DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

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

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [2]:

```
# Citation https://www.kaggle.com/shashank49/donors-choose-knn
# I referenced few parts of my code from above link
```

In [1]:

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

import sqlite3
```

```
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 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 [2]:

```
project_data = pd.read_csv('train_data.csv', nrows = 50000)
resource_data = pd.read_csv('resources.csv')
```

In [3]:

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

```
Number of data points in train data (50000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

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


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


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

```
project_data = project_data[cols]


project_data.head(2)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	L

In [5]:

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

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

Out[5]:

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

1.2 preprocessing of `project_subject_categories`

In [6]:

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

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

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

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

cat_dict = dict(my_counter)
```

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

1.3 preprocessing of `project_subject_subcategories`

In [7]:

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

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

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

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

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

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

1.3 Text preprocessing

In [8]:

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

In [9]:

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

Out[9]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_t
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA	2016-04-27 00:53:00	Grades PreK-2	Flexi Seating Flexi Learn
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA	2016-04-27 01:05:25	Grades 3-5	Going De The Ar In Thinki

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_t

In [10]:

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

In [11]:

```
# 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)
```

```
I recently read an article about giving students a choice about how they learn. We already set goa
ls; why not let them choose where to sit, and give them options of what to sit on?I teach at a low
-income (Title 1) school. Every year, I have a class with a range of abilities, yet they are all t
he same age. They learn differently, and they have different interests. Some have ADHD, and some a
re fast learners. Yet they are eager and active learners that want and need to be able to move aro
und the room, yet have a place that they can be comfortable to complete their work.We need a class
room rug that we can use as a class for reading time, and students can use during other learning t
imes. I have also requested four Kore Kids wobble chairs and four Back Jack padded portable chairs
so that students can still move during whole group lessons without disrupting the class. Having th
ese areas will provide these little ones with a way to wiggle while working.Benjamin Franklin once
said, \"Tell me and I forget, teach me and I may remember, involve me and I learn.\" I want these
children to be involved in their learning by having a choice on where to sit and how to learn, all
by giving them options for comfortable flexible seating.
==================================================
At the beginning of every class we start out with a Math Application problem to help students see
the relevance of topics in math. We are always in groups and do a lot of cooperative activities. W
e also use lots of technology in our class. I love seeing my students grow and love math!I have a
very diverse population of students from all different races, SES, and experiences. My students lo
ve school and are starting to embrace the hard work it takes to be a fifth grader.  My school is a
5th/6th grade school only and is considered a school for the middle grades. It is located in a sub
urban area. It is now more diverse than it has been in many years.I am in an inclusion setting and
many of my students have disabilities. It is hard for them to see the board because our resources
are old and outdated.  A new document camera for our classroom will allow our students to see the
board more clearly during instructional times and will create a classroom environment where lots o
f movement isn't necessary just because my students cannot see the board.It's frustrating to teach
a lesson when many of my students can't see the board because the resources I have are old and out
dated. Oftentimes students will tell me to wait before moving on because it takes them forever to
write notes because they cannot see the materials.  I want students to enjoy coming to my class to
learn math and not feel frustrated because they cannot see the board.
==================================================
My students love coming to school and they love learning. I strive daily to make our classroom a r
elaxed, comfortable and welcoming environment where all learners will excel and grow in their lear
ning. And a new rug will make our days even brighter!My 2nd grade classroom is filled with 20 amaz
ing young learners. These students fill my heart everyday with their passion for learning new thin
gs. Working with these students and how engaged they are in each subject matter is so much fun. We
are small elementary school in mid-Missouri and we have an 80 percent free and reduced lunch rate.
I have a wide range of learners in my classroom, and all of my students learn in different ways. S
o it is important to provide a learning environment that meets all students.A beautiful new carpet
will be the focal point of our classroom. The carpet will be full of students all day long. It wil
l be a clean and comfortable place where my students will find comfort in learning. Students will
be sitting in small groups, laying and reading a book or even dancing on the carpet for brain brea
ks during the day. A carpet in an elementary classroom is the heart of where learning takes
place!Thank you for donating or considering a donation to this project. I want to make my 2nd grad
e classroom as comfortable and inviting as Starbucks or as cozy as a grandma's living room! This b
eautiful carpet will be a perfect addition to a classroom the is filled with so much excitement an
d enthusiasm!
==================================================
I teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school pr
ovides free breakfast for all students. I am a Special Education certified teacher and I teach Kin
dergarten in a general education setting with my class that consists 52% students with special nee
ds. The disabilities include Autism Spectrum Disorder, Speech Impaired, Language Impaired, Other H
ealth Impaired (ADHD), and Developmentally Delayed. I also have about 42% of my students who are E
nglish Language Learners.\r\n\r\n\"Self-motivated learners\" is a synonym of \"my students\".  The
y love to learn and they possess a positive outlook and attitude in school. Almost everyday, my st
```

```
udents would ask me, \"Ms. Perez, what are we going to learn today?\" I could not ask for a better
greeting from my students.This project will greatly impact my students' learning on a daily basis.
The wobble chairs will provide assistance for my students who have difficulties focusing and atten
ding during lessons and discussions. Despite the fact that students participate in physical activi
ties in P.E., Recess, and GoNoodle (dance videos) sessions in our classroom, students still have e
nergy to stand or wiggle from their seats during lessons. Due to these special needs that are
beyond the students' control, there is a lot of distraction and student learning is not really ach
ieved at its full potential. The lack of appropriate stimulation hinders them to focus and learn i
n class. Students with special needs will be able to sit on the wobble chairs during whole
group/small group lessons. This will enable their little active bodies to move while "sitting stil
l" without disrupting other students. As a result, all students will improve focus and increase st
udent attention in learning all content areas. In addition, the visual timer will help my students
to actually see the allotted time for activities. This will benefit especially ELL students and st
udents with special needs. Whenever we do independent classwork or work in our centers, the studen
ts can refer to it and self-monitor their progress in completing assignments. It will encourage th
em to use their time wisely and finish tasks on time. It will also help the students have a smooth
er transition from one activity to another. \r\nBy donating to this project, you will
significantly help students with special needs have an equal opportunity to learn with their
peers. Behavior issues will be greatly minimized and classroom management will be optimized. Help
me set all students for success! I am looking forward to seeing my students become active listener
s and engaged learners, and always happy to go to school!\r\nnannan
==================================================


---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-11-009ab9740f10> in <module>
      8 print(project_data['essay'].values[20000])
      9 print("="*50)
---> 10 print(project_data['essay'].values[99999])
     11 print("="*50)

IndexError: index 99999 is out of bounds for axis 0 with size 50000
```

In [12]:

```
# 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 [13]:

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

```
I teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school pr
ovides free breakfast for all students. I am a Special Education certified teacher and I teach Kin
dergarten in a general education setting with my class that consists 52% students with special nee
ds. The disabilities include Autism Spectrum Disorder, Speech Impaired, Language Impaired, Other H
ealth Impaired (ADHD), and Developmentally Delayed. I also have about 42% of my students who are E
nglish Language Learners.\r\n\r\n\"Self-motivated learners\" is a synonym of \"my students\".  The
y love to learn and they possess a positive outlook and attitude in school. Almost everyday, my st
udents would ask me, \"Ms. Perez, what are we going to learn today?\" I could not ask for a better
greeting from my students.This project will greatly impact my students' learning on a daily basis.
The wobble chairs will provide assistance for my students who have difficulties focusing and atten
ding during lessons and discussions. Despite the fact that students participate in physical activi
ties in P.E., Recess, and GoNoodle (dance videos) sessions in our classroom, students still have e
nergy to stand or wiggle from their seats during lessons. Due to these special needs that are
beyond the students' control, there is a lot of distraction and student learning is not really ach
```

beyond the students' control, there is a lot of distraction and student learning is not really achieved at its full potential. The lack of appropriate stimulation hinders them to focus and learn in class. Students with special needs will be able to sit on the wobble chairs during whole group/small group lessons. This will enable their little active bodies to move while "sitting still" without disrupting other students. As a result, all students will improve focus and increase student attention in learning all content areas. In addition, the visual timer will help my students to actually see the allotted time for activities. This will benefit especially ELL students and students with special needs. Whenever we do independent classwork or work in our centers, the students can refer to it and self-monitor their progress in completing assignments. It will encourage them to use their time wisely and finish tasks on time. It will also help the students have a smoother transition from one activity to another. \r\nBy donating to this project, you will significantly help students with special needs have an equal opportunity to learn with their peers. Behavior issues will be greatly minimized and classroom management will be optimized. Help me set all students for success! I am looking forward to seeing my students become active listeners and engaged learners, and always happy to go to school!\r\nnannan
==

In [14]:

```
# \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)
```

I teach at a Title 1 school, with 73% of my students who receive free/reduced lunch. Our school provides free breakfast for all students. I am a Special Education certified teacher and I teach Kindergarten in a general education setting with my class that consists 52% students with special needs. The disabilities include Autism Spectrum Disorder, Speech Impaired, Language Impaired, Other Health Impaired (ADHD), and Developmentally Delayed. I also have about 42% of my students who are English Language Learners. Self-motivated learners is a synonym of my students . They love to learn and they possess a positive outlook and attitude in school. Almost everyday, my students would ask me, Ms. Perez, what are we going to learn today? I could not ask for a better greeting from my students.This project will greatly impact my students' learning on a daily basis. The wobble chairs will provide assistance for my students who have difficulties focusing and attending during lessons and discussions. Despite the fact that students participate in physical activities in P.E., Recess, and GoNoodle (dance videos) sessions in our classroom, students still have energy to stand or wiggle from their seats during lessons. Due to these special needs that are beyond the students' control, there is a lot of distraction and student learning is not really achieved at its full potential. The lack of appropriate stimulation hinders them to focus and learn in class. Students with special needs will be able to sit on the wobble chairs during whole group/small group lessons. This will enable their little active bodies to move while "sitting still" without disrupting other students. As a result, all students will improve focus and increase student attention in learning all content areas. In addition, the visual timer will help my students to actually see the allotted time for activities. This will benefit especially ELL students and students with special needs. Whenever we do independent classwork or work in our centers, the students can refer to it and self-monitor their progress in completing assignments. It will encourage them to use their time wisely and finish tasks on time. It will also help the students have a smoother transition from one activity to another. By donating to this project, you will significantly help students with special needs have an equal opportunity to learn with their peers. Behavior issues will be greatly minimized and classroom management will be optimized. Help me set all students for success! I am looking forward to seeing my students become active listeners and engaged learners, and always happy to go to school! nannan

In [15]:

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

I teach at a Title 1 school with 73 of my students who receive free reduced lunch Our school provides free breakfast for all students I am a Special Education certified teacher and I teach Kindergarten in a general education setting with my class that consists 52 students with special needs The disabilities include Autism Spectrum Disorder Speech Impaired Language Impaired Other Health Impaired ADHD and Developmentally Delayed I also have about 42 of my students who are English Language Learners Self motivated learners is a synonym of my students They love to learn and they possess a positive outlook and attitude in school Almost everyday my students would ask me Ms Perez what are we going to learn today I could not ask for a better greeting from my students This project will greatly impact my students learning on a daily basis The wobble chairs will provide assistance for my students who have difficulties focusing and attending during lessons and discussions Despite the fact that students participate in physical activities in P E Recess and GoNoodle dance videos sessions in our classroom students still have energy to stand or wiggle from their seats during lessons Due to these special needs that are beyond the students control there is a lot of distraction and student learning is not really achieved at its full potential The lack of appropriate

stimulation hinders them to focus and learn in class Students with special needs will be able to sit on the wobble chairs during whole group small group lessons This will enable their little active bodies to move while sitting still without disrupting other students As a result all students will improve focus and increase student attention in learning all content areas In addition the visual timer will help my students to actually see the allotted time for activities T his will benefit especially ELL students and students with special needs Whenever we do independent classwork or work in our centers the students can refer to it and self monitor their p rogress in completing assignments It will encourage them to use their time wisely and finish tasks on time It will also help the students have a smoother transition from one activity to another By donating to this project you will significantly help students with special needs have an equal opportunity to learn with their peers Behavior issues will be greatly minimized and classroom management will be optimized Help me set all students for success I am looking forward to seeing m y students become active listeners and engaged learners and always happy to go to school nannan

In [16]:

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

In [17]:

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

```
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000 [00:
52<00:00, 955.78it/s]
```

In [18]:

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

Out[18]:

'teach title 1 school 73 students receive free reduced lunch school provides free breakfast students special education certified teacher teach kindergarten general education setting class consists 52 students special needs disabilities include autism spectrum disorder speech impaired language impaired health impaired adhd developmentally delayed also 42 students english language learners self motivated learners synonym students love learn possess positive outlook attitude school almost everyday students would ask ms perez going learn today could not ask better greeting students project greatly impact students learning daily basis wobble chairs provide assistance students difficulties focusing attending lessons discussions despite fact students participate physical activities p e recess gonoodle dance videos sessions classroom students still energy stand wiggle seats lessons due special needs beyond students control lot distraction student learning not really achieved full potential lack appropriate stimulation hinders focus learn class students special needs able sit wobble chairs whole group small group lessons enable little active bodies move sitting still without disrupting students result students improve focus increase student attention learning content areas addition visual timer help students actually see allotted time activities benefit especially ell students students special needs whenever independent classwork work centers students refer self monitor progress completing assignments encourage use time wisely finish tasks time also help students smoother transition one activity another donating project significantly help students special needs equal opportunity learn peers behavior issues greatly minimized classroom management optimized help set students success looking forward seeing students become active listeners engaged learners always happy go school nannan'

1.4 Preprocessing of `project_title`

In [19]:

```
# similarly you can preprocess the titles also
```

In [20]:

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

```
100%|██████████████████████████████████████████████████████████████████████| 50000/50000
[00:02<00:00, 21378.88it/s]
```

In [21]:

```
# Preprocess teacher_prefix
from tqdm import tqdm
preprocessed_teacher_prefix = []
# tqdm is for printing the status bar
for teacher_prefix in tqdm(project_data['teacher_prefix'].values):
    teacher_prefix = str(teacher_prefix)
    clean_teacher_prefix = decontracted(teacher_prefix)
    clean_teacher_prefix = clean_teacher_prefix.replace('\\r', ' ')
    clean_teacher_prefix = clean_teacher_prefix.replace('\\"', ' ')
    clean_teacher_prefix = clean_teacher_prefix.replace('\\n', ' ')
    clean_teacher_prefix = re.sub('[^A-Za-z0-9]+', ' ', clean_teacher_prefix)
    if clean_teacher_prefix in stopwords:
        continue
    preprocessed_teacher_prefix.append(clean_teacher_prefix.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████| 50000/50000
[00:01<00:00, 37121.49it/s]
```

In [22]:

```
# Preprocess project_grade_category
from tqdm import tqdm
preprocessed_project_grade_category = []
# tqdm is for printing the status bar
```

```
for project_grade_category in tqdm(project_data['project_grade_category'].values):
    project_grade_category = str(project_grade_category)
    clean_project_grade_category = decontracted(project_grade_category)
    clean_project_grade_category = clean_project_grade_category.replace('\\r', ' ')
    clean_project_grade_category = clean_project_grade_category.replace('\\"', ' ')
    clean_project_grade_category = clean_project_grade_category.replace('\\n', ' ')
    clean_project_grade_category = re.sub('[^A-Za-z0-9]+', ' ', clean_project_grade_category)
    if clean_project_grade_category in stopwords:
        continue
    preprocessed_project_grade_category.append(clean_project_grade_category.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████████| 50000/50000
[00:01<00:00, 35373.30it/s]
```

In [23]:

```
# Replace original columns with preprocessed column values
project_data['clean_essays'] = preprocessed_essays
project_data['clean_titles'] = preprocessed_title
project_data['teacher_prefix'] = preprocessed_teacher_prefix
# project_data['project_grade_category'] = preprocessed_project_grade_category
# Drop essays column
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [24]:

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')
```

1.5 Preparing data for models

In [25]:

```
project_data.columns
```

Out[25]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'clean_essays',
       'clean_titles'],
      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/

In [26]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
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)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (50000, 9)
```

In [27]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (50000, 30)
```

In [28]:

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

In [29]:

```
# we use count vectorizer to convert the values into one hot encoded features for state
countries_list = sorted(project_data["school_state"].value_counts().keys())
vectorizer = CountVectorizer(vocabulary=list(countries_list), lowercase=False, binary=True)
vectorizer.fit(project_data['school_state'].values)
print(vectorizer.get_feature_names())


school_state_one_hot = vectorizer.transform(project_data['school_state'].values)
print("Shape of matrix after one hot encodig ",school_state_one_hot.shape)
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'K
S', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV
', 'WY']
Shape of matrix after one hot encodig  (50000, 51)
```

In [30]:

```
# Project_Grade_Category - replacing hyphens, spaces with Underscores
project_data['project_grade_category'] = project_data['project_grade_category'].map({'Grades PreK-
2': 'Grades_PreK_2',
                                                                                     'Grades 6-8' :
rades_6_8',
                                                                                     'Grades 3-5' :
rades_3_5',
                                                                                     'Grades 9-12'
Grades_9_12'})
project_data['teacher_prefix'] = project_data['teacher_prefix'].map({'Mrs.': 'Mrs', 'Ms.': 'Ms', 'M
r.' : 'Mr',
                                                                     'Teacher': 'Teacher', 'Dr.' :
'Dr'})
```

In [31]:

```
# we use count vectorizer to convert the values into one hot encoded features for teacher_prefix
project_data["teacher_prefix"].fillna("Mrs", inplace=True)
teacher_prefix_list = sorted(project_data["teacher_prefix"].fillna("Mrs").value_counts().keys())
vectorizer = CountVectorizer(vocabulary=list(teacher_prefix_list), lowercase=False, binary=True)
vectorizer.fit(project_data['teacher_prefix'].fillna("Mrs").values)
print(vectorizer.get_feature_names())


teacher_prefix_one_hot = vectorizer.transform(project_data['teacher_prefix'].fillna("Mrs").values)
print("Shape of matrix after one hot encodig ",teacher_prefix_one_hot.shape)
```

```
['Mrs']
Shape of matrix after one hot encodig  (50000, 1)
```

In [32]:

```
# we use count vectorizer to convert the values into one hot encoded features for
project_grade_category
grade_list = sorted(project_data["project_grade_category"].value_counts().keys())
vectorizer = CountVectorizer(vocabulary=list(grade_list), lowercase=False, binary=True)
vectorizer.fit(project_data['project_grade_category'].values)
print(vectorizer.get_feature_names())


grade_one_hot = vectorizer.transform(project_data['project_grade_category'].values)
print("Shape of matrix after one hot encodig ",grade_one_hot.shape)
```

```
['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
Shape of matrix after one hot encodig  (50000, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [33]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
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  (50000, 12101)
```

In [34]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer_title = CountVectorizer(min_df=10)
text_bow_title = vectorizer.fit_transform(preprocessed_title)
print("Shape of matrix after one hot encodig ",text_bow_title.shape)
```

```
Shape of matrix after one hot encodig  (50000, 2039)
```

1.5.2.2 TFIDF vectorizer

In [35]:

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

```
Shape of matrix after one hot encodig  (50000, 12101)
```

Shape of matrix after one hot encodig (50000, 12101)

In [36]:

```
# TFIDF  Vectorizer for Preprocessed Title
vectorizer_title = TfidfVectorizer(min_df=10)
text_tfidf_title = vectorizer_title.fit_transform(preprocessed_title)
print("Shape of matrix after one hot encodig ",text_tfidf_title.shape)
```

```
Shape of matrix after one hot encodig  (50000, 2039)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [37]:

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

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

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

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

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

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

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

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


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/

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


'''
```

Out[37]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
```

```
\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n# ==========================\nOutput:\n    \nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
==========================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", 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 tha
t are present in both glove vectors and our coupus",       len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\n
print("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [38]:

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

In [39]:

```
# 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 tqdm(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]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000
[00:24<00:00, 2041.12it/s]
```

```
50000
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [40]:

```
# 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 [41]:

```
# 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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

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

```
100%|████████████████████████████████████████████████████████████████| 50000/50000 [02:
26<00:00, 341.76it/s]
```

```
50000
300
```

In [42]:

```
# Similarly you can vectorize for title also
```

In [43]:

```
tfidf_w2v_vectors_title = []; # 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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_title.append(vector)

print(len(tfidf_w2v_vectors_title))
print(len(tfidf_w2v_vectors_title[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 50000/50000
[00:02<00:00, 19392.66it/s]
```

```
50000
300
```

1.5.3 Vectorizing Numerical features

In [44]:

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

In [45]:

```
# 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 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.
price_standardized = price_scalar.transform(project_data['price'].values.reshape(-1, 1))
```

```
Mean : 299.3336762, Standard deviation : 378.20927190421384
```

In [46]:

```
price_standardized
```

Out[46]:

```
array([[ 0.48043858],
       [-0.74454461],
       [-0.52043588],
       ...,
       [ 0.08338326],
       [-0.08134035],
       [ 0.26352163]])
```

1.5.4 Merging all the above features

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

In [47]:

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
```

```
(50000, 9)
(50000, 30)
(50000, 12101)
(50000, 1)
```

In [48]:

```
# 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[48]:

```
(50000, 12141)
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**

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

2. **Hyper paramter tuning to find best K**
    - Find the best hyper parameter which results in the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
    - Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task
3. **Representation of results**
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
    - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
    - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points
4. **[Task-2]**
    - Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features
        - ```
          from sklearn.datasets import load_digits
          from sklearn.feature_selection import SelectKBest, chi2
          X, y = load_digits(return_X_y=True)
          X.shape
          X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
          X_new.shape
          ========
          output:
          (1797, 64)
          (1797, 20)
          ```
    - Repeat the steps 2 and 3 on the data matrix after feature selection
5. **Conclusion**
    - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

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

2. K Nearest Neighbor

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

In [49]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# 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 [50]:
```

```
y = project_data['project_is_approved'].values
X = project_data.drop(['project_is_approved'], axis=1)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [51]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# 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 [52]:

```
# School State
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)
```

In [53]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)
```

In [54]:

```
# project_grade_category
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)
```

In [55]:

```
# categories
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_category_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_category_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_category_ohe = vectorizer.transform(X_test['clean_categories'].values)
```

In [56]:

```
# sub categories
vectorizer = CountVectorizer()
```

```
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategory_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_subcategory_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcategory_ohe = vectorizer.transform(X_test['clean_subcategories'].values)
```

In [57]:

```
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))
```

In [58]:

```
# teacher previously posted projects
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_teach_prev_norm =
normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_cv_teach_prev_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].v
alues.reshape(-1,1))
X_test_teach_prev_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [59]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# 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 [60]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'].values)
```

Out[60]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=5000, min_df=10,
                ngram_range=(1, 4), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

In [61]:

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)
```

In [62]:

```
# Preprocessing project title
```

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_title'].values)
```

Out[62]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=5000, min_df=10,
                ngram_range=(1, 4), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

In [63]:

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_pj_title_bow = vectorizer.transform(X_train['project_title'].values)
X_cv_pj_title_bow = vectorizer.transform(X_cv['project_title'].values)
X_test_pj_title_bow = vectorizer.transform(X_test['project_title'].values)
```

In [64]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_resource_summary'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_resource_summary_bow = vectorizer.transform(X_train['project_resource_summary'].va
lues)
X_cv_project_resource_summary_bow = vectorizer.transform(X_cv['project_resource_summary'].values)
X_test_project_resource_summary_bow =
vectorizer.transform(X_test['project_resource_summary'].values)
```

2.4 Appling KNN on different kind of featurization as mentioned in the instructions

Apply KNN 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 instructions

In [65]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# 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 KNN brute force on BOW, SET 1

In [66]:

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

In [68]:

```
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow, X_train_state_ohe, X_train_teacher_ohe,
               X_train_grade_ohe, X_train_price_norm, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_teach_prev_norm,
               X_train_pj_title_bow)).tocsr()

X_cr = hstack((X_cv_essay_bow, X_cv_state_ohe, X_cv_teacher_ohe,
               X_cv_grade_ohe, X_cv_category_ohe, X_cv_subcategory_ohe,
```

```
                X_cv_price_norm, X_cv_teach_prev_norm, X_cv_pj_title_bow)).tocsr()

X_te = hstack((X_test_essay_bow, X_test_state_ohe, X_test_teacher_ohe,
                X_test_grade_ohe, X_test_category_ohe, X_test_subcategory_ohe,
                X_test_price_norm, X_test_teach_prev_norm,
                X_test_pj_title_bow)).tocsr()
```

In [69]:

```
# Since there was memory errors while trying with RandomizedSearchCV, trying the for loop approach
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [70]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))
```

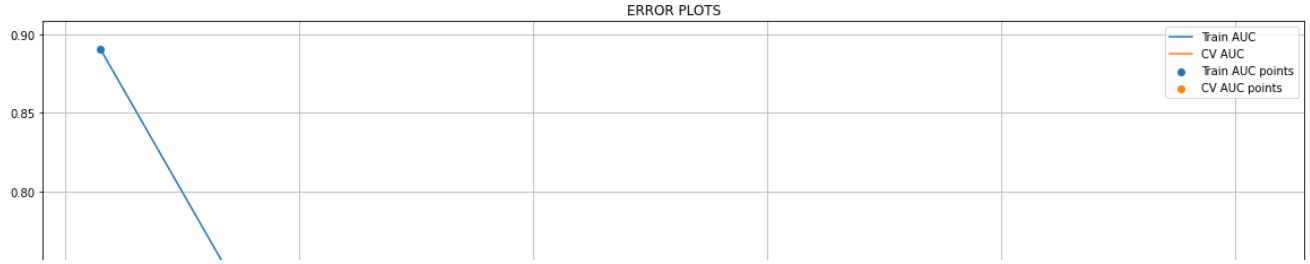
```
100%|██████████████████████████████████████████████████████████████████████████████████| 5/5 [08:
41<00:00, 104.21s/it]
```

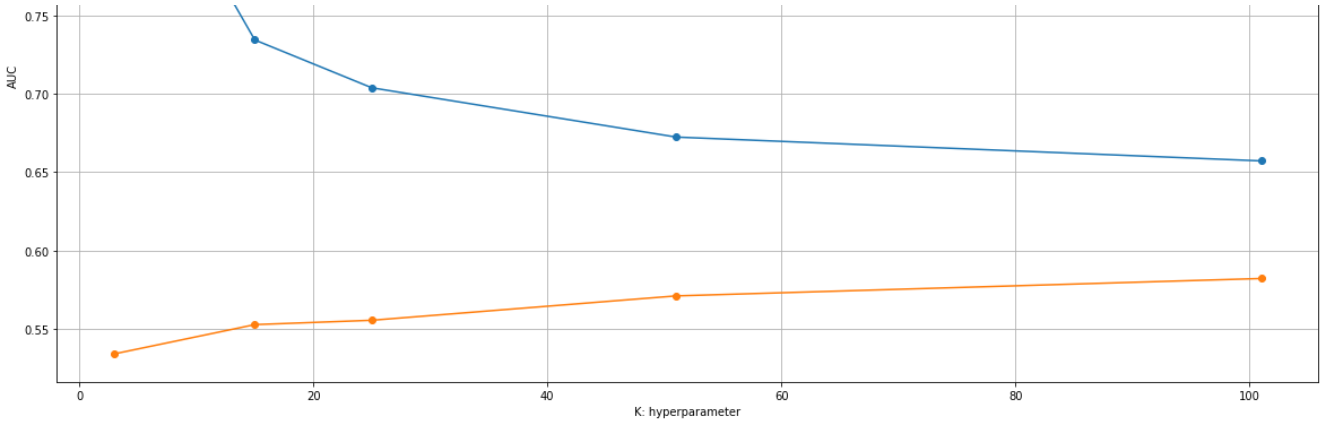
In [71]:

```
plt.figure(figsize=(20,10))
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```





In [72]:

```
# from the above error plot, we have max AUC for CV with least difference between Train and CV AUC
s at K=105
best_k = 105
```

Testing the performance on test data, plotting ROC Curves

In [73]:

```
from sklearn.metrics import roc_curve, auc

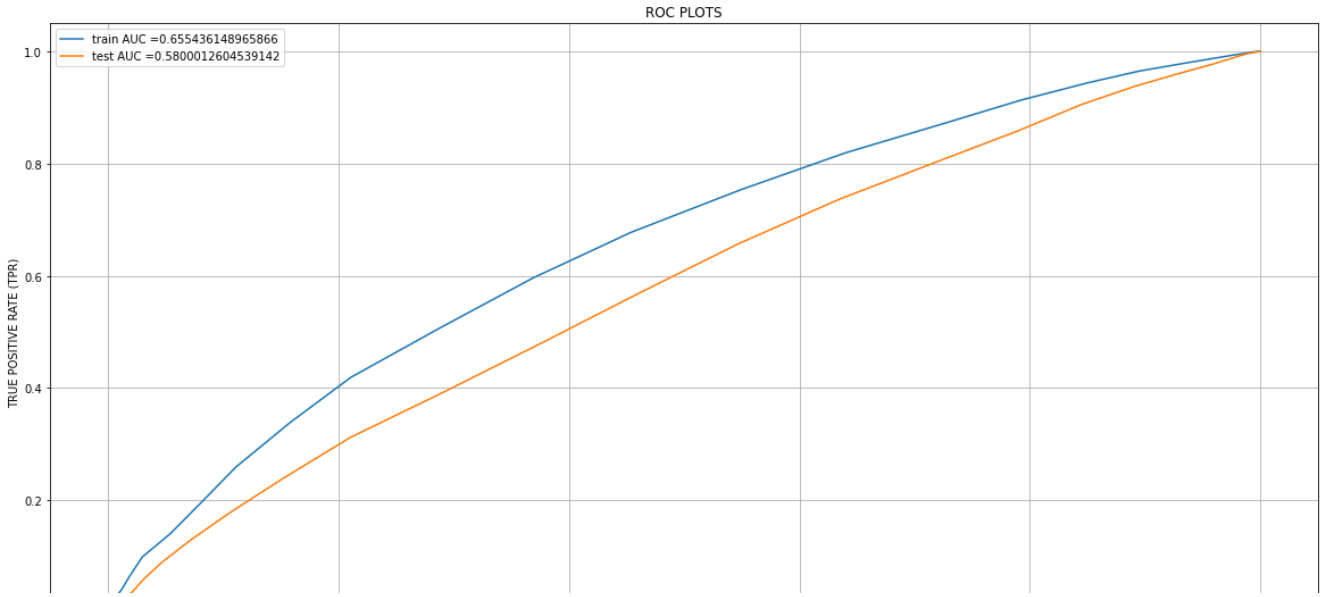

neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_tr, y_train)

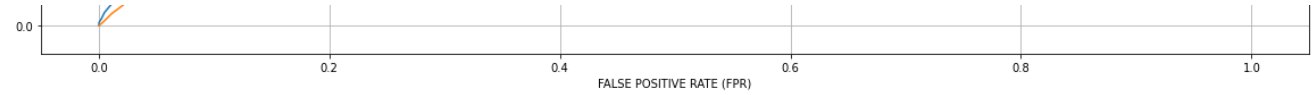
y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

In [74]:

```
plt.figure(figsize=(20,10))
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```





In [79]:

```
from sklearn.metrics import confusion_matrix
```

In [77]:

```
# Please compare all your models using Prettytable library
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr

import seaborn as sns
def predict(proba, threshould, fpr, tpr):

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

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

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

print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))

print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

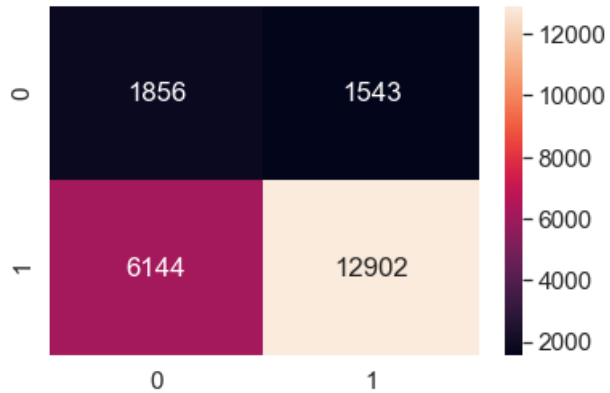
```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24788004640445532 for threshold 0.819
[[ 1856  1543]
 [ 6144 12902]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24833140906457088 for threshold 0.829
[[1377 1169]
 [6039 7915]]
```

In [78]:

```
ax = sns.heatmap(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)), annot=True, fmt="d")
```

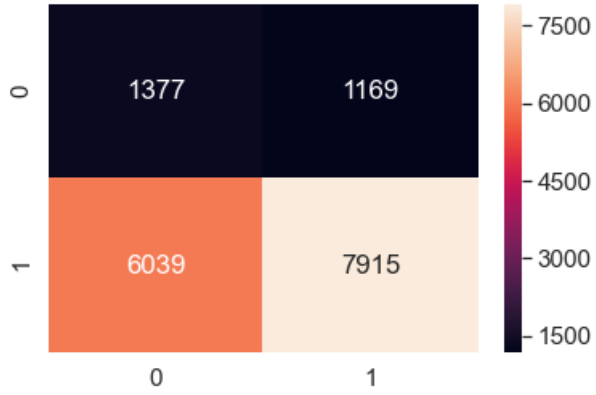
```
the maximum value of tpr*(1-fpr) 0.24788004640445532 for threshold 0.819
```



In [80]:

```
ax = sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)),
annot=True, fmt="d")
```

```
the maximum value of tpr*(1-fpr) 0.24833140906457088 for threshold 0.829
```



2.4.2 Applying KNN brute force on TFIDF, SET 2

In [81]:

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

In [82]:

```
# preprocessing TFIDF of Text Essays and Project Titles
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train["essay"].values)
X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("Shape of Datamatrix after TFIDF Vectorization")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
Shape of Datamatrix after TFIDF Vectorization
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
====================================================================================================
```

In [83]:

```
# Similarly you can vectorize for title also
vectorizer_titles = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer_titles.fit(X_train["project_title"])

X_train_pj_title_tfidf = vectorizer.transform(X_train['project_title'].values)
X_cv_pj_title_tfidf = vectorizer.transform(X_cv['project_title'].values)
X_test_pj_title_tfidf = vectorizer.transform(X_test['project_title'].values)

print("Shape of Datamatrix after TFIDF Vectorization")
print(X_train_pj_title_tfidf.shape, y_train.shape)
print(X_cv_pj_title_tfidf.shape, y_cv.shape)
print(X_test_pj_title_tfidf.shape, y_test.shape)
print("="*100)
```

```
Shape of Datamatrix after TFIDF Vectorization
```

```
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
==========================================================================================================
```

In [84]:

```
# Concatinating all the features for Set 2

X_tr = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe,
               X_train_grade_ohe, X_train_price_norm, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_teach_prev_norm,
               X_train_pj_title_tfidf)).tocsr()

X_cr = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe,
               X_cv_grade_ohe, X_cv_category_ohe, X_cv_subcategory_ohe,
               X_cv_price_norm, X_cv_teach_prev_norm, X_cv_pj_title_tfidf)).tocsr()

X_te = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe,
               X_test_grade_ohe, X_test_category_ohe, X_test_subcategory_ohe,
               X_test_price_norm, X_test_teach_prev_norm,
               X_test_pj_title_tfidf)).tocsr()
```

In [85]:

```
train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

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

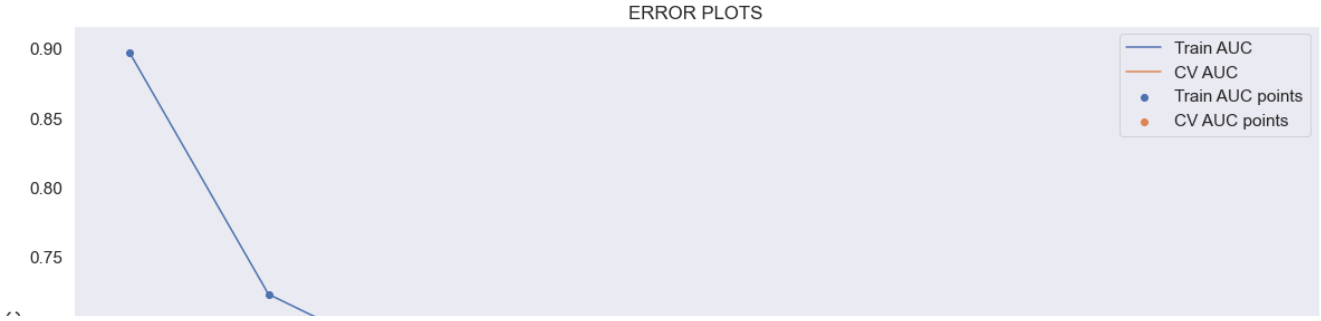
```
100%|██████████████████████████████████████████████████████████████████████████████████████████| 5/5 [08:
45<00:00, 105.16s/it]
```

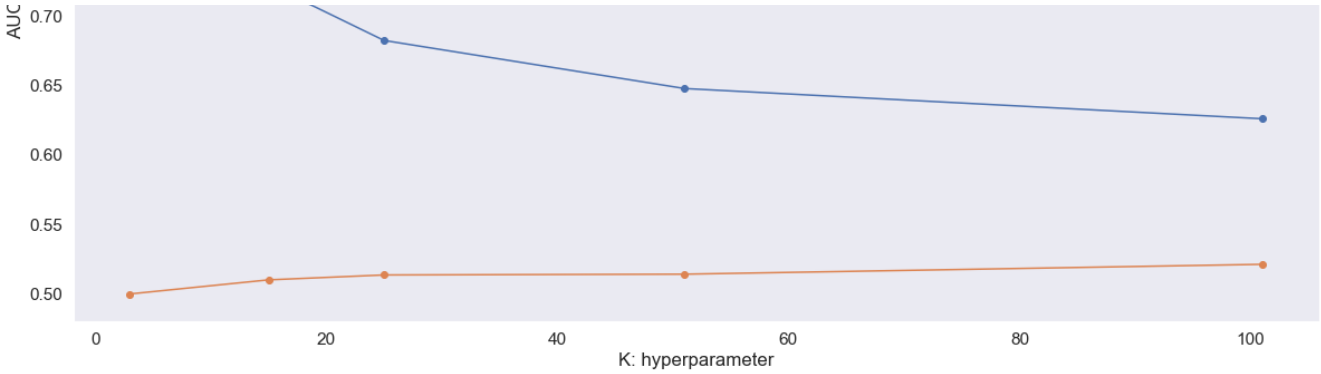
In [86]:

```
plt.figure(figsize=(20,10))
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```





In [87]:

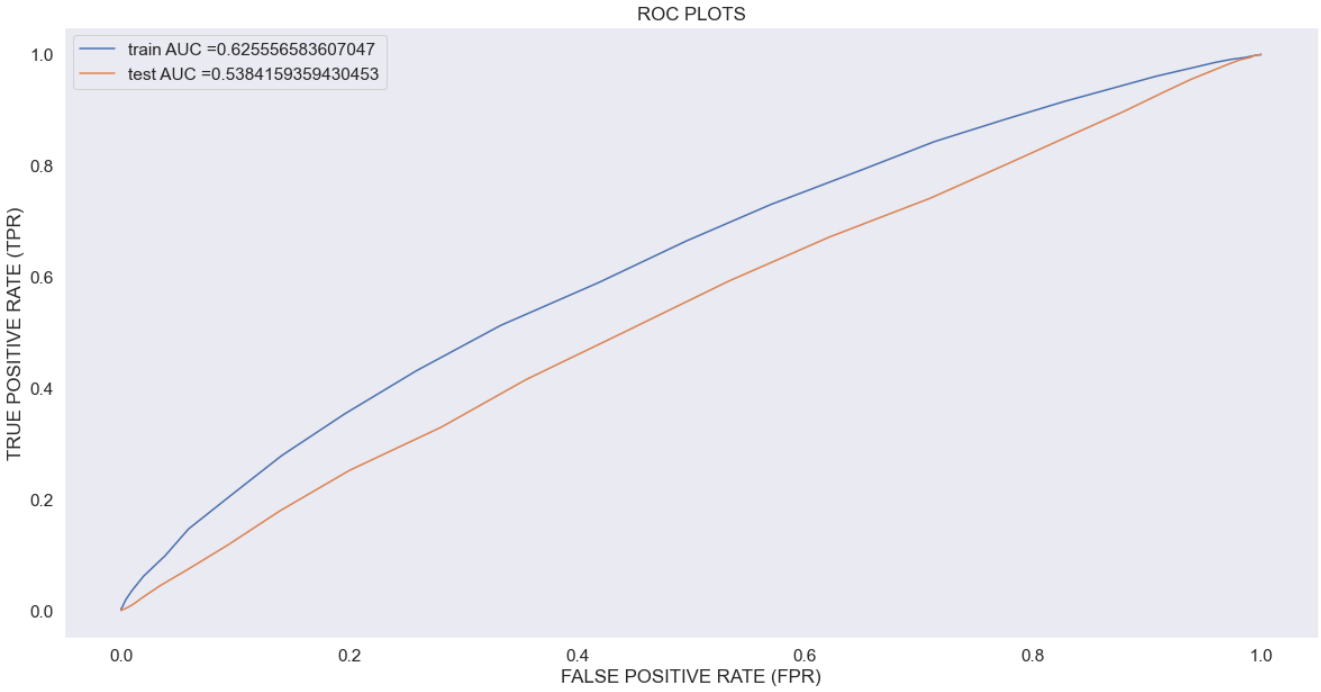
```
best_k = 105
# Testing the performance of the model on test data, plotting ROC curves
neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_tr, y_train)

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

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

In [88]:

```
plt.figure(figsize=(20,10))
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```



In [89]:

```
# Please compare all your models using Prettytable library
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

```
    t = thresholds[np.argmax(tpr*(1-tpr))]

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

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

print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

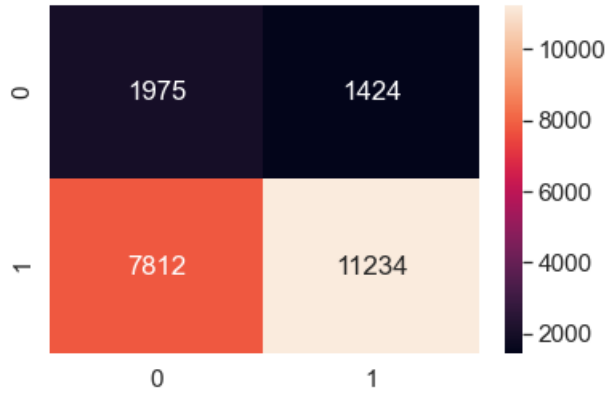
```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.34272562329316086 for threshold 0.857
[[ 1975  1424]
 [ 7812 11234]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2797508500886258 for threshold 0.867
[[1419 1127]
 [6950 7004]]
```

In [90]:

```
ax = sns.heatmap(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)), annot=True, fmt="d")
```

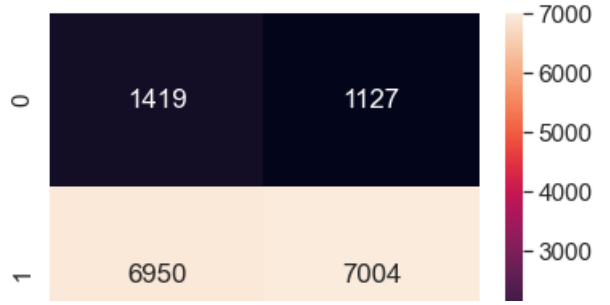
```
the maximum value of tpr*(1-fpr) 0.34272562329316086 for threshold 0.857
```



In [91]:

```
ax = sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)), annot=True, fmt="d")
```

```
the maximum value of tpr*(1-fpr) 0.2797508500886258 for threshold 0.867
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [92]:

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

In [93]:

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

In [94]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_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_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
print(len(avg_w2v_vectors_train[0]))
print(avg_w2v_vectors_train[0])
```

```
100%|██████████████████████████████████████████████████████████████████████████| 22445/22445
[00:13<00:00, 1675.24it/s]
```

```
22445
300
[ 1.53534579e-04 -4.28546125e-02 -5.69575370e-02 -1.90171575e-01
  2.21217811e-02 -5.69257047e-02 -3.47310441e+00  1.57879635e-01
 -7.10680808e-03 -1.90174249e-01  1.76609185e-01  2.70418242e-03
 -2.34331603e-02 -1.07688647e-01 -3.78535144e-02 -1.21082094e-01
 -7.51460815e-02 -1.18151207e-01  7.78165003e-02  7.65125532e-02
  1.11175081e-02 -1.45427486e-02 -3.36557390e-02  4.04743727e-02
 -2.58198525e-02 -3.84598296e-02  7.12255650e-02 -1.76111095e-01
 -7.85728283e-02 -7.91298584e-02 -2.68784503e-01 -1.21258788e-02
  1.80446492e-02  3.55476337e-02 -1.15931708e-01 -4.87077384e-02
 -1.47452663e-02 -7.29523047e-02  3.52324427e-02 -6.15930460e-02
 -3.75640343e-02  1.18556872e-01 -7.47008788e-03 -1.73067356e-01
 -8.08489867e-02 -4.56465256e-02  9.89089094e-02 -1.27743258e-01
 -1.05556055e-01  3.48995032e-02  7.83102551e-02  8.05355197e-02
 -3.20598283e-02  2.34304421e-02  6.57686458e-02 -7.14034576e-02
  1.15134984e-01 -1.37352670e-02 -7.67558240e-02  6.46511937e-02
 -5.28622753e-02 -1.97021852e-02  5.56774293e-02 -4.57208946e-02
 -4.01989613e-02  1.17601828e-01  4.00959671e-02  2.64030114e-02
  2.05724391e-01 -1.37607132e-01 -1.90012614e-01  1.70372104e-02
 -1.26872239e-02 -8.34049932e-02  1.33659182e-03 -2.18107374e-01
  1.07877044e-01  1.57439293e-03  3.67468684e-02 -5.02546485e-02
  6.81130714e-02 -5.12967395e-01 -4.98652515e-02 -1.24174473e-01
 -1.04657810e-02  3.78434989e-02  4.17072710e-02 -1.07148173e-01
  1.24952761e-01 -6.08942356e-02  1.90634592e-02 -7.88555044e-02
 -5.06932101e-02  1.07770040e-01  1.93508721e-03 -1.89314488e-01
 -2.40237253e+00 -2.77810164e-02  1.40348795e-01  8.89986404e-02
 -5.38669006e-02  1.06049688e-01  2.37412090e-01  2.18132327e-02
 -1.93445926e-02  4.41950515e-03  7.49787495e-02 -1.70705933e-01
 -4.39817549e-02 -1.13795283e-02 -6.25155761e-02  4.98209091e-02
  4.03724000e-02  2.02137033e-01 -4.39022825e-02  3.92053852e-02
```

```
-6.35996620e-02 -3.73826111e-02  1.48755971e-01  6.36142145e-02
-9.33294424e-02  5.44825205e-02  3.06763735e-02 -1.87649491e-01
-1.10784625e-02  2.56970303e-02  5.26676791e-02 -6.87956327e-02
 4.10687635e-02  1.08399060e-01  1.65370540e-02  7.23889451e-02
-8.41982098e-02 -1.00721473e-01  7.94010006e-02  1.68767408e-02
 8.02448253e-02 -1.23237994e-02  2.23121358e-01  2.70480821e-01
 9.52972852e-02  4.15879067e-02 -3.27717845e-04 -2.61557138e-03
-2.81319420e-02 -5.33013567e-02  3.47893411e-02 -8.36868020e-02
 2.39709692e-01  9.94445913e-02  3.54321960e-03 -4.48137178e-02
 4.94112495e-02 -3.92332436e-02 -1.30051303e-02 -1.13531999e-01
 7.31014980e-02  1.46592636e-02 -1.52759398e-01 -5.63842195e-02
 9.66035825e-02 -4.51066098e-02 -2.48268518e-02 -4.00953158e-02
-3.93487874e-02  5.14858566e-02 -5.21047846e-03  8.01856562e-02
 1.11063395e-01 -8.83023660e-02 -3.43880034e-03  2.32758811e-03
-9.70687701e-02 -1.10851552e-01 -6.71213582e-02  2.39219430e-01
-7.28277003e-02 -1.90773744e-02 -4.62151909e-02 -3.17629664e-02
 1.37409854e-01  1.24575429e-01 -9.79829630e-04 -8.49259322e-02
-4.61327912e-03 -1.27906369e-01 -1.07847911e-02  1.29141215e-02
 1.20942783e-01 -1.04975952e-02 -3.69722997e-03 -6.71144609e-02
-7.02607532e-03 -8.56488953e-03  6.58510269e-03 -8.33913037e-02
 8.36729582e-02  8.42892473e-02  9.12723754e-02 -5.46740107e-02
 1.23134571e-01 -1.06669672e-01  1.16934007e-03 -2.89965094e-03
-1.76265229e-02  1.43220097e-01  1.11145997e-02 -6.06593700e-02
 1.53791924e-01 -4.71841688e-02  4.99331182e-02 -1.68420725e-02
-5.43978253e-02 -1.11797382e-01 -4.46126522e-02 -3.75282099e-02
-9.08653707e-02 -4.10927630e-02 -3.85826774e-02  2.46847667e-02
-1.12379702e-01 -5.11535152e-02 -3.79937428e-02 -6.99560544e-03
-2.59677148e+00  9.24904441e-02 -3.93185818e-02 -8.51443232e-03
 1.85125751e-02 -1.00166873e-01  5.44828256e-02  6.18036606e-02
 7.15435993e-03 -8.24297047e-02 -9.08189628e-02  1.32181658e-01
-1.69753781e-02 -1.90535556e-03 -9.94065822e-02 -7.16488707e-03
-1.38101500e-01  1.39526846e-02 -1.92115428e-01  1.06924883e-01
-2.70110438e-02 -4.88171017e-02 -5.19558144e-02 -5.48112933e-02
-1.18849882e-01  8.57037212e-02 -4.53071323e-02 -1.76117246e-02
-1.58496818e-02 -5.27358034e-02  5.17805405e-02 -2.39472294e-02
 6.62267354e-02 -8.30444811e-02  4.58463522e-02  7.01749158e-05
 7.68730710e-02 -4.29169579e-02 -3.81456108e-02 -3.20112720e-02
 7.50570192e-02 -8.40895448e-02 -2.45876025e-01 -7.84814626e-02
 1.00132464e-01  2.47244428e-02 -4.66187677e-02 -8.24664865e-02
-1.59767046e-01  9.81480090e-02  4.49604502e-02  7.96902808e-02
 5.18625825e-02  4.96708556e-02 -5.81403320e-02  2.43870397e-03
 3.60295625e-01 -1.17365805e-02  3.47605481e-02  1.57531464e-01
 9.35545387e-03  1.27177620e-01  2.35508731e-02  9.24199562e-02
-1.22733727e-02 -5.86773596e-02  2.08865892e-02 -1.56236938e-01
-7.44234037e-02 -2.71533722e-02 -4.65981886e-03  2.38943420e-02
 1.29474579e-02 -6.40413877e-02  1.05971788e-01 -3.04305455e-03]
```

In [95]:

```
avg_w2v_vectors_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_vectors_cv.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 11055/11055
[00:06<00:00, 1673.90it/s]
```

In [96]:

```
avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['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_vectors_test.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████████████| 16500/16500
[00:10<00:00, 1565.81it/s]
```

In [97]:

```
# avg w2v for project_titles
avg_w2v_vectors_pj_title_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['project_title'].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_vectors_pj_title_train.append(vector)

print(len(avg_w2v_vectors_pj_title_train))
print(len(avg_w2v_vectors_pj_title_train[0]))
print(avg_w2v_vectors_pj_title_train[0])
```

```
100%|████████████████████████████████████████████████████████████████████████| 22445/22445
[00:00<00:00, 80304.93it/s]
```

```
22445
300
[-2.0838e-01 -1.4932e-01 -1.7528e-02 -2.8432e-02 -6.0104e-02 -2.6460e-01
 -4.1445e+00  6.2932e-01  3.3672e-01 -4.3395e-01  3.9899e-01 -1.9573e-01
  1.3977e-01 -2.1519e-02  3.7823e-01 -5.5250e-01 -1.1230e-01 -8.1443e-03
  2.9059e-01  6.6817e-02  1.0465e-01 -8.6943e-02 -4.8983e-02 -2.6757e-01
 -4.7038e-01  2.7469e-01  6.9245e-02 -2.7967e-02 -1.9719e-01  1.6749e-02
 -2.9681e-01  1.7838e-01  5.8374e-02 -2.4806e-01  8.5846e-02  3.5043e-01
  4.9157e-02 -1.6431e-01  5.0012e-01 -1.8053e-01  3.1422e-01  1.0671e-01
  3.1852e-02  7.4278e-02  2.7956e-01  8.0317e-02  5.4780e-02 -3.0349e-01
 -4.3215e-01  3.2417e-01  4.0856e-01  3.6192e-01  1.3445e-01 -1.2933e-01
  1.1331e-01 -1.5755e-01  3.5755e-01  3.0463e-01 -9.8488e-02  1.2032e-02
  4.5581e-01  3.7101e-01  1.4270e-01 -4.3329e-01 -1.0869e-01  4.9849e-01
  5.4455e-01  4.4352e-01  3.1804e-01  2.2171e-02 -4.1186e-01 -2.5428e-02
  2.1062e-01 -3.5830e-01  2.2028e-01 -5.5391e-01 -3.5364e-02 -5.3998e-02
  3.2172e-01 -5.1928e-01 -2.7427e-01 -4.5214e-01 -3.2900e-01 -4.8519e-01
  5.2966e-01  4.1434e-03 -1.7180e-01 -1.8748e-01 -2.4365e-01 -6.0786e-02
  5.0733e-02 -2.1335e-01  2.7627e-01  4.2745e-01  1.1461e-02 -2.9794e-01
 -3.2881e+00 -3.9842e-01  1.6796e-01 -1.2894e-01  2.0005e-03  4.5613e-01
  1.5215e-01  1.5364e-01 -2.1281e-01 -2.5339e-01  2.8955e-01 -5.7817e-01
 -6.0740e-01  1.0301e-01  2.8324e-01  2.1506e-01 -4.2325e-02  4.0479e-01
 -2.0579e-01 -1.1674e-02  2.6092e-01 -1.5402e-01  5.7961e-02 -5.8576e-02
 -4.1974e-01  5.2015e-01  1.5074e-01 -8.8039e-02 -1.4446e-01 -1.7074e-01
  8.3752e-02 -2.5708e-01  1.6362e-01  1.4795e-01 -5.9821e-02  3.4473e-02
 -1.4534e-01 -1.7965e-01  7.6303e-02  3.3354e-01 -1.4434e-01  1.7618e-01
  4.5345e-01  1.5262e-01 -7.5100e-02  2.7592e-01  8.1456e-02  3.0738e-01
 -7.2327e-02  1.0706e-01 -3.5581e-01 -2.6690e-02  6.1236e-01  7.0829e-01
 -2.8945e-01 -2.4637e-02  1.1890e-02 -9.1899e-02 -2.7272e-01 -1.0157e-01
  4.4713e-01  9.2418e-02 -1.0711e-01 -1.5552e-02  1.2822e-01  2.2256e-01
 -6.9059e-02  2.9927e-01 -1.0913e-01  1.6180e-01  1.4796e-01  1.1360e-01
  2.6634e-01  1.0832e-02  7.1946e-02  1.6973e-01 -2.2769e-01  3.2200e-01
 -8.3748e-02  6.5269e-01  6.8244e-02 -3.2687e-01  3.1782e-01  1.7035e-01
  7.9803e-01 -1.9194e-01 -1.6485e-01 -3.2437e-01  7.9105e-02 -3.5672e-01
 -2.6786e-01 -2.4786e-01  7.0512e-01 -1.1909e-01  1.6256e-01 -4.3259e-01
 -5.0078e-02  5.0232e-02 -1.1450e-01 -4.1885e-02  4.7866e-01  1.2767e-02
  1.9642e-01  2.6196e-01 -2.9425e-01  8.9615e-02 -1.7736e-01 -2.2448e-01
  2.2624e-01  1.6749e-01  5.5770e-02  1.4399e-01  2.1580e-01  3.3819e-01
  2.3459e-01  1.5826e-01 -2.8560e-01  2.4199e-01  1.1018e-01  3.8164e-01
 -2.9840e-01 -2.0169e-01  2.6950e-01  1.1186e-01 -2.1006e-01 -4.2070e-02
  1.6507e-02 -2.2866e-01 -3.3882e+00  2.9204e-01 -8.8358e-02 -1.4966e-02
 -2.5225e-01 -1.1503e-01  3.6337e-02 -1.4817e-01  4.6220e-02 -7.3466e-02
 -1.3866e-01  2.3612e-01  3.3882e-02  2.9495e-01 -6.1234e-01  2.0289e-01
 -4.2091e-01  3.7767e-01  3.6260e-02  2.1708e-01  1.2561e-01 -2.1682e-01
 -3.7997e-03 -1.7791e-01 -2.6431e-01  3.1678e-01 -5.1229e-02  4.9269e-02
 -1.2622e-01 -1.0117e-01  1.7246e-02 -2.1950e-02 -1.9820e-01  3.7250e-02
 -1.6791e-01 -5.5459e-02  5.7670e-01  5.9123e-02  2.2931e-01  6.4201e-02
```

```
  2.7424e-01 -3.7129e-01 -9.1375e-02 -7.1342e-02 -3.7218e-02 -1.2668e-02
 -1.7976e-02 -4.2622e-01 -1.0095e-01  4.4992e-02 -9.0225e-02  2.2915e-01
  1.8610e-01  3.6366e-01 -2.0676e-01 -3.3037e-01  4.7302e-01  2.3380e-01
  7.9306e-02  2.1083e-01  2.1013e-01  1.5275e-01  8.0873e-02 -3.3013e-01
 -1.7181e-01 -7.0170e-02 -4.1244e-02 -4.6182e-01  2.7903e-02  5.4657e-01
 -2.5894e-01  3.9515e-01  2.6144e-01 -5.4066e-01  2.1199e-01 -9.4357e-03]
```

In [98]:

```
avg_w2v_vectors_pj_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_title'].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_vectors_pj_title_cv.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 72772.52it/s]
```

In [99]:

```
avg_w2v_vectors_pj_title_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['project_title'].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_vectors_pj_title_test.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████| 16500/16500
[00:00<00:00, 70552.87it/s]
```

In [100]:

```
X_tr = hstack((avg_w2v_vectors_train, X_train_state_ohe, X_train_teacher_ohe,
               X_train_grade_ohe, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_price_norm,
               X_train_teach_prev_norm, avg_w2v_vectors_pj_title_train)).tocsr()

X_cr = hstack((avg_w2v_vectors_cv, X_cv_state_ohe, X_cv_teacher_ohe,
               X_cv_grade_ohe,X_cv_category_ohe,
               X_cv_subcategory_ohe, X_cv_price_norm,
              X_cv_teach_prev_norm, avg_w2v_vectors_pj_title_cv)).tocsr()

X_te = hstack((avg_w2v_vectors_test, X_test_state_ohe, X_test_teacher_ohe,
               X_test_grade_ohe, X_test_category_ohe,
               X_test_subcategory_ohe, X_test_price_norm,
              X_test_teach_prev_norm, avg_w2v_vectors_pj_title_test)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 697) (22445,)
(11055, 697) (11055,)
(16500, 697) (16500,)
====================================================================================================
```

In [101]:

```
train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

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

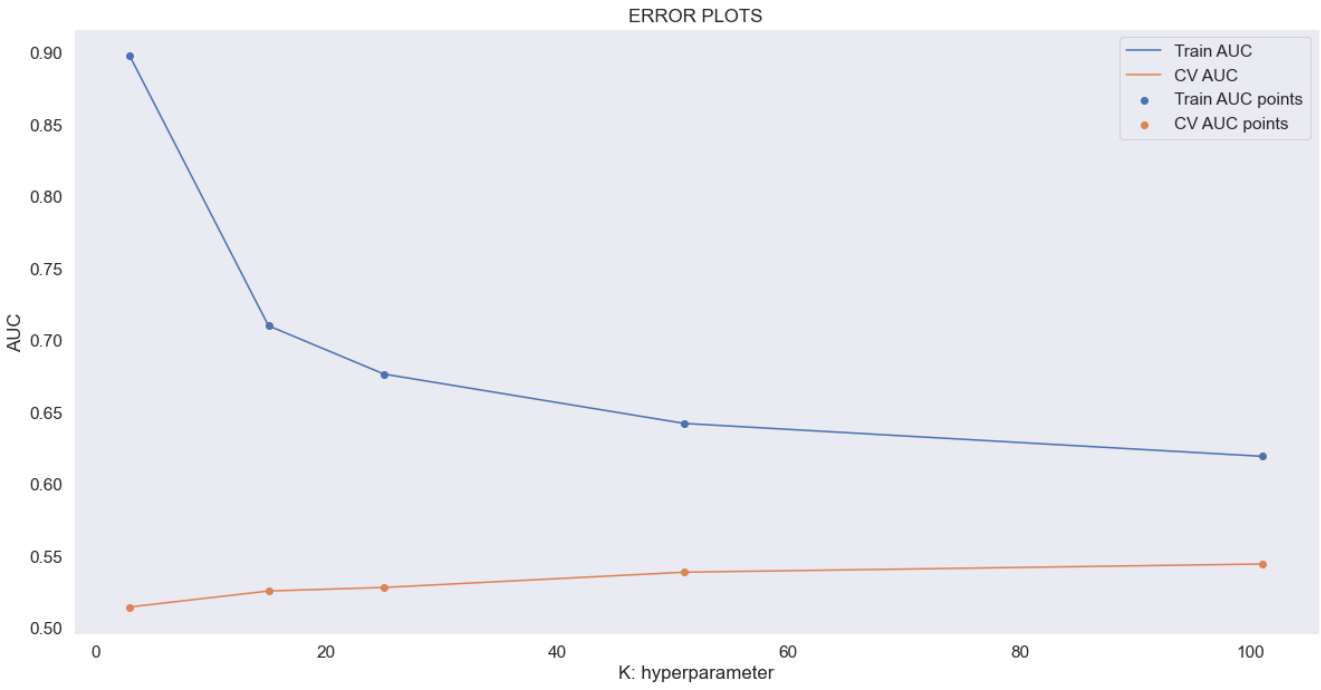
```
100%|██████████████████████████████████████████████████████████████████████████████████████████| 5/5 [38:
15<00:00, 459.08s/it]
```

In [102]:

```
plt.figure(figsize=(20,10))
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [103]:

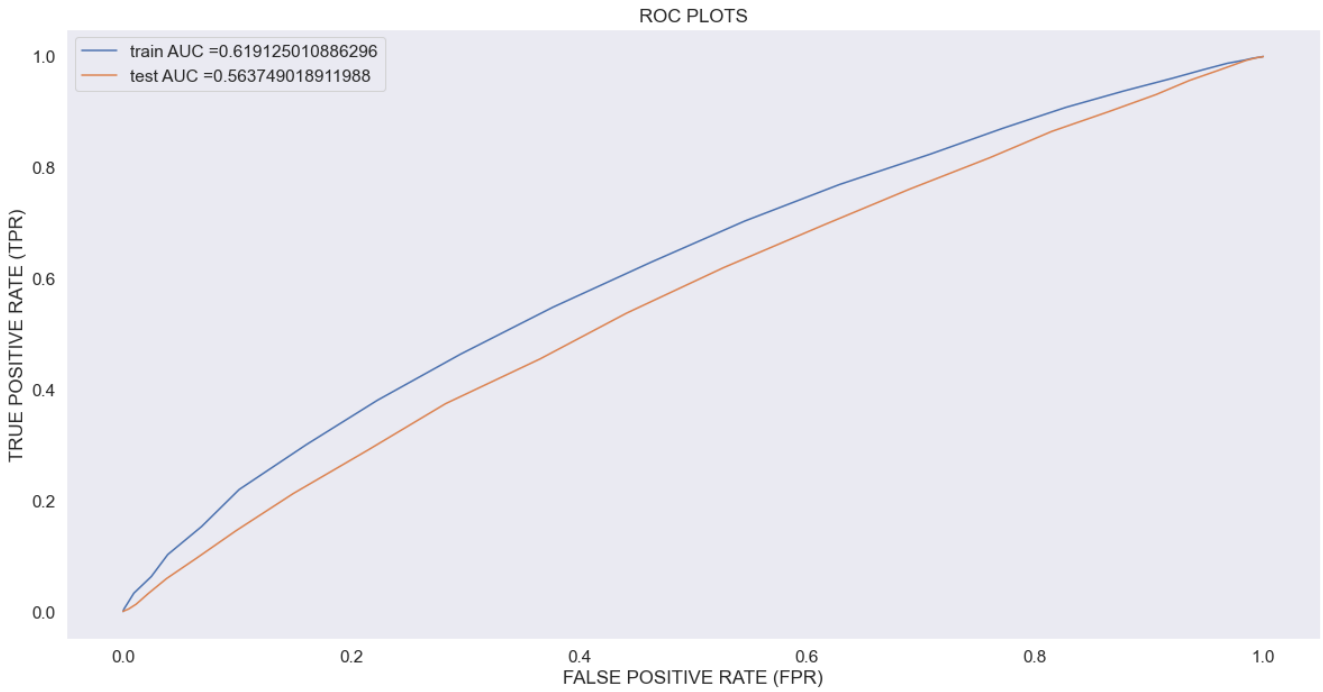
```
best_k = 101
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
```

```
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.figure(figsize=(20,10))
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```



In [104]:

```
# Please compare all your models using Prettytable library
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

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

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

print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.341530826236735 for threshold 0.851
[[ 1025  1574]
```

```
[[ 1825  1574]
 [ 7060 11986]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3001551726292686 for threshold 0.871
[[1615  931]
 [7605 6349]]
```

In [105]:

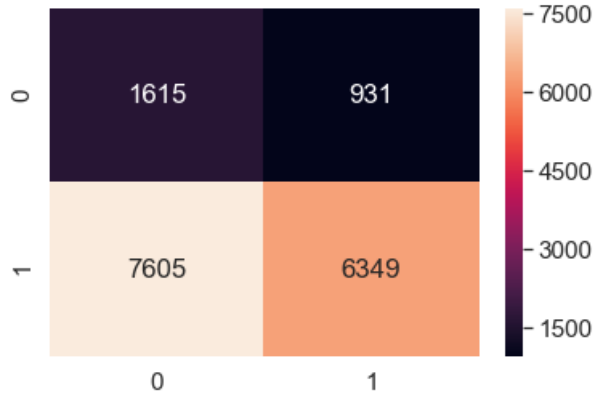
```
ax = sns.heatmap(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)), annot=True, fmt="d")
```

```
the maximum value of tpr*(1-fpr) 0.341530826236735 for threshold 0.851
```

In [106]:

```
ax = sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)), annot=True, fmt="d")
```

```
the maximum value of tpr*(1-fpr) 0.3001551726292686 for threshold 0.871
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [107]:

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

In [108]:

```
# preprocessing project_title and essay with TFIDF W2V Vectorization
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [109]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 22445/22445 [02:
25<00:00, 154.03it/s]
```

```
22445
300
```

In [110]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_cv.append(vector)

print(len(tfidf_w2v_vectors_cv))
print(len(tfidf_w2v_vectors_cv[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 11055/11055 [01:
11<00:00, 154.92it/s]
```

```
11055
300
```

In [111]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
```

```
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

print(len(tfidf_w2v_vectors_test))
print(len(tfidf_w2v_vectors_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 16500/16500 [01:
48<00:00, 152.55it/s]
```

```
16500
300
```

In [112]:

```
# preprocessing for Project_title with TFIDF Vectorization
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['project_title'])
# 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 [113]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_pj_title_train = []; # the avg-w2v for each sentence/review is stored in this li
st
for sentence in tqdm(X_train['project_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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_pj_title_train.append(vector)

print(len(tfidf_w2v_vectors_pj_title_train))
print(len(tfidf_w2v_vectors_pj_title_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 22445/22445
[00:00<00:00, 60753.00it/s]
```

```
22445
300
```

In [114]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_pj_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['project_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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
```

```
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_pj_title_cv.append(vector)

print(len(tfidf_w2v_vectors_pj_title_cv))
print(len(tfidf_w2v_vectors_pj_title_cv[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 11055/11055
[00:00<00:00, 51688.31it/s]
```

```
11055
300
```

In [115]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_pj_title_test = []; # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_test['project_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((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_pj_title_test.append(vector)

print(len(tfidf_w2v_vectors_pj_title_test))
print(len(tfidf_w2v_vectors_pj_title_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 16500/16500
[00:00<00:00, 56859.86it/s]
```

```
16500
300
```

In [116]:

```
# Concatinating all the features
X_tr = hstack((tfidf_w2v_vectors_train, X_train_state_ohe, X_train_teacher_ohe,
               X_train_grade_ohe, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_price_norm,
               X_train_teach_prev_norm, tfidf_w2v_vectors_pj_title_train)).tocsr()

X_cr = hstack((tfidf_w2v_vectors_cv, X_cv_state_ohe, X_cv_teacher_ohe,
               X_cv_grade_ohe,X_cv_category_ohe,
               X_cv_subcategory_ohe, X_cv_price_norm,
              X_cv_teach_prev_norm, tfidf_w2v_vectors_pj_title_cv)).tocsr()

X_te = hstack((tfidf_w2v_vectors_test, X_test_state_ohe, X_test_teacher_ohe,
               X_test_grade_ohe, X_test_category_ohe,
               X_test_subcategory_ohe, X_test_price_norm,
              X_test_teach_prev_norm, tfidf_w2v_vectors_pj_title_test)).tocsr()

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
```

```
(22445, 697) (22445,)
(11055, 697) (11055,)
(16500, 697) (16500,)
==========================================================================================================
```

In [117]:

```
# Hyper parameter Tuning
train_auc = []
cv_auc = []
K = [3, 15, 25, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr, y_train)

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, X_cr)

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

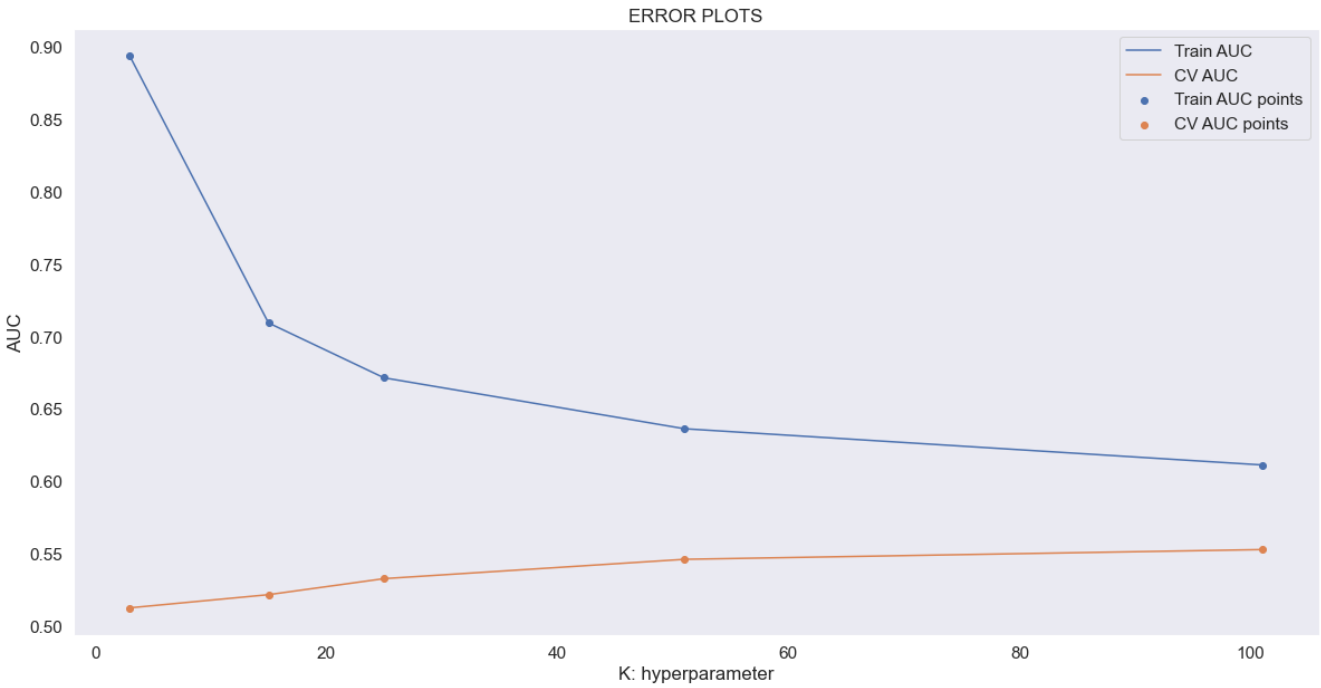
```
100%|██████████████████████████████████████████████████████████████████████████████████████████| 5/5 [37:
22<00:00, 448.54s/it]
```

In [118]:

```
plt.figure(figsize=(20,10))
plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



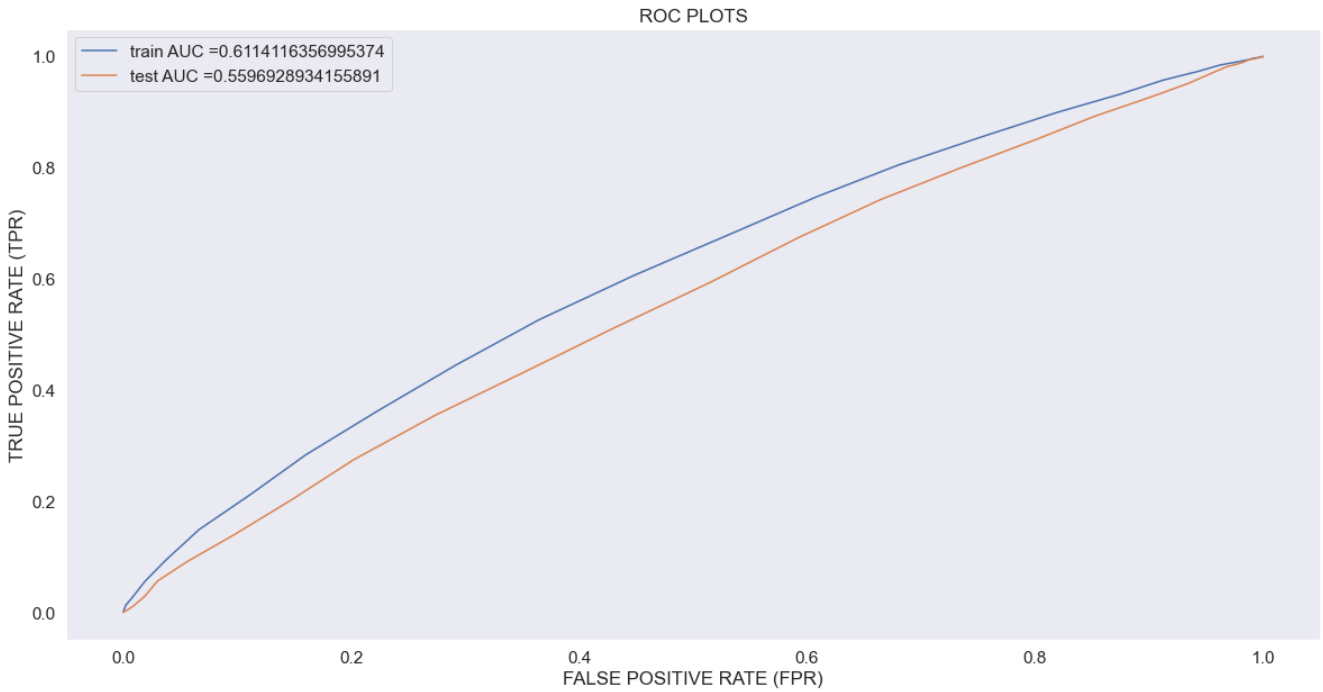
In [119]:

```
best_k = 101
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k, n_jobs=-1)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr)
y_test_pred = batch_predict(neigh, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
plt.figure(figsize=(20,10))
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FALSE POSITIVE RATE (FPR)")
plt.ylabel("TRUE POSITIVE RATE (TPR)")
plt.title("ROC PLOTS")
plt.grid()
plt.show()
```



In [120]:

```
# Please compare all your models using Prettytable library
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

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

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

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

```
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

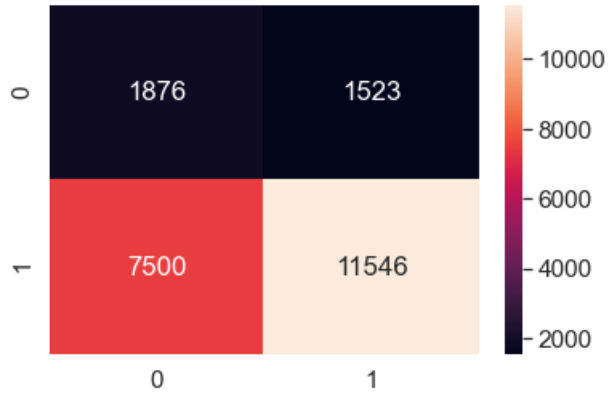
```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.3345872925235715 for threshold 0.851
[[ 1876  1523]
 [ 7500 11546]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2913723027327699 for threshold 0.871
[[1650  896]
 [7921 6033]]
```

In [121]:

```
ax = sns.heatmap(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)), annot=True, fmt="d")
```

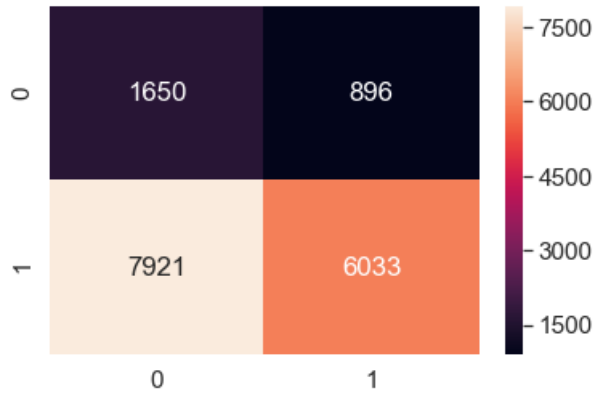
```
the maximum value of tpr*(1-fpr) 0.3345872925235715 for threshold 0.851
```



In [122]:

```
ax = sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)), annot=True, fmt="d")
```

```
the maximum value of tpr*(1-fpr) 0.2913723027327699 for threshold 0.871
```



2.5 Feature selection with `SelectKBest`

In [123]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# 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.feature_selection import SelectKBest, chi2
```

In [126]:

```
X_tr = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_ohe,
               X_train_grade_ohe, X_train_price_norm, X_train_category_ohe,
               X_train_subcategory_ohe, X_train_teach_prev_norm,
               X_train_pj_title_tfidf)).tocsr()

X_cr = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe,
               X_cv_grade_ohe, X_cv_category_ohe, X_cv_subcategory_ohe,
               X_cv_price_norm, X_cv_teach_prev_norm, X_cv_pj_title_tfidf)).tocsr()

X_te = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe,
               X_test_grade_ohe, X_test_category_ohe, X_test_subcategory_ohe,
               X_test_price_norm, X_test_teach_prev_norm,
               X_test_pj_title_tfidf)).tocsr()
```

In [141]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

chi2_features = SelectKBest(chi2, k = 2000)
X_kbest_features_X_tr = chi2_features.fit(abs(X_tr), y_train)
```

In [142]:

```
X_kbest_features_X_tr_t = chi2_features.transform(abs(X_tr))
X_kbest_features_X_cr = chi2_features.transform(abs(X_cr))
X_kbest_features_X_te = chi2_features.transform(abs(X_te))
```

In [145]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by "decision_function" on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
K = [1, 5, 11, 15, 21, 31, 41, 51, 101]
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i,algorithm = 'brute')
    neigh.fit(X_kbest_features_X_tr_t, y_train)

    y_train_pred = batch_predict(neigh, X_kbest_features_X_tr_t)
    y_cv_pred = batch_predict(neigh, X_kbest_features_X_cr)

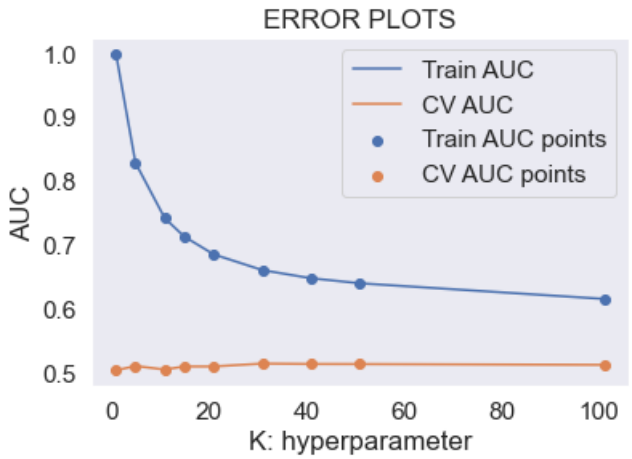
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(K, train_auc, label='Train AUC')
plt.plot(K, cv_auc, label='CV AUC')

plt.scatter(K, train_auc, label='Train AUC points')
```

```
plt.scatter(K, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [146]:

```
# Please compare all your models using Prettytable library
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

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

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

print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

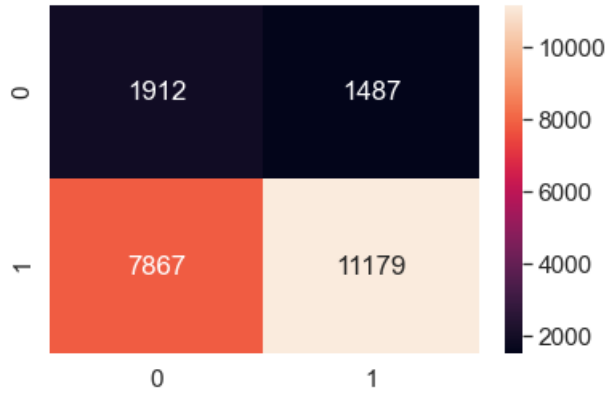
```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.3345872925235715 for threshold 0.851
[[ 1912  1487]
 [ 7867 11179]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2913723027327699 for threshold 0.871
[[1650  896]
 [7921 6033]]
```

In [147]:

```
ax = sns.heatmap(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)), annot=True, fmt="d")
```

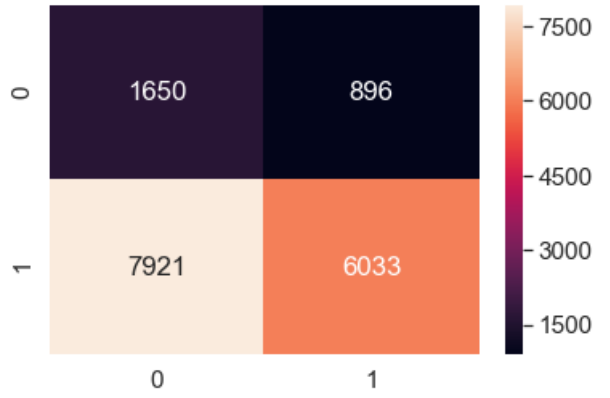
```
the maximum value of tpr*(1-fpr) 0.3345872925235715 for threshold 0.851
```



In [148]:

```
ax = sns.heatmap(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)),
annot=True, fmt="d")
```

```
the maximum value of tpr*(1-fpr) 0.2913723027327699 for threshold 0.871
```



3. Conclusions

In [149]:

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

In [150]:

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

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameter", "Train AUC", "Test AUC"]

x.add_row(["BOW","Brute",105,0.647,0.587])
x.add_row(["TFIDF","Brute",105,0.6322,0.546])
x.add_row(["W2V","Brute",101,0.620,0.548])
x.add_row(["TFIDF W2V","Brute",101,0.617,0.548])
print(x)
```

```
+------------+-------+----------------+-----------+----------+
| Vectorizer | Model | Hyperparameter | Train AUC | Test AUC |
+------------+-------+----------------+-----------+----------+
|    BOW     | Brute |      105       |   0.647   |  0.587   |
|   TFIDF    | Brute |      105       |   0.6322  |  0.546   |
|    W2V     | Brute |      101       |    0.62   |  0.548   |
| TFIDF W2V  | Brute |      101       |   0.617   |  0.548   |
+------------+-------+----------------+-----------+----------+
```

All the AUC scores was almost same for all the methods. By selecting the best k value the train auc was about 0.6 and test auc was 0.54. The value of the best k is 101.

In [ ]:

```
```