_auc=0.6813223977278747

 62%|
| 5/8 [23:45<15:04, 301.61s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 12 max depth and 2 min samples the train_auc=0.860948883362653
for 12 max depth and 2 min samples the cv_auc=0.6844295883251134

 12%|
| 1/8 [01:38<11:26, 98.04s/it]

for 12 max depth and 10 min samples the train_auc=0.8564041348508504
for 12 max depth and 10 min samples the cv_auc=0.6829457764127956

 25%|
| 2/8 [03:18<09:51, 98.64s/it]
```

```
for 12 max depth and 20 min samples the train_auc=0.847327843604111
for 12 max depth and 20 min samples the cv_auc=0.6841707380844053

 38%|████████████████████████████▍
| 3/8 [04:49<08:01, 96.35s/it]

for 12 max depth and 30 min samples the train_auc=0.8412350165610224
for 12 max depth and 30 min samples the cv_auc=0.6841110698938686

 50%|████████████████████████████████████▌
| 4/8 [06:21<06:20, 95.22s/it]

for 12 max depth and 40 min samples the train_auc=0.8373413870238955
for 12 max depth and 40 min samples the cv_auc=0.6841210351695567

 62%|████████████████████████████████████████████▋
| 5/8 [07:51<04:40, 93.59s/it]

for 12 max depth and 50 min samples the train_auc=0.8340972682524145
for 12 max depth and 50 min samples the cv_auc=0.6839580544085004

 75%|████████████████████████████████████████████████████▊
| 6/8 [09:14<03:01, 90.52s/it]

for 12 max depth and 60 min samples the train_auc=0.8313017717792736
for 12 max depth and 60 min samples the cv_auc=0.6835441868084584

 88%|████████████████████████████████████████████████████████████▉
████           | 7/8 [10:41<01:29, 89.24s/it]

for 12 max depth and 70 min samples the train_auc=0.8289853887942017
for 12 max depth and 70 min samples the cv_auc=0.6848790751716197

 75%|████████████████████████████████████████████████████▎
| 6/8 [35:56<14:21, 430.60s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 14 max depth and 2 min samples the train_auc=0.89101951752278
for 14 max depth and 2 min samples the cv_auc=0.6777569190937432

 12%|████████▍
| 1/8 [01:33<10:52, 93.19s/it]

for 14 max depth and 10 min samples the train_auc=0.8853436441784692
for 14 max depth and 10 min samples the cv_auc=0.6781461292505782

 25%|████████████████▉
| 2/8 [03:25<09:53, 98.94s/it]

for 14 max depth and 20 min samples the train_auc=0.8755615661868678
for 14 max depth and 20 min samples the cv_auc=0.6775277265717427

 38%|█████████████████████████▎
| 3/8 [05:21<08:40, 104.07s/it]

for 14 max depth and 30 min samples the train_auc=0.8681058070941317
for 14 max depth and 30 min samples the cv_auc=0.678555428697694

 50%|█████████████████████████████████▌
| 4/8 [07:06<06:57, 104.44s/it]

for 14 max depth and 40 min samples the train_auc=0.8636152178151183
for 14 max depth and 40 min samples the cv_auc=0.676801557814226

 62%|█████████████████████████████████████████▋
| 5/8 [08:59<05:21, 107.02s/it]
```

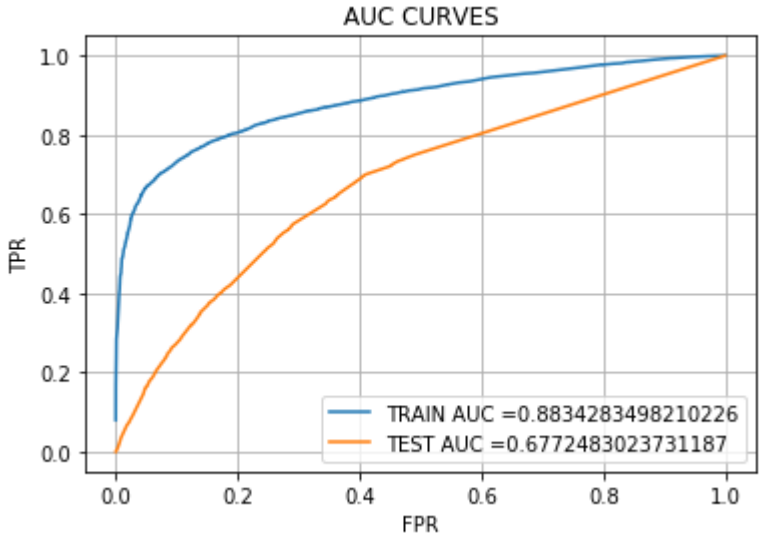
```
for 14 max depth and 50 min samples the train_auc=0.8600143276686673
for 14 max depth and 50 min samples the cv_auc=0.6776279260959635

 75%|
| 6/8 [10:48<03:34, 107.49s/it]

for 14 max depth and 60 min samples the train_auc=0.8567941474044037
for 14 max depth and 60 min samples the cv_auc=0.6771500073478476

 88%|
|  | 7/8 [12:38<01:48, 108.25s/it]

for 14 max depth and 70 min samples the train_auc=0.853512124781836
for 14 max depth and 70 min samples the cv_auc=0.6774716652818755

 88%|
|  | 7/8 [49:44<09:09, 549.87s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 17 max depth and 2 min samples the train_auc=0.9248801140901384
for 17 max depth and 2 min samples the cv_auc=0.6738592087507607

 12%|
| 1/8 [01:41<11:50, 101.43s/it]

for 17 max depth and 10 min samples the train_auc=0.9195063848392577
for 17 max depth and 10 min samples the cv_auc=0.6720643655922064

 25%|
| 2/8 [03:18<10:00, 100.02s/it]

for 17 max depth and 20 min samples the train_auc=0.9098696303163747
for 17 max depth and 20 min samples the cv_auc=0.6708966645710779

 38%|
| 3/8 [05:02<08:26, 101.34s/it]

for 17 max depth and 30 min samples the train_auc=0.902164127536204
for 17 max depth and 30 min samples the cv_auc=0.6725110656890514

 50%|
| 4/8 [06:42<06:43, 100.78s/it]

for 17 max depth and 40 min samples the train_auc=0.8959500515734659
for 17 max depth and 40 min samples the cv_auc=0.6739444339049651

 62%|
| 5/8 [08:18<04:58, 99.57s/it]

for 17 max depth and 50 min samples the train_auc=0.8912435395830156
for 17 max depth and 50 min samples the cv_auc=0.6717267543982495

 75%|
| 6/8 [09:53<03:16, 98.12s/it]

for 17 max depth and 60 min samples the train_auc=0.8865252453965025
for 17 max depth and 60 min samples the cv_auc=0.6736672934148965

 88%|
|  | 7/8 [11:26<01:36, 96.47s/it]

for 17 max depth and 70 min samples the train_auc=0.8834283498210226
for 17 max depth and 70 min samples the cv_auc=0.6749805403740411

100%|
| 8/8 [1:02:41<00:00, 617.97s/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_weigh
t='balanced')
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)))
```

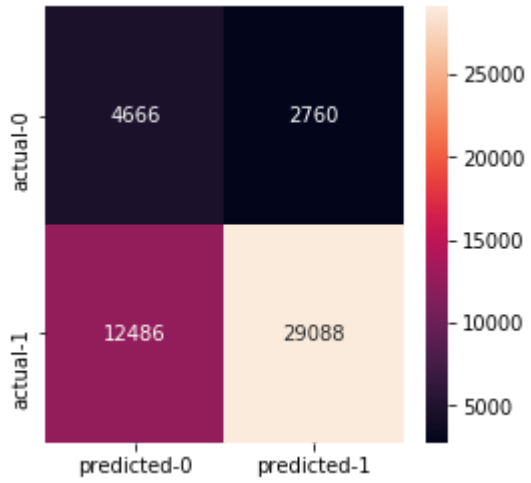
```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.6631577521486158 for threshold 0.829
TRAIN CONFUSION MATRIX
[[ 4666  2760]
 [12486 29088]]
test confusion matrix
[[ 2898  1648]
 [ 9014 16440]]
```

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','actua
l-1'])
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[184]:

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

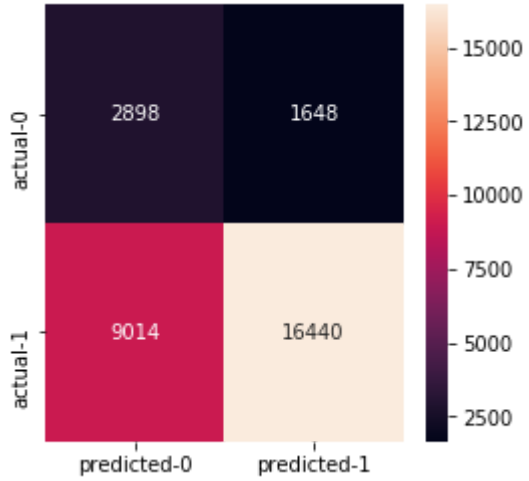


In [185]:

```
#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','actua
l-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]:

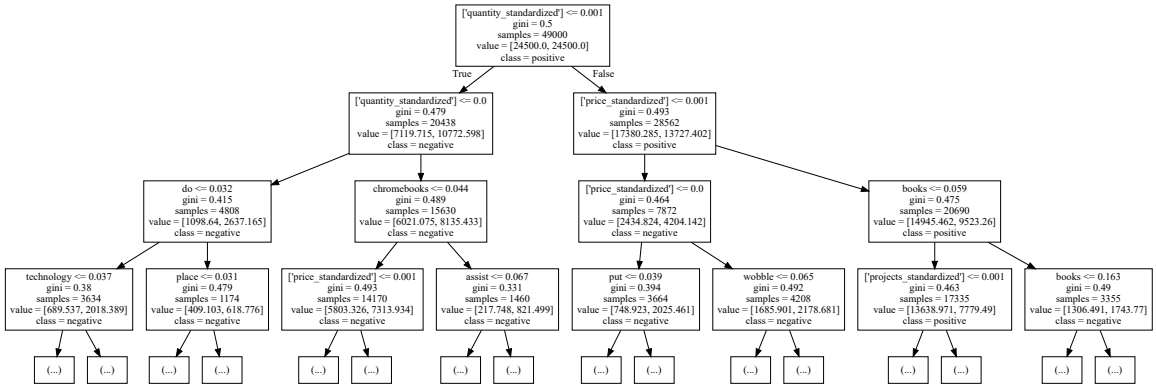
```
feature_names_tfidf=[]
feature_names_tfidf.extend(statevectorizer.get_feature_names())
feature_names_tfidf.extend(prefixvectorizer.get_feature_names())
feature_names_tfidf.extend(gradevectorizer.get_feature_names())
feature_names_tfidf.extend(categoryvectorizer.get_feature_names())
feature_names_tfidf.extend(subcategoryvectorizer.get_feature_names())
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_weigh
t='balanced')
tfidf_Decision_tree.fit(final_train_tfidf,y_train)

dot_data=export_graphviz(tfidf_Decision_tree,out_file=None,max_depth=3,feature_names=fe
ature_names_tfidf, class_names=['positive','negative'])
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]:



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
```

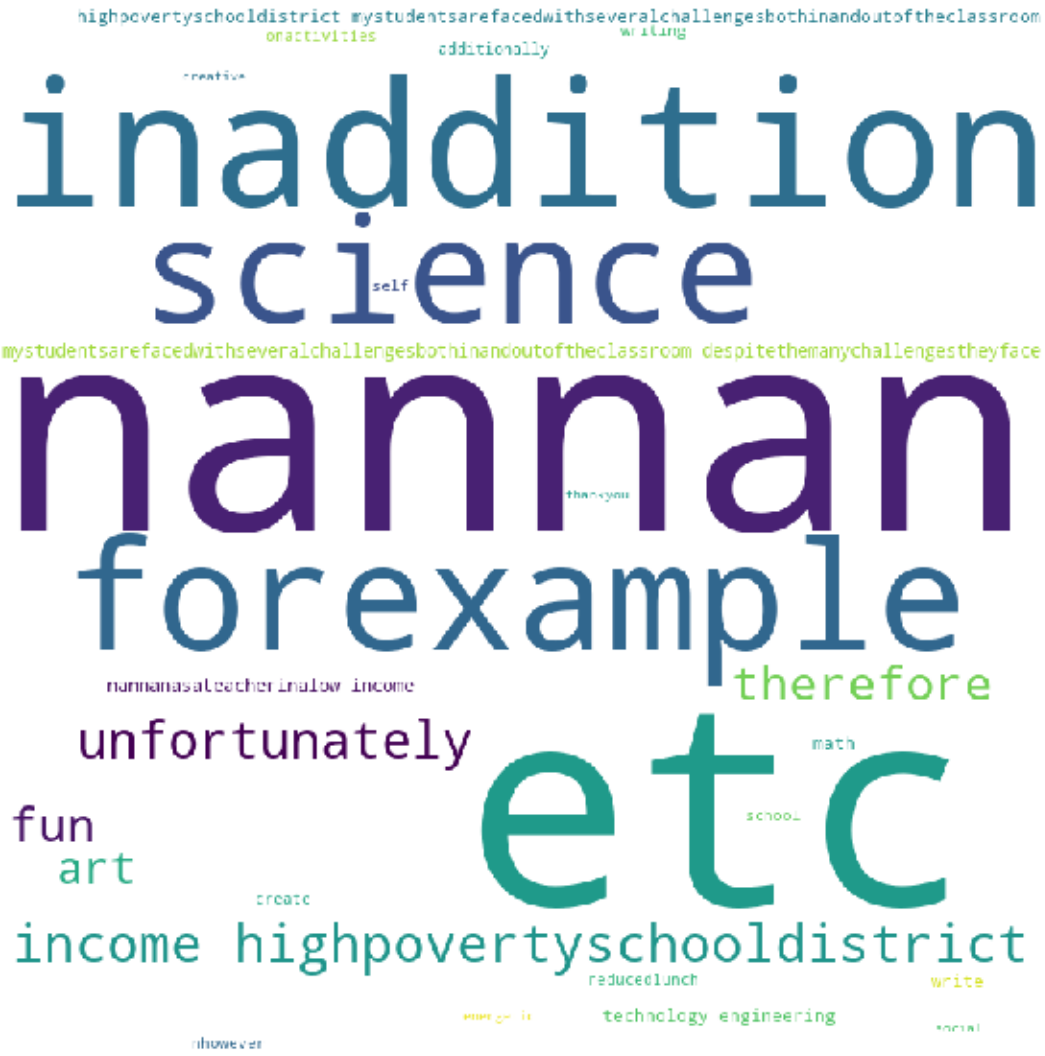
Out[194]:

```
(1000, 15)
```

WORD CLOUD FOR FALSE POSITIVES

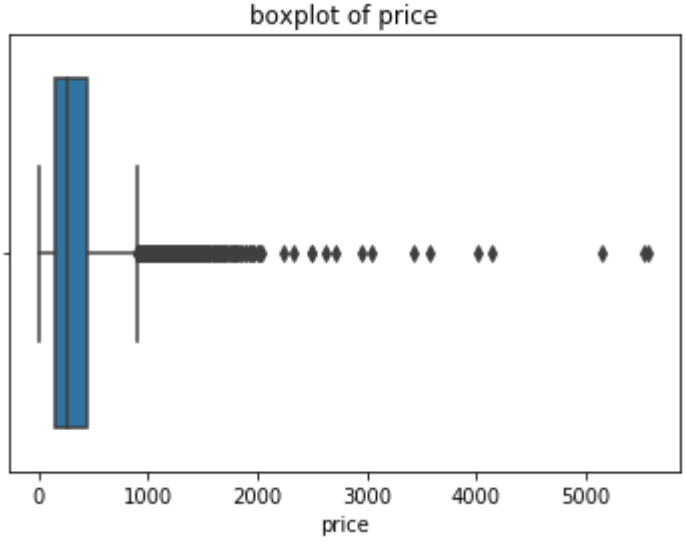
In [195]:

```
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()
```



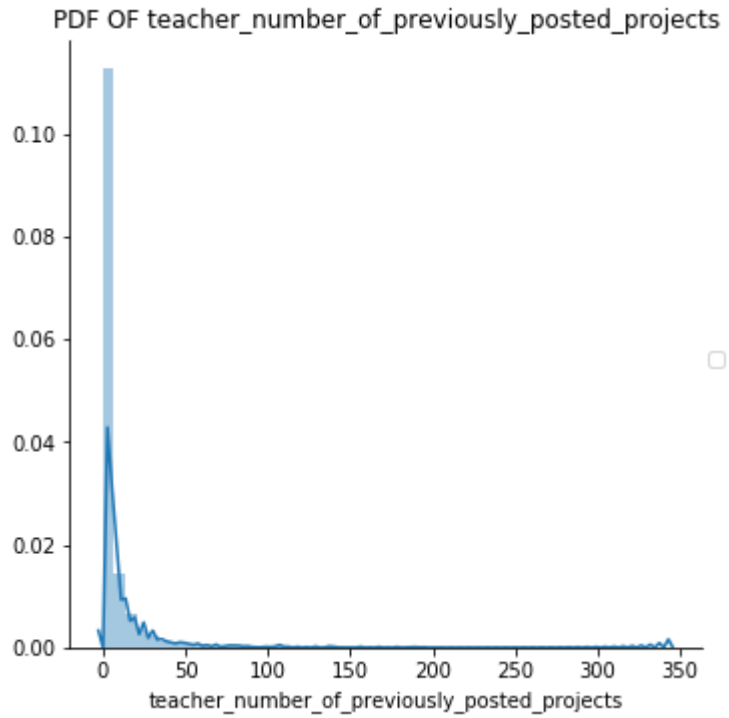
In [196]:

```
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_e
ncoded,X_train_category_encoded,X_train_subcategories_encoded,price_train_norm,projects
_train_norm,train_quantity_norm,avg_w2v_train,title_train_avgw2v)).tocsr()
final_cv_avgw2v=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_cv_
category_encoded,X_cv_subcategories_encoded,price_cv_norm,projects_cv_norm,cv_quantity_
norm,avg_w2v_cv,title_cv_avgw2v)).tocsr()
final_test_avgw2v=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_encod
ed,X_test_category_encoded,X_test_subcategories_encoded,price_test_norm,projects_test_n
orm,test_quantity_norm,avg_w2v_test,title_test_avgw2v)).tocsr()
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_sa
mples_split=j)
        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
_score(y_train,y_tr_pred)))
        print('for {} max depth and {} min samples the cv_auc={}'.format(i,j,roc_auc_sc
ore(y_cv,y_cv_pred)))

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

 25%|████████████████████
| 2/8 [00:59<02:57, 29.66s/it]

for 4 max depth and 20 min samples the train_auc=0.6832056713360246
for 4 max depth and 20 min samples the cv_auc=0.6509459621725895

 38%|██████████████████████████████
| 3/8 [01:28<02:27, 29.54s/it]

for 4 max depth and 30 min samples the train_auc=0.6832056713360246
for 4 max depth and 30 min samples the cv_auc=0.6509459621725895

 50%|████████████████████████████████████████
| 4/8 [01:58<01:58, 29.64s/it]

for 4 max depth and 40 min samples the train_auc=0.6832056713360246
for 4 max depth and 40 min samples the cv_auc=0.6509459621725895

 62%|██████████████████████████████████████████████████
| 5/8 [02:27<01:28, 29.55s/it]

for 4 max depth and 50 min samples the train_auc=0.6832056713360246
for 4 max depth and 50 min samples the cv_auc=0.6509459621725895

 75%|████████████████████████████████████████████████████████████
| 6/8 [02:56<00:58, 29.45s/it]

for 4 max depth and 60 min samples the train_auc=0.6832056713360246
for 4 max depth and 60 min samples the cv_auc=0.6509459621725895

 88%|████████████████████████████████████████████████████████████
████           | 7/8 [03:26<00:29, 29.41s/it]

for 4 max depth and 70 min samples the train_auc=0.6832056713360246
for 4 max depth and 70 min samples the cv_auc=0.6509459621725895

 12%|██████████
| 1/8 [03:55<27:30, 235.81s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 6 max depth and 2 min samples the train_auc=0.7294107816224976
for 6 max depth and 2 min samples the cv_auc=0.6632054207007808

 12%|██████████
| 1/8 [00:48<05:40, 48.58s/it]

for 6 max depth and 10 min samples the train_auc=0.7294839024981055
for 6 max depth and 10 min samples the cv_auc=0.6632705389270478

 25%|████████████████████
| 2/8 [01:35<04:49, 48.21s/it]
```

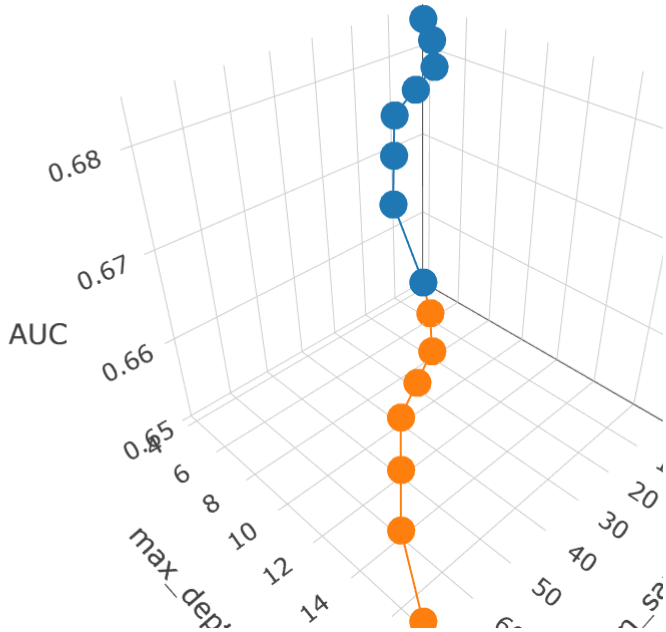
```
for 6 max depth and 20 min samples the train_auc=0.7294698658942184
for 6 max depth and 20 min samples the cv_auc=0.6635186125598879

 38%|████████████████████████████
| 3/8 [02:24<04:02, 48.42s/it]

for 6 max depth and 30 min samples the train_auc=0.7290523129634761
for 6 max depth and 30 min samples the cv_auc=0.6636068361156267

 50%|████████████████████████████████████
| 4/8 [03:28<03:32, 53.11s/it]

for 6 max depth and 40 min samples the train_auc=0.7287717995244262
for 6 max depth and 40 min samples the cv_auc=0.6631501795619216

 62%|████████████████████████████████████████████
| 5/8 [04:33<02:50, 56.68s/it]

for 6 max depth and 50 min samples the train_auc=0.7286931592365595
for 6 max depth and 50 min samples the cv_auc=0.6634761587216904

 75%|████████████████████████████████████████████████████
| 6/8 [05:38<01:58, 59.18s/it]

for 6 max depth and 60 min samples the train_auc=0.728119862679096
for 6 max depth and 60 min samples the cv_auc=0.6639117470952014

 88%|████████████████████████████████████████████████████████
████           | 7/8 [06:40<00:59, 59.80s/it]

for 6 max depth and 70 min samples the train_auc=0.7280502691743507
for 6 max depth and 70 min samples the cv_auc=0.6638455176966012

 25%|██████████████████
| 2/8 [11:32<30:12, 302.03s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 8 max depth and 2 min samples the train_auc=0.7974157386247862
for 8 max depth and 2 min samples the cv_auc=0.6612361587611988

 12%|████████
| 1/8 [01:19<09:16, 79.49s/it]

for 8 max depth and 10 min samples the train_auc=0.7962649492665602
for 8 max depth and 10 min samples the cv_auc=0.6617913216241402

 25%|████████████████
| 2/8 [02:33<07:46, 77.79s/it]

for 8 max depth and 20 min samples the train_auc=0.7946856394130917
for 8 max depth and 20 min samples the cv_auc=0.6613850823103552

 38%|████████████████████████
| 3/8 [03:52<06:31, 78.22s/it]

for 8 max depth and 30 min samples the train_auc=0.7922711443403914
for 8 max depth and 30 min samples the cv_auc=0.661688123698385

 50%|████████████████████████████████
| 4/8 [05:04<05:05, 76.35s/it]

for 8 max depth and 40 min samples the train_auc=0.7907140838078182
for 8 max depth and 40 min samples the cv_auc=0.6618607522573201

 62%|████████████████████████████████████████
| 5/8 [06:18<03:47, 75.72s/it]
```

```
for 8 max depth and 50 min samples the train_auc=0.7886980034925442
for 8 max depth and 50 min samples the cv_auc=0.6630268658894011

 75%|███████████████████████████████████████████████████████████
| 6/8 [07:35<02:32, 76.04s/it]

for 8 max depth and 60 min samples the train_auc=0.7874932816379481
for 8 max depth and 60 min samples the cv_auc=0.6633595914526226

 88%|████████████████████████████████████████████████████████████
████           | 7/8 [09:01<01:18, 78.97s/it]

for 8 max depth and 70 min samples the train_auc=0.7860166088832142
for 8 max depth and 70 min samples the cv_auc=0.6631197105110341

 38%|████████████████████████████
| 3/8 [21:58<33:15, 399.18s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 9 max depth and 2 min samples the train_auc=0.8309915380543198
for 9 max depth and 2 min samples the cv_auc=0.6543415549033071

 12%|█████████
| 1/8 [01:32<10:50, 92.99s/it]

for 9 max depth and 10 min samples the train_auc=0.8294211664744006
for 9 max depth and 10 min samples the cv_auc=0.6543819186792584

 25%|██████████████████
| 2/8 [03:06<09:19, 93.23s/it]

for 9 max depth and 20 min samples the train_auc=0.8262895753681638
for 9 max depth and 20 min samples the cv_auc=0.6546981706716962

 38%|████████████████████████████
| 3/8 [04:47<07:56, 95.35s/it]

for 9 max depth and 30 min samples the train_auc=0.8217059836686811
for 9 max depth and 30 min samples the cv_auc=0.6556712313390951

 50%|████████████████████████████████████
| 4/8 [06:49<06:53, 103.46s/it]

for 9 max depth and 40 min samples the train_auc=0.819090794474177
for 9 max depth and 40 min samples the cv_auc=0.6557685735630302

 62%|█████████████████████████████████████████████
| 5/8 [08:39<05:16, 105.55s/it]

for 9 max depth and 50 min samples the train_auc=0.8167948679727437
for 9 max depth and 50 min samples the cv_auc=0.6564136796531788

 75%|████████████████████████████████████████████████████████
| 6/8 [10:28<03:33, 106.60s/it]

for 9 max depth and 60 min samples the train_auc=0.814834950268476
for 9 max depth and 60 min samples the cv_auc=0.6566609507726668

 88%|████████████████████████████████████████████████████████████
███            | 7/8 [12:13<01:46, 106.08s/it]

for 9 max depth and 70 min samples the train_auc=0.8128181184839274
for 9 max depth and 70 min samples the cv_auc=0.6562168169547825
```

```
 50%|
| 4/8 [36:07<35:36, 534.12s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 10 max depth and 2 min samples the train_auc=0.8644346027450318
for 10 max depth and 2 min samples the cv_auc=0.6456241610207941

 12%|
| 1/8 [02:20<16:23, 140.46s/it]

for 10 max depth and 10 min samples the train_auc=0.8621355974869365
for 10 max depth and 10 min samples the cv_auc=0.6465258861881561

 25%|
| 2/8 [04:41<14:04, 140.77s/it]

for 10 max depth and 20 min samples the train_auc=0.8572938145488624
for 10 max depth and 20 min samples the cv_auc=0.6458587153902446

 38%|
| 3/8 [06:50<11:26, 137.25s/it]

for 10 max depth and 30 min samples the train_auc=0.851758616576679
for 10 max depth and 30 min samples the cv_auc=0.6483317705194198

 50%|
| 4/8 [09:19<09:22, 140.66s/it]

for 10 max depth and 40 min samples the train_auc=0.848315125232808
for 10 max depth and 40 min samples the cv_auc=0.6477607514036575

 62%|
| 5/8 [11:54<07:15, 145.06s/it]

for 10 max depth and 50 min samples the train_auc=0.8447781537024418
for 10 max depth and 50 min samples the cv_auc=0.6487700927296242

 75%|
| 6/8 [14:12<04:45, 142.81s/it]

for 10 max depth and 60 min samples the train_auc=0.8418386261581713
for 10 max depth and 60 min samples the cv_auc=0.6493082440732714

 88%|
           | 7/8 [16:29<02:21, 141.22s/it]

for 10 max depth and 70 min samples the train_auc=0.8388857600342754
for 10 max depth and 70 min samples the cv_auc=0.6493861672378564

 62%|
| 5/8 [55:21<36:00, 720.16s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 12 max depth and 2 min samples the train_auc=0.9149672788251986
for 12 max depth and 2 min samples the cv_auc=0.6327021439417579

 12%|
| 1/8 [02:25<17:01, 145.91s/it]

for 12 max depth and 10 min samples the train_auc=0.9121826851347237
for 12 max depth and 10 min samples the cv_auc=0.630917301334204

 25%|
| 2/8 [05:24<15:34, 155.79s/it]
```

```
for 12 max depth and 20 min samples the train_auc=0.905565332537916
for 12 max depth and 20 min samples the cv_auc=0.6318021119893802

 38%|████████████████████████████▌
| 3/8 [08:11<13:15, 159.03s/it]

for 12 max depth and 30 min samples the train_auc=0.8980421711859705
for 12 max depth and 30 min samples the cv_auc=0.6329901139526629

 50%|████████████████████████████████████▌
| 4/8 [10:59<10:46, 161.73s/it]

for 12 max depth and 40 min samples the train_auc=0.8920527845363586
for 12 max depth and 40 min samples the cv_auc=0.6357645437113678

 62%|████████████████████████████████████████████▊
| 5/8 [13:47<08:10, 163.64s/it]

for 12 max depth and 50 min samples the train_auc=0.8869384611834571
for 12 max depth and 50 min samples the cv_auc=0.6363151472402111

 75%|████████████████████████████████████████████████████▉
| 6/8 [18:01<06:21, 190.69s/it]

for 12 max depth and 60 min samples the train_auc=0.8820082478028496
for 12 max depth and 60 min samples the cv_auc=0.6358466205439608

 88%|████████████████████████████████████████████████████████████▏
███▍          | 7/8 [21:09<03:09, 189.91s/it]

for 12 max depth and 70 min samples the train_auc=0.8768347268100177
for 12 max depth and 70 min samples the cv_auc=0.6372263085535788

 75%|████████████████████████████████████████████████████▉
| 6/8 [1:19:30<31:17, 938.84s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 14 max depth and 2 min samples the train_auc=0.9477847502033858
for 14 max depth and 2 min samples the cv_auc=0.6169628375997295

 12%|████████▍
| 1/8 [04:32<31:47, 272.51s/it]

for 14 max depth and 10 min samples the train_auc=0.9441033751063442
for 14 max depth and 10 min samples the cv_auc=0.6171063816637798

 25%|█████████████████▌
| 2/8 [08:09<25:34, 255.78s/it]

for 14 max depth and 20 min samples the train_auc=0.9360941200237137
for 14 max depth and 20 min samples the cv_auc=0.6216648250602026

 38%|████████████████████████████▌
| 3/8 [12:21<21:13, 254.70s/it]

for 14 max depth and 30 min samples the train_auc=0.9272830520836489
for 14 max depth and 30 min samples the cv_auc=0.6236145797521542

 50%|████████████████████████████████████▌
| 4/8 [15:51<16:05, 241.31s/it]

for 14 max depth and 40 min samples the train_auc=0.9190493992061453
for 14 max depth and 40 min samples the cv_auc=0.6273656418036153

 62%|████████████████████████████████████████████▊
| 5/8 [18:57<11:14, 224.84s/it]
```

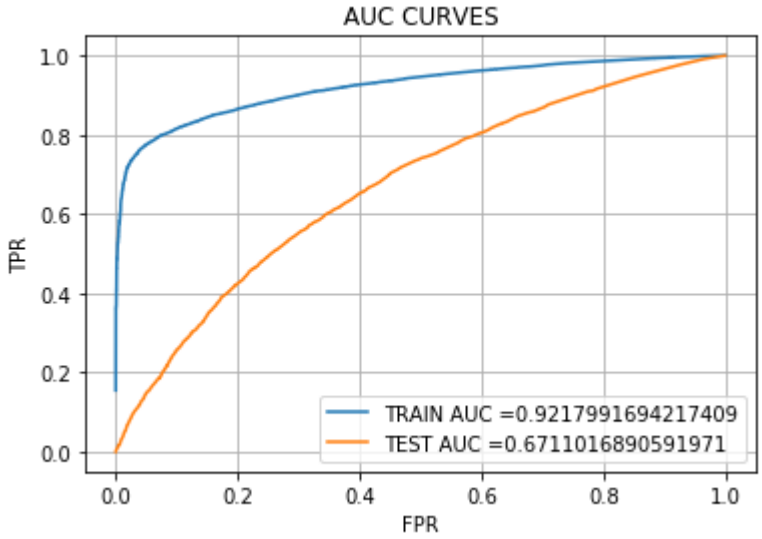
```
for 14 max depth and 50 min samples the train_auc=0.9130065464893682
for 14 max depth and 50 min samples the cv_auc=0.6272333241076633

 75%|███████████████████████████████████████████████████████████▊
| 6/8 [22:16<07:13, 217.00s/it]

for 14 max depth and 60 min samples the train_auc=0.9058794062060815
for 14 max depth and 60 min samples the cv_auc=0.6293023005359237

 88%|████████████████████████████████████████████████████████████
███▊          | 7/8 [25:18<03:26, 206.51s/it]

for 14 max depth and 70 min samples the train_auc=0.8999913432035195
for 14 max depth and 70 min samples the cv_auc=0.6279816281235671

 88%|████████████████████████████████████████████████████████████
▊          | 7/8 [1:47:23<19:19, 1159.02s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 17 max depth and 2 min samples the train_auc=0.9711102139690857
for 17 max depth and 2 min samples the cv_auc=0.6088378396721541

 12%|████████▊
| 1/8 [03:16<22:56, 196.58s/it]

for 17 max depth and 10 min samples the train_auc=0.9689122975238919
for 17 max depth and 10 min samples the cv_auc=0.6058571287067033

 25%|█████████████████▊
| 2/8 [06:24<19:23, 193.88s/it]

for 17 max depth and 20 min samples the train_auc=0.961095784592939
for 17 max depth and 20 min samples the cv_auc=0.6118781130022048

 38%|██████████████████████████▊
| 3/8 [09:30<15:58, 191.62s/it]

for 17 max depth and 30 min samples the train_auc=0.9516541740082299
for 17 max depth and 30 min samples the cv_auc=0.6148357715511522

 50%|███████████████████████████████████▊
| 4/8 [12:42<12:46, 191.67s/it]

for 17 max depth and 40 min samples the train_auc=0.9427601869401611
for 17 max depth and 40 min samples the cv_auc=0.6196233633754353

 62%|████████████████████████████████████████████▊
| 5/8 [15:43<09:25, 188.53s/it]

for 17 max depth and 50 min samples the train_auc=0.9348109230749277
for 17 max depth and 50 min samples the cv_auc=0.6206861467993404

 75%|███████████████████████████████████████████████████████▊
| 6/8 [18:47<06:14, 187.12s/it]

for 17 max depth and 60 min samples the train_auc=0.9276885037677958
for 17 max depth and 60 min samples the cv_auc=0.6204900601578118

 88%|████████████████████████████████████████████████████████████
███▊          | 7/8 [22:12<03:12, 192.54s/it]

for 17 max depth and 70 min samples the train_auc=0.9217991694217409
for 17 max depth and 70 min samples the cv_auc=0.621953094911261

100%|████████████████████████████████████████████████████████████
██████████| 8/8 [2:12:56<00:00, 1271.34s/it]
```



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='bal
anced')
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)))
```

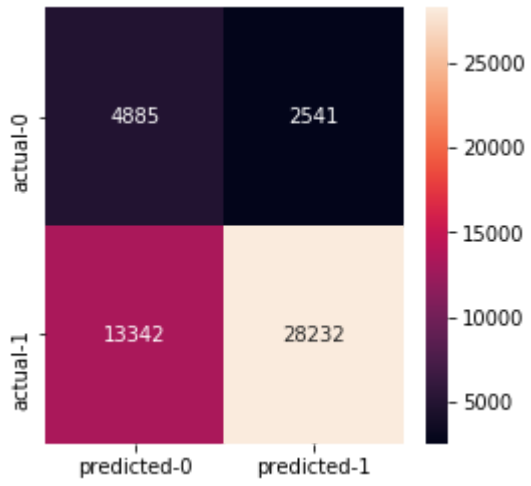
```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.7397290248438463 for threshold 0.834
TRAIN CONFUSION MATRIX
[[ 4885  2541]
 [13342 28232]]
test confusion matrix
[[ 2989  1557]
 [10284 15170]]
```

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','actua
l-1'])
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[212]:

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



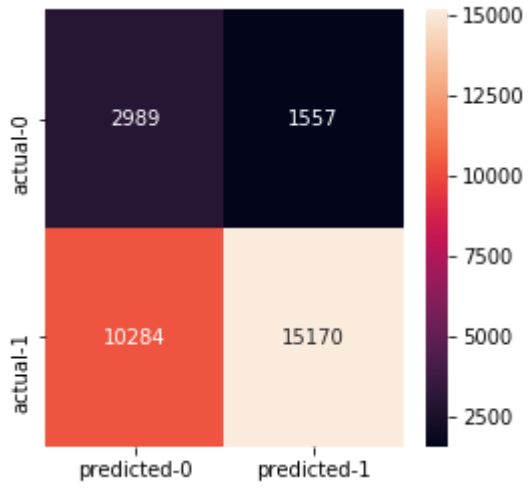
OBSERVATIONS: We got a decent tnr and tpr values.

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','actua
l-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((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_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
_encoded,X_train_category_encoded,X_train_subcategories_encoded,price_train_norm,projec
ts_train_norm,train_quantity_norm,tfidf_w2v_train,title_train_tfidfw2v)).tocsr()
final_cv_tfidfw2v=hstack((X_cv_state_encoded,X_cv_prefix_encoded,X_cv_grade_encoded,X_c
v_category_encoded,X_cv_subcategories_encoded,price_cv_norm,projects_cv_norm,cv_quantit
y_norm,tfidf_w2v_cv,title_cv_tfidfw2v)).tocsr()
final_test_tfidfw2v=hstack((X_test_state_encoded,X_test_prefix_encoded,X_test_grade_enc
oded,X_test_category_encoded,X_test_subcategories_encoded,price_test_norm,projects_test
_norm,test_quantity_norm,tfidf_w2v_test,title_test_tfidfw2v)).tocsr()
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_sa
mples_split=j)
        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
_score(y_train,y_tr_pred)))
        print('for {} max depth and {} min samples the cv_auc={}'.format(i,j,roc_auc_sc
ore(y_cv,y_cv_pred)))

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]

for 4 max depth and 20 min samples the train_auc=0.6860884467545991
for 4 max depth and 20 min samples the cv_auc=0.6566151281421572

 38%|
| 3/8 [01:46<02:59, 35.98s/it]

for 4 max depth and 30 min samples the train_auc=0.6860884467545991
for 4 max depth and 30 min samples the cv_auc=0.6566151281421572

 50%|
| 4/8 [02:16<02:17, 34.42s/it]

for 4 max depth and 40 min samples the train_auc=0.6860884467545991
for 4 max depth and 40 min samples the cv_auc=0.6566151281421572

 62%|
| 5/8 [02:47<01:39, 33.27s/it]

for 4 max depth and 50 min samples the train_auc=0.6860884467545991
for 4 max depth and 50 min samples the cv_auc=0.6566151281421572


 75%|
| 6/8 [03:17<01:04, 32.22s/it]

for 4 max depth and 60 min samples the train_auc=0.6860884467545991
for 4 max depth and 60 min samples the cv_auc=0.6566151281421572

 88%|
          | 7/8 [03:47<00:31, 31.55s/it]

for 4 max depth and 70 min samples the train_auc=0.6860884467545991
for 4 max depth and 70 min samples the cv_auc=0.6566151281421572

 12%|
| 1/8 [04:17<30:02, 257.43s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 6 max depth and 2 min samples the train_auc=0.7299943201879201
for 6 max depth and 2 min samples the cv_auc=0.6643930434544577

 12%|
| 1/8 [00:51<06:00, 51.48s/it]

for 6 max depth and 10 min samples the train_auc=0.7298133699495807
for 6 max depth and 10 min samples the cv_auc=0.664301512838203

 25%|
| 2/8 [01:43<05:10, 51.71s/it]
```

```
for 6 max depth and 20 min samples the train_auc=0.7292977097898476
for 6 max depth and 20 min samples the cv_auc=0.6641491323084538

 38%|████████████████████████████
| 3/8 [02:35<04:18, 51.79s/it]

for 6 max depth and 30 min samples the train_auc=0.7290547779122606
for 6 max depth and 30 min samples the cv_auc=0.6641970890953499

 50%|██████████████████████████████████████
| 4/8 [03:25<03:24, 51.13s/it]

for 6 max depth and 40 min samples the train_auc=0.7291166056946523
for 6 max depth and 40 min samples the cv_auc=0.6641371563399718

 62%|███████████████████████████████████████████████
| 5/8 [04:19<02:36, 52.07s/it]

for 6 max depth and 50 min samples the train_auc=0.7289228610440932
for 6 max depth and 50 min samples the cv_auc=0.6644035202221724

 75%|████████████████████████████████████████████████████████
| 6/8 [05:25<01:52, 56.34s/it]

for 6 max depth and 60 min samples the train_auc=0.7289444576880107
for 6 max depth and 60 min samples the cv_auc=0.6643040791171633

 88%|████████████████████████████████████████████████████████████████
████           | 7/8 [06:28<00:58, 58.28s/it]

for 6 max depth and 70 min samples the train_auc=0.7286451154089022
for 6 max depth and 70 min samples the cv_auc=0.664599439305968

 25%|███████████████████▌
| 2/8 [11:45<31:27, 314.57s/it]
  0%|
| 0/8 [00:00<?, ?it/s]


for 8 max depth and 2 min samples the train_auc=0.7911198772161396
for 8 max depth and 2 min samples the cv_auc=0.6634716963947008

 12%|██████████
| 1/8 [01:25<09:59, 85.63s/it]

for 8 max depth and 10 min samples the train_auc=0.7904948701792129
for 8 max depth and 10 min samples the cv_auc=0.6651210553470355

 25%|███████████████████
| 2/8 [02:45<08:23, 83.90s/it]

for 8 max depth and 20 min samples the train_auc=0.7880290776112414
for 8 max depth and 20 min samples the cv_auc=0.6645659189405779

 38%|████████████████████████████
| 3/8 [04:01<06:47, 81.49s/it]

for 8 max depth and 30 min samples the train_auc=0.7865951511496878
for 8 max depth and 30 min samples the cv_auc=0.6645958676806109

 50%|██████████████████████████████████████
| 4/8 [05:17<05:19, 79.97s/it]

for 8 max depth and 40 min samples the train_auc=0.7842703189939133
for 8 max depth and 40 min samples the cv_auc=0.6653108541641695

 62%|███████████████████████████████████████████████
| 5/8 [06:32<03:55, 78.36s/it]
```

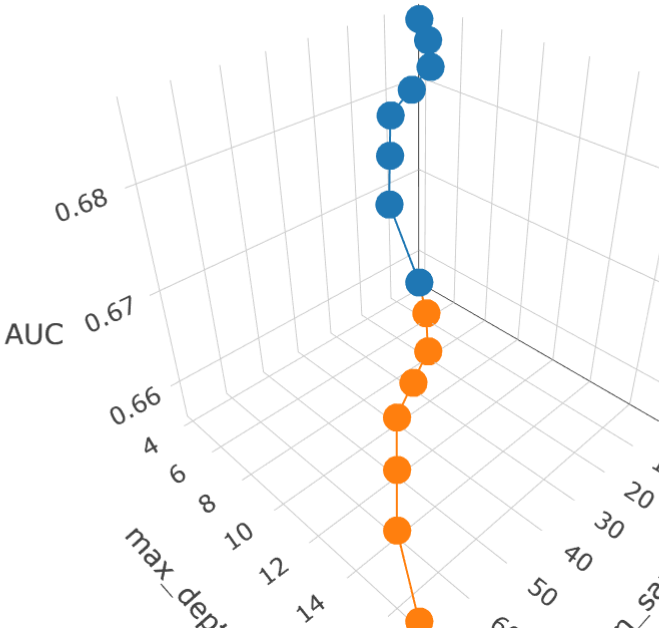
```
for 8 max depth and 50 min samples the train_auc=0.7826865100420718
for 8 max depth and 50 min samples the cv_auc=0.665972901222988

 75%|
| 6/8 [07:47<02:34, 77.50s/it]

for 8 max depth and 60 min samples the train_auc=0.7819206381461534
for 8 max depth and 60 min samples the cv_auc=0.6664255981229019

 88%|
          | 7/8 [09:02<01:16, 76.77s/it]

for 8 max depth and 70 min samples the train_auc=0.7806230563911225
for 8 max depth and 70 min samples the cv_auc=0.6661020353220167

 38%|
| 3/8 [22:06<33:53, 406.62s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 9 max depth and 2 min samples the train_auc=0.8267061063654746
for 9 max depth and 2 min samples the cv_auc=0.6570542264092294

 12%|
| 1/8 [01:33<10:53, 93.42s/it]

for 9 max depth and 10 min samples the train_auc=0.8246433653146996
for 9 max depth and 10 min samples the cv_auc=0.6561726169886326

 25%|
| 2/8 [03:06<09:19, 93.23s/it]

for 9 max depth and 20 min samples the train_auc=0.8214365317277906
for 9 max depth and 20 min samples the cv_auc=0.6550883068054755


 38%|
| 3/8 [04:40<07:48, 93.65s/it]

for 9 max depth and 30 min samples the train_auc=0.8189365230146339
for 9 max depth and 30 min samples the cv_auc=0.6557515355872517

 50%|
| 4/8 [06:10<06:10, 92.57s/it]

for 9 max depth and 40 min samples the train_auc=0.8161223726123861
for 9 max depth and 40 min samples the cv_auc=0.6556759935062383

 62%|
| 5/8 [07:43<04:37, 92.51s/it]

for 9 max depth and 50 min samples the train_auc=0.8132185884450378
for 9 max depth and 50 min samples the cv_auc=0.6577650151306397

 75%|
| 6/8 [09:17<03:05, 92.99s/it]

for 9 max depth and 60 min samples the train_auc=0.8113482235285782
for 9 max depth and 60 min samples the cv_auc=0.6586317030941882

 88%|
          | 7/8 [10:46<01:31, 91.72s/it]

for 9 max depth and 70 min samples the train_auc=0.8096949927438515
for 9 max depth and 70 min samples the cv_auc=0.6583303108270022
```

```
 50%|
| 4/8 [34:20<33:39, 504.90s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 10 max depth and 2 min samples the train_auc=0.8606269160280118
for 10 max depth and 2 min samples the cv_auc=0.6494189203652068

 12%|
| 1/8 [01:46<12:22, 106.01s/it]

for 10 max depth and 10 min samples the train_auc=0.8583953697132307
for 10 max depth and 10 min samples the cv_auc=0.6487117826386061

 25%|
| 2/8 [03:31<10:35, 105.87s/it]

for 10 max depth and 20 min samples the train_auc=0.8537007273743193
for 10 max depth and 20 min samples the cv_auc=0.6480944911321039

 38%|
| 3/8 [05:16<08:47, 105.57s/it]

for 10 max depth and 30 min samples the train_auc=0.8496544021957622
for 10 max depth and 30 min samples the cv_auc=0.6484289451856219

 50%|
| 4/8 [07:00<07:00, 105.17s/it]

for 10 max depth and 40 min samples the train_auc=0.8460731377059283
for 10 max depth and 40 min samples the cv_auc=0.6499813411236273

 62%|
| 5/8 [08:44<05:14, 104.83s/it]

for 10 max depth and 50 min samples the train_auc=0.8419823819065064
for 10 max depth and 50 min samples the cv_auc=0.6510540369102523

 75%|
| 6/8 [10:28<03:28, 104.45s/it]

for 10 max depth and 60 min samples the train_auc=0.8393568810635715
for 10 max depth and 60 min samples the cv_auc=0.6517137205231555

 88%|
          | 7/8 [12:11<01:44, 104.01s/it]

for 10 max depth and 70 min samples the train_auc=0.8362374883766813
for 10 max depth and 70 min samples the cv_auc=0.651200517644041

 62%|
| 5/8 [48:14<30:10, 603.56s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 12 max depth and 2 min samples the train_auc=0.9150695709606671
for 12 max depth and 2 min samples the cv_auc=0.633273154238692

 12%|
| 1/8 [02:18<16:06, 138.10s/it]

for 12 max depth and 10 min samples the train_auc=0.9122762317225992
for 12 max depth and 10 min samples the cv_auc=0.632908063576554

 25%|
| 2/8 [04:57<14:26, 144.41s/it]
```

```
for 12 max depth and 20 min samples the train_auc=0.9051805041506304
for 12 max depth and 20 min samples the cv_auc=0.6292705703926262

 38%|████████████████████████████▍
| 3/8 [07:50<12:45, 153.01s/it]

for 12 max depth and 30 min samples the train_auc=0.8985591658514845
for 12 max depth and 30 min samples the cv_auc=0.6322633455148392

 50%|████████████████████████████████████▌
| 4/8 [10:37<10:29, 157.32s/it]

for 12 max depth and 40 min samples the train_auc=0.8920300590689831
for 12 max depth and 40 min samples the cv_auc=0.6346822442210044

 62%|████████████████████████████████████████████▋
| 5/8 [13:16<07:53, 157.86s/it]

for 12 max depth and 50 min samples the train_auc=0.8861341817576922
for 12 max depth and 50 min samples the cv_auc=0.6340424029711973

 75%|████████████████████████████████████████████████████▊
| 6/8 [16:01<05:19, 159.95s/it]

for 12 max depth and 60 min samples the train_auc=0.8811854521093748
for 12 max depth and 60 min samples the cv_auc=0.6351596602959217

 88%|████████████████████████████████████████████████████████████▉
|          | 7/8 [18:49<02:42, 162.23s/it]

for 12 max depth and 70 min samples the train_auc=0.8757969574589746
for 12 max depth and 70 min samples the cv_auc=0.6359916638087784

 75%|████████████████████████████████████████████████████▊
| 6/8 [1:09:46<27:00, 810.14s/it]
  0%|
| 0/8 [00:00<?, ?it/s]


for 14 max depth and 2 min samples the train_auc=0.9506371235072533
for 14 max depth and 2 min samples the cv_auc=0.6205853475948128

 12%|████████▍
| 1/8 [03:40<25:46, 220.97s/it]

for 14 max depth and 10 min samples the train_auc=0.9471785590501511
for 14 max depth and 10 min samples the cv_auc=0.6234444292838992

 25%|████████████████▊
| 2/8 [07:01<21:28, 214.79s/it]

for 14 max depth and 20 min samples the train_auc=0.9379260320630431
for 14 max depth and 20 min samples the cv_auc=0.6210276206399802

 38%|████████████████████████▉
| 3/8 [09:52<16:48, 201.65s/it]

for 14 max depth and 30 min samples the train_auc=0.9300146671902594
for 14 max depth and 30 min samples the cv_auc=0.6197533793572683

 50%|█████████████████████████████████
| 4/8 [12:41<12:48, 192.03s/it]

for 14 max depth and 40 min samples the train_auc=0.9222218595519214
for 14 max depth and 40 min samples the cv_auc=0.6227765441609163
```

```
 62%|
| 5/8 [15:33<09:17, 185.94s/it]

for 14 max depth and 50 min samples the train_auc=0.9158244817702688
for 14 max depth and 50 min samples the cv_auc=0.6255123915469346

 75%|
| 6/8 [18:14<05:56, 178.32s/it]

for 14 max depth and 60 min samples the train_auc=0.9089826390644746
for 14 max depth and 60 min samples the cv_auc=0.6284393870307774

 88%|
           | 7/8 [20:53<02:52, 172.53s/it]

for 14 max depth and 70 min samples the train_auc=0.9022920894734041
for 14 max depth and 70 min samples the cv_auc=0.6253856914444458

 88%|
          | 7/8 [1:33:15<16:29, 989.80s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 17 max depth and 2 min samples the train_auc=0.975293010178742
for 17 max depth and 2 min samples the cv_auc=0.606738526475427

 12%|
| 1/8 [03:10<22:16, 190.99s/it]

for 17 max depth and 10 min samples the train_auc=0.9730685817679742
for 17 max depth and 10 min samples the cv_auc=0.6046755203937516

 25%|
| 2/8 [06:14<18:52, 188.72s/it]

for 17 max depth and 20 min samples the train_auc=0.9637794740987393
for 17 max depth and 20 min samples the cv_auc=0.6047816197139326

 38%|
| 3/8 [09:14<15:30, 186.19s/it]

for 17 max depth and 30 min samples the train_auc=0.955593327359671
for 17 max depth and 30 min samples the cv_auc=0.6068253037433667

 50%|
| 4/8 [12:13<12:15, 183.91s/it]

for 17 max depth and 40 min samples the train_auc=0.9466855028918545
for 17 max depth and 40 min samples the cv_auc=0.6102408975760851

 62%|
| 5/8 [15:08<09:03, 181.32s/it]

for 17 max depth and 50 min samples the train_auc=0.937544624156594
for 17 max depth and 50 min samples the cv_auc=0.6127191029008371

 75%|
| 6/8 [18:08<06:01, 180.85s/it]

for 17 max depth and 60 min samples the train_auc=0.9295560620112964
for 17 max depth and 60 min samples the cv_auc=0.6164932085499737

 88%|
           | 7/8 [21:13<03:02, 182.25s/it]

for 17 max depth and 70 min samples the train_auc=0.921291820771313
for 17 max depth and 70 min samples the cv_auc=0.6178566875536493
```

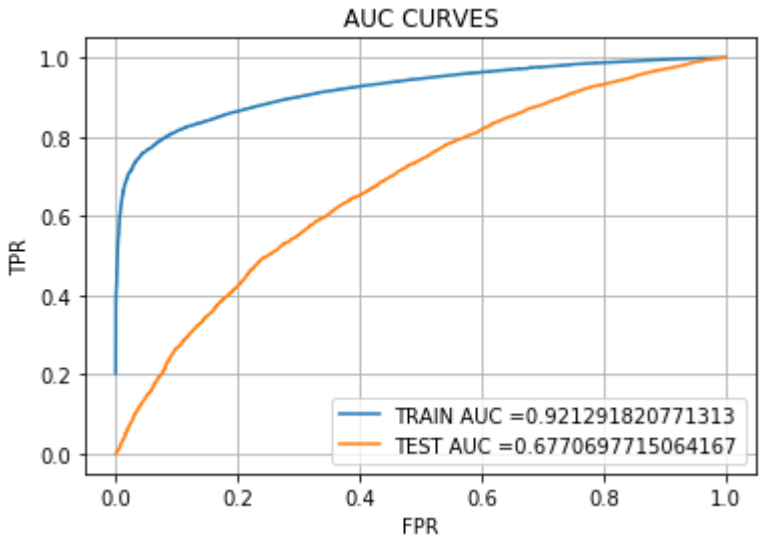
```
100%|██████████| 8/8 [1:57:22<00:00, 1126.72s/it]
```



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='bal
anced')
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)))
```

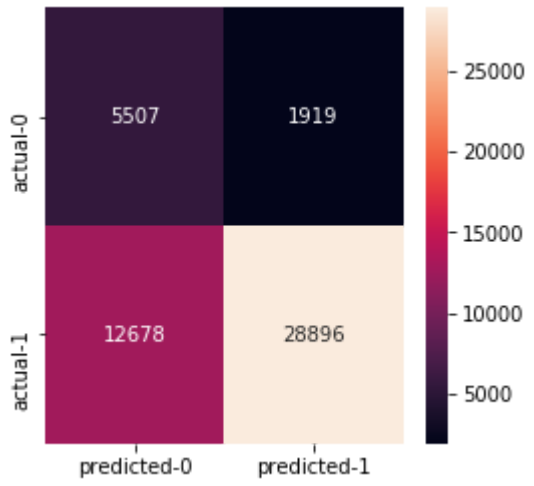
```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.7346660070839455 for threshold 0.834
TRAIN CONFUSION MATRIX
[[ 5507  1919]
 [12678 28896]]
test confusion matrix
[[ 2939  1607]
 [ 9936 15518]]
```

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','actua
l-1'])
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[227]:

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



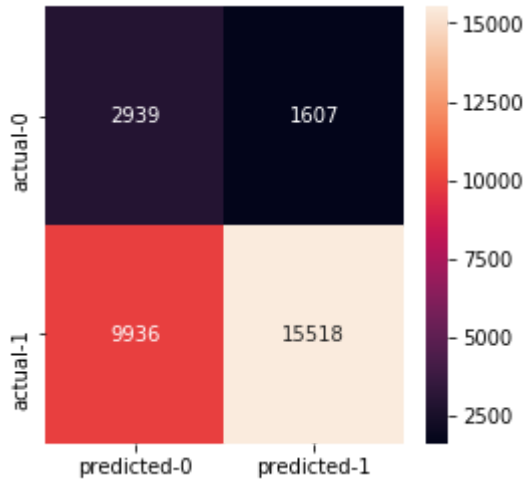
OBSERVATIONS: For training data we got good tnr and tpr values. And the value of tnr is a bit high.

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','actua
l-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 subse
ction
# 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
code
# 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_sa
mples_split=j)
        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
_score(y_train,y_tr_pred)))
        print('for {} max depth and {} min samples the cv_auc={}'.format(i,j,roc_auc_sc
ore(y_cv,y_cv_pred)))


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

 38%|██████████████████████████████
| 3/8 [00:22<00:37,  7.51s/it]

for 4 max depth and 30 min samples the train_auc=0.6750697353769618
for 4 max depth and 30 min samples the cv_auc=0.6484488316428582

 50%|████████████████████████████████████████
| 4/8 [00:30<00:30,  7.63s/it]

for 4 max depth and 40 min samples the train_auc=0.6750697353769618
for 4 max depth and 40 min samples the cv_auc=0.6484488316428582

 62%|██████████████████████████████████████████████████
| 5/8 [00:37<00:22,  7.61s/it]

for 4 max depth and 50 min samples the train_auc=0.6750697353769618
for 4 max depth and 50 min samples the cv_auc=0.6484488316428582

 75%|████████████████████████████████████████████████████████████
| 6/8 [00:45<00:15,  7.54s/it]

for 4 max depth and 60 min samples the train_auc=0.6750697353769618
for 4 max depth and 60 min samples the cv_auc=0.6484488316428582

 88%|████████████████████████████████████████████████████████████
████           | 7/8 [00:53<00:07,  7.64s/it]

for 4 max depth and 70 min samples the train_auc=0.6750697353769618
for 4 max depth and 70 min samples the cv_auc=0.6484488316428582

 12%|██████████
| 1/8 [01:00<07:05, 60.82s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 6 max depth and 2 min samples the train_auc=0.7175808170546626
for 6 max depth and 2 min samples the cv_auc=0.6597974639731474

 12%|██████████
| 1/8 [00:11<01:23, 11.98s/it]

for 6 max depth and 10 min samples the train_auc=0.7173519995191634
for 6 max depth and 10 min samples the cv_auc=0.6599471459415155

 25%|████████████████████
| 2/8 [00:23<01:11, 11.98s/it]
```

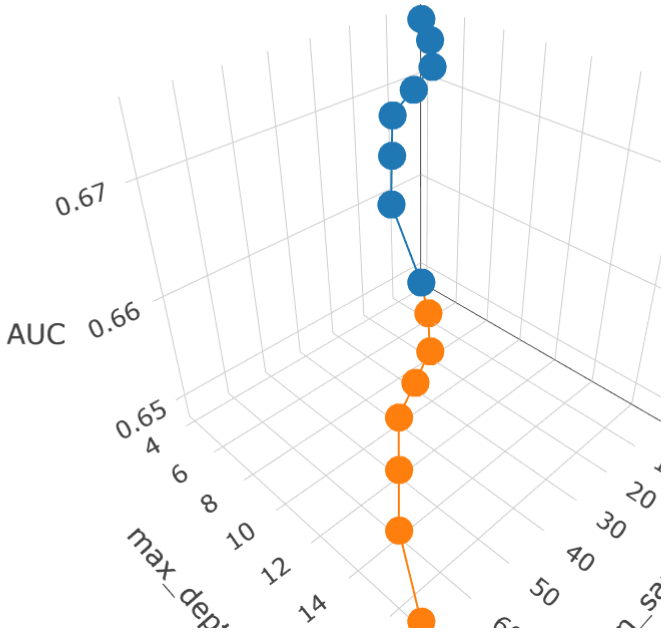
```
for 6 max depth and 20 min samples the train_auc=0.7168706138082661
for 6 max depth and 20 min samples the cv_auc=0.6600769590903032

 38%|████████████████████████████
| 3/8 [00:35<00:59, 11.95s/it]

for 6 max depth and 30 min samples the train_auc=0.7161217973496999
for 6 max depth and 30 min samples the cv_auc=0.6596961620954214

 50%|██████████████████████████████████████
| 4/8 [00:47<00:47, 11.91s/it]

for 6 max depth and 40 min samples the train_auc=0.7157682974573479
for 6 max depth and 40 min samples the cv_auc=0.6598171387785106

 62%|████████████████████████████████████████████████
| 5/8 [00:59<00:35, 11.91s/it]

for 6 max depth and 50 min samples the train_auc=0.7154627636544525
for 6 max depth and 50 min samples the cv_auc=0.6597804083597129

 75%|████████████████████████████████████████████████████████████
| 6/8 [01:11<00:23, 11.86s/it]

for 6 max depth and 60 min samples the train_auc=0.715097651618352
for 6 max depth and 60 min samples the cv_auc=0.659838956559088

 88%|████████████████████████████████████████████████████████████████████
████           | 7/8 [01:23<00:11, 11.82s/it]

for 6 max depth and 70 min samples the train_auc=0.7148450186611199
for 6 max depth and 70 min samples the cv_auc=0.6598930336126456

 25%|████████████████████
| 2/8 [02:36<07:07, 71.17s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 8 max depth and 2 min samples the train_auc=0.7704808545646401
for 8 max depth and 2 min samples the cv_auc=0.674762821147324

 12%|██████████
| 1/8 [00:18<02:07, 18.17s/it]

for 8 max depth and 10 min samples the train_auc=0.7681045500026425
for 8 max depth and 10 min samples the cv_auc=0.6746904097502656

 25%|████████████████████
| 2/8 [00:36<01:48, 18.16s/it]

for 8 max depth and 20 min samples the train_auc=0.7661886272614058
for 8 max depth and 20 min samples the cv_auc=0.6740304703913494

 38%|██████████████████████████████
| 3/8 [00:54<01:30, 18.10s/it]

for 8 max depth and 30 min samples the train_auc=0.7633890009463459
for 8 max depth and 30 min samples the cv_auc=0.6745483913434667

 50%|████████████████████████████████████████
| 4/8 [01:12<01:12, 18.14s/it]


for 8 max depth and 40 min samples the train_auc=0.7624539755840637
for 8 max depth and 40 min samples the cv_auc=0.6743932769770242

 62%|██████████████████████████████████████████████████
| 5/8 [01:30<00:54, 18.04s/it]
```

```
for 8 max depth and 50 min samples the train_auc=0.7612717783731574
for 8 max depth and 50 min samples the cv_auc=0.6745000906222769

 75%|███████████████████████████████████████████████████████████
| 6/8 [01:48<00:35, 17.94s/it]

for 8 max depth and 60 min samples the train_auc=0.7596430075894121
for 8 max depth and 60 min samples the cv_auc=0.6746240480692446

 88%|████████████████████████████████████████████████████████████
████           | 7/8 [02:05<00:17, 17.86s/it]

for 8 max depth and 70 min samples the train_auc=0.7585167332967265
for 8 max depth and 70 min samples the cv_auc=0.6748095256606377

 38%|████████████████████████
| 3/8 [04:59<07:44, 92.80s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 9 max depth and 2 min samples the train_auc=0.7971823604481717
for 9 max depth and 2 min samples the cv_auc=0.6795657930077135

 12%|███████
| 1/8 [00:21<02:31, 21.58s/it]

for 9 max depth and 10 min samples the train_auc=0.7945556854344952
for 9 max depth and 10 min samples the cv_auc=0.6785233017071346

 25%|███████████████
| 2/8 [00:42<02:09, 21.53s/it]

for 9 max depth and 20 min samples the train_auc=0.7908629216910323
for 9 max depth and 20 min samples the cv_auc=0.6788667950593963

 38%|████████████████████████
| 3/8 [01:04<01:48, 21.67s/it]

for 9 max depth and 30 min samples the train_auc=0.7871168408786227
for 9 max depth and 30 min samples the cv_auc=0.6795104107676055

 50%|██████████████████████████████
| 4/8 [01:26<01:26, 21.51s/it]

for 9 max depth and 40 min samples the train_auc=0.7851259590772378
for 9 max depth and 40 min samples the cv_auc=0.6796980577906973

 62%|████████████████████████████████████
| 5/8 [01:47<01:04, 21.39s/it]

for 9 max depth and 50 min samples the train_auc=0.7828826888700443
for 9 max depth and 50 min samples the cv_auc=0.6793949987650113

 75%|███████████████████████████████████████████████
| 6/8 [02:08<00:42, 21.23s/it]

for 9 max depth and 60 min samples the train_auc=0.7809936635462942
for 9 max depth and 60 min samples the cv_auc=0.6788267399424266

 88%|████████████████████████████████████████████████████████████
████           | 7/8 [02:29<00:21, 21.20s/it]

for 9 max depth and 70 min samples the train_auc=0.7793802509158501
for 9 max depth and 70 min samples the cv_auc=0.6797320367351456
```

```
 50%|████████████████████████████████████
| 4/8 [07:49<07:43, 115.90s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 10 max depth and 2 min samples the train_auc=0.8188712763709517
for 10 max depth and 2 min samples the cv_auc=0.684133690187798

 12%|████████▊
| 1/8 [00:25<02:57, 25.38s/it]

for 10 max depth and 10 min samples the train_auc=0.8147097528960426
for 10 max depth and 10 min samples the cv_auc=0.6851346183518119

 25%|█████████████████▌
| 2/8 [00:50<02:32, 25.40s/it]

for 10 max depth and 20 min samples the train_auc=0.8097303992552368
for 10 max depth and 20 min samples the cv_auc=0.6838562322199199

 38%|██████████████████████████▎
| 3/8 [01:15<02:06, 25.22s/it]

for 10 max depth and 30 min samples the train_auc=0.8043573469745219
for 10 max depth and 30 min samples the cv_auc=0.6843241239605512

 50%|████████████████████████████████████
| 4/8 [01:40<01:40, 25.00s/it]

for 10 max depth and 40 min samples the train_auc=0.8015863186000916
for 10 max depth and 40 min samples the cv_auc=0.6845287031334848

 62%|███████████████████████████████████████████▏
| 5/8 [02:04<01:14, 24.77s/it]

for 10 max depth and 50 min samples the train_auc=0.7996660344218793
for 10 max depth and 50 min samples the cv_auc=0.6846139282876891

 75%|████████████████████████████████████████████████████▊
| 6/8 [02:28<00:49, 24.55s/it]

for 10 max depth and 60 min samples the train_auc=0.7975744427813221
for 10 max depth and 60 min samples the cv_auc=0.6844312550836134

 88%|███████████████████████████████████████████████████████████
████           | 7/8 [02:52<00:24, 24.39s/it]

for 10 max depth and 70 min samples the train_auc=0.7954292781835733
for 10 max depth and 70 min samples the cv_auc=0.6843725481453335

 62%|███████████████████████████████████████████▏
| 5/8 [11:06<07:00, 140.22s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 12 max depth and 2 min samples the train_auc=0.858394087032917
for 12 max depth and 2 min samples the cv_auc=0.6859942600717541

 12%|████████▊
| 1/8 [00:33<03:54, 33.43s/it]

for 12 max depth and 10 min samples the train_auc=0.8527256214913268
for 12 max depth and 10 min samples the cv_auc=0.6857026567036956

 25%|█████████████████▌
| 2/8 [01:06<03:19, 33.23s/it]
```

```
for 12 max depth and 20 min samples the train_auc=0.8454289568656765
for 12 max depth and 20 min samples the cv_auc=0.686318792943724

 38%|████████████████████████████
| 3/8 [01:39<02:46, 33.33s/it]

for 12 max depth and 30 min samples the train_auc=0.8387924628564609
for 12 max depth and 30 min samples the cv_auc=0.686626993346159

 50%|██████████████████████████████████████
| 4/8 [02:11<02:11, 32.88s/it]

for 12 max depth and 40 min samples the train_auc=0.8352239798224799
for 12 max depth and 40 min samples the cv_auc=0.6878445207457287

 62%|████████████████████████████████████████████████
| 5/8 [02:42<01:37, 32.36s/it]

for 12 max depth and 50 min samples the train_auc=0.8320054126258836
for 12 max depth and 50 min samples the cv_auc=0.6877088695327764

 75%|██████████████████████████████████████████████████████████
| 6/8 [03:13<01:03, 31.90s/it]

for 12 max depth and 60 min samples the train_auc=0.8281005078105449
for 12 max depth and 60 min samples the cv_auc=0.68791795512684

 88%|████████████████████████████████████████████████████████████████████
████           | 7/8 [03:44<00:31, 31.69s/it]

for 12 max depth and 70 min samples the train_auc=0.8257159921510848
for 12 max depth and 70 min samples the cv_auc=0.6879961869504062

 75%|██████████████████████████████████████████████████████████▌
| 6/8 [15:20<05:49, 174.57s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 14 max depth and 2 min samples the train_auc=0.8883402202253265
for 14 max depth and 2 min samples the cv_auc=0.6823459020916779

 12%|█████████
| 1/8 [00:41<04:52, 41.72s/it]

for 14 max depth and 10 min samples the train_auc=0.8823067981240375
for 14 max depth and 10 min samples the cv_auc=0.6849121105014675

 25%|███████████████████
| 2/8 [01:22<04:07, 41.29s/it]

for 14 max depth and 20 min samples the train_auc=0.8732690747421836
for 14 max depth and 20 min samples the cv_auc=0.685084650872122

 38%|████████████████████████████
| 3/8 [02:01<03:23, 40.63s/it]

for 14 max depth and 30 min samples the train_auc=0.8659887416168905
for 14 max depth and 30 min samples the cv_auc=0.686050621201775

 50%|██████████████████████████████████████
| 4/8 [02:39<02:39, 39.88s/it]

for 14 max depth and 40 min samples the train_auc=0.8609832290715062
for 14 max depth and 40 min samples the cv_auc=0.68607731579426


 62%|████████████████████████████████████████████████
| 5/8 [03:17<01:58, 39.35s/it]
```

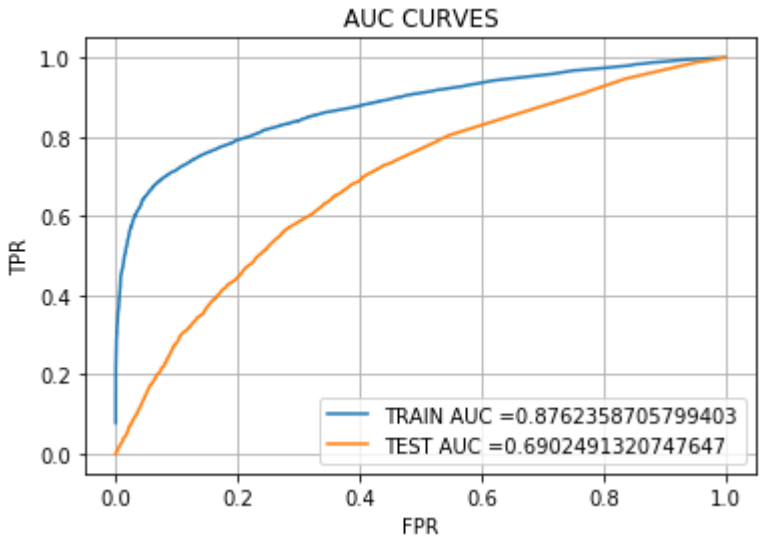
```
for 14 max depth and 50 min samples the train_auc=0.8566453435316524
for 14 max depth and 50 min samples the cv_auc=0.6852275511617255

 75%|
| 6/8 [03:54<01:17, 38.67s/it]

for 14 max depth and 60 min samples the train_auc=0.8526696613235517
for 14 max depth and 60 min samples the cv_auc=0.6862089632592808

 88%|
          | 7/8 [04:31<00:38, 38.26s/it]

for 14 max depth and 70 min samples the train_auc=0.8490982517702188
for 14 max depth and 70 min samples the cv_auc=0.68722378989629

 88%|
          | 7/8 [20:29<03:34, 214.63s/it]
  0%|
| 0/8 [00:00<?, ?it/s]

for 17 max depth and 2 min samples the train_auc=0.9209202013999848
for 17 max depth and 2 min samples the cv_auc=0.674547632924255

 12%|
| 1/8 [00:53<06:16, 53.73s/it]

for 17 max depth and 10 min samples the train_auc=0.9143240373215401
for 17 max depth and 10 min samples the cv_auc=0.6730700030103951

 25%|
| 2/8 [01:44<05:17, 52.93s/it]

for 17 max depth and 20 min samples the train_auc=0.9045047405467465
for 17 max depth and 20 min samples the cv_auc=0.6769810474213782

 38%|
| 3/8 [02:35<04:21, 52.23s/it]

for 17 max depth and 30 min samples the train_auc=0.8966779240650922
for 17 max depth and 30 min samples the cv_auc=0.6784963513686363

 50%|
| 4/8 [03:23<03:24, 51.06s/it]

for 17 max depth and 40 min samples the train_auc=0.8894384391252426
for 17 max depth and 40 min samples the cv_auc=0.6777293690749382

 62%|
| 5/8 [04:10<02:29, 49.83s/it]

for 17 max depth and 50 min samples the train_auc=0.8849294906744672
for 17 max depth and 50 min samples the cv_auc=0.6790009294339251

 75%|
| 6/8 [04:57<01:37, 48.85s/it]

for 17 max depth and 60 min samples the train_auc=0.879909408694611
for 17 max depth and 60 min samples the cv_auc=0.6798599450170765

 88%|
          | 7/8 [05:43<00:47, 47.93s/it]

for 17 max depth and 70 min samples the train_auc=0.8762358705799403
for 17 max depth and 70 min samples the cv_auc=0.6808106146800752

100%|
          | 8/8 [26:59<00:00, 267.38s/it]
```



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='bal
anced')
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)))
```

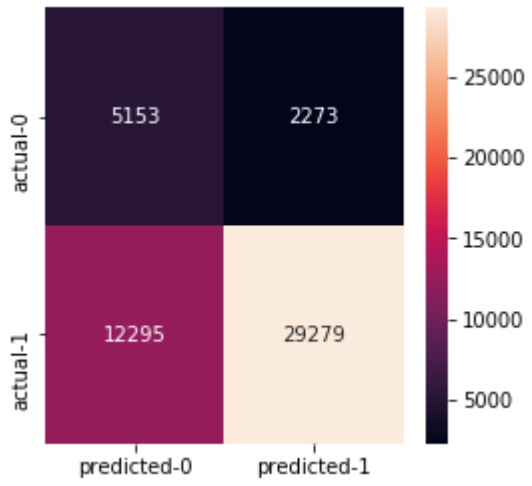
```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.6487601385351746 for threshold 0.83
TRAIN CONFUSION MATRIX
[[ 5153  2273]
 [12295 29279]]
test confusion matrix
[[ 2991  1555]
 [ 9388 16066]]
```

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','actua
l-1'])
plt.figure(figsize=(4,4))
sn.heatmap(train,annot=True,fmt='d')
```

Out[236]:

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



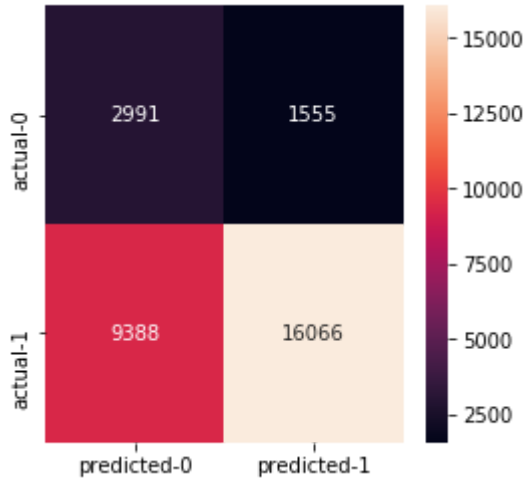
OBSERVATIONS: For test set we got a decent tnr and tpr values.

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','actua
l-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.