DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

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

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

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

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

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

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

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

Notes on the Essay Data

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

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_4:__ "Close by sharing why your project will make a difference"

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

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

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

In [1]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns

import re
from tqdm import tqdm

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 the 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)
print('='*100)
project_data.head(2)
```

```
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']
====================================================================
==========================
```

Out[3]:

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

Renaming and sorting the column name [project_submitted_datetime]

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/49702492/40840
39

#project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
#project_data.drop('project_submitted_datetime', axis = 1, inplace=True)
#project_data.sort_values(by = ['Date'], inplace = True)


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

#project_data.head(2)
```

In [5]:

```
# We can also do the same work by using pandas rename method.
project_data = project_data.rename(columns = {'project_submitted_datetime': 'Date'})

# Sorting the dataframe according to time
project_data['Date'] = pd.to_datetime(project_data['Date'])
project_data.sort_values(by = ['Date'], inplace = True)

project_data.head(2)
```

Out[5]:

	Unnamed: 0	id	teacher_id	teacher_prefix	schoc
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA

In [6]:

```
print("Number of data points in resource data", resource_data.shape)
print('-'*50)
print("The attributes of resource data :", resource_data.columns.values)
print('='*100)
resource_data.head(2)
```

```
Number of data points in resource data (1541272, 4)
--------------------------------------------------
The attributes of resource data : ['id' 'description' 'quantity' 'price']
====================================================================================
==========================
```

Out[6]:

	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 [7]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [8]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47
301924/4084039

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

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

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

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

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

1.3 Text preprocessing

In [9]:

```
# 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 [10]:

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

Out[10]:

	Unnamed: 0	id	teacher_id	teacher_prefix	schoo
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA

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

In [11]:

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

In [12]:

```
X.head(2)
```

Out[12]:

	Unnamed: 0	id	teacher_id	teacher_prefix	schoo
473	100660	p234804	cbc0e38f522143b86d372f8b43d4cff3	Mrs.	GA
41558	33679	p137682	06f6e62e17de34fcf81020c77549e1d5	Mrs.	WA

In [13]:

```
y[10:]
```

Out[13]:

```
array([0, 1, 1, ..., 1, 1, 1], dtype=int64)
```

In [14]:

```
# Splitting into train, cv and test
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, str
atify = y_train)
```

In [15]:

```
print('The shape of X_train & y_train: ',X_train.shape, y_train.shape)
print('The shape of X_cv & y_cv:       ',X_cv.shape, y_cv.shape)
print('The shape of X_test & y_test:   ',X_test.shape, y_test.shape)
```

```
The shape of X_train & y_train:  (22445, 17) (22445,)
The shape of X_cv & y_cv:        (11055, 17) (11055,)
The shape of X_test & y_test:    (16500, 17) (16500,)
```

In [16]:

```
# Printing some random text
print(X_train['essay'].values[0])
print("="*127)
print(X_train['essay'].values[500])
print("="*127)
print(X_train['essay'].values[1000])
print("="*127)
print(X_train['essay'].values[10000])
print("="*127)
print(X_train['essay'].values[20000])
print("="*127)
```

Growing up in rural Tennessee, I always knew that I was supposed to stay h
ere and make a difference in my community. My students are inquisitive an
d energetic middle-schoolers who live in a very sheltered community. My g
oal is quite simple: bring the world to them.\r\n\r\nMy students live in a
sheltered community and often are unaware of the beauty and opportunities
that lie outside our majestic valley. However, I want them to reach beyon
d their current challenges and see what lies beyond the mountains, beyond
the trees and rivers. There is a whole world out there that is calling to
them and I want to help them embrace the new world.Student in rural Tennes
see need to see and hear the outside world. They can utilize these headph
ones with microphones to work cooperatively in groups with students from o
ther schools around the world.\r\n\r\nCan You Hear Me Now will help my stu
dents understand the power of global education. They will see and interac
t with students from various countries and learn about their daily lives,
their education, and their culture. \r\n\r\nBy harnessing the power of Sky
pe, ePals, and Google Classroom, my students and I can begin to discover a
world that lies outside our majestic valley and travel the globe through t
he sights and sounds it has to offer.nannan
==========
==========
My classroom is a place of challenge and discovery. My first graders spend
the school year learning the foundation of knowledge and skills they will
need to grow as young readers, writers, mathematicians, historians, and sc
ientists. My students are hard-working, determined, and enthusiastic learn
ers. They have a zeal for learning and are eager to explore the world arou
nd them.\r\n\r\nIt is so important I provide all my students with the best
chance of self discovery through multiple engaging resources that promote
critical thinking.The Wonder Workshop Dash & Dot robots will be used as an
extremely motivating, engaging, and exciting way to teach students about c
oding and problem solving. Problem solving is such a huge skill in first g
rade! Allowing students to independently and collaboratively use an iPad t
o code the robot's actions will help promote these skills. The various acc
essories provided with the Wonder Workshop Dash and Dot Wonder Pack will p
rovide multiple layers of challenging activities that will encourage creat
ivity, nurture critical thinking and problem solving skills, and help faci
litate and develop students' coding skills. Having these robots will provi
de them with opportunities to prepare for how we learn and work together i
n the 21st century!nannan
==========
==========
Our rural, low-income, racially diverse high school in central Florida has
a free & reduced lunch rate of 67%. Approximately 15% of county residents
have a 4 yr. college degree. Most students will be the first in their fami
lies to go to college. Our students must possess incredible focus and dete
rmination to be college-ready and work hard to realize their goals. \r\nOu
r students are changing the game and beating the odds at the home of the B
luestreaks! During the past two years, our student registration for the SA
T and ACT has surged. Our 2016-17 FAFSA completion rate is above the state
average. Almost every senior at our school completes a college applicatio
n. \r\n\r\nOur students, who typically attend the local state college, are
also applying for \"reach\" schools and getting accepted! This year, accep
tances included selective Florida universities as well as ivy league colle
ges like Brown University and Yale.A pep rally event in the school courtya
rd on May 1 will highlight the college choices of individual seniors and e
ncourage younger students to set college planning goals. Teachers, adminis
trators and local dignitaries will also attend. Decorations, an emcee, the
school band, cheerleaders and refreshments for seniors will add to the fes
tivities.\r\n\r\nDoor prizes for college bound seniors will help bring a w
ell-deserved focus and reward to hard-working students who strive to lead
productive and fulfilling lives.\r\n\r\nThe remarkable academic achievemen
ts of our students will be recognized with great fanfare during Bluestreak

College Decision Day! \r\n\r\nWe would be grateful for your support.\r\nn
annan
==========
==========
My students come from a variety of diverse backgrounds and experiences. So
me live middle-class lives enjoying the love of both their parents, partic
ipating in extra-curricular activities, and focusing on school and their f
uture academics or careers. Others have had a very different experience, e
nduring the challenges of poverty and struggle. Often, these young people
know all too well that the odds are stacked against them, and they must wo
rk harder than their middle-class counterparts to overcome scarcity, negle
ct, cultural and linguistic barriers, or the academic and social challenge
s they experience every day.\r\n\r\nDespite these great advantages or obst
acles, each and every one of my beautiful students comes with the desire t
o improve their lives, as well as the lives of others. They know the impor
tance of a strong education, and have repeatedly demonstrated their abilit
ies as scholars to grow and explore during their time here at MLA. I am co
ntinually impressed with their passion for learning, and their concern for
the world around them. Their creativity, curiosity, and and persistence wi
ll guide them to any place they choose to go in their spectacular futures!
Students first will examine historical case studies, focusing on the role
of government or leadership, and analyzing the effects of dramatic change
on these communities. They will then read dystopian novels that connect to
what they have studied in an effort to examine similar events and experien
ces in the present.\r\n\r\nAs part of our studies of United States Histor
y, we are examining both past and present events that change the lives of
the participants, and alter their future in an often uncontrollable, and s
ometimes horrifying way. Through this study, my students can determine wha
t the problems are, and try to construct a solution to the situation, much
like what is needed in their real lives.nannan
==========
==========
My wonderful students are an amazingly cheerful and inquisitive group. The
se students were not accustomed to having a designated PE teacher, but the
y have welcomed me and eagerly accepted all of the challenges I have throw
n at them. This includes a successful inclusion experience for all of our
students. Our school includes 5 ASD classes and 3 SDC classes. \r\nI combi
ne all of these Special Ed classes with my General PE classes and teach th
e students how to celebrate their diversity, accept all of their peers no
matter their ability, and practice leadership skills as peer tutors. \r\nW
e emphasize positive social skills in low competition modified games and a
ctivities. Today we were discussing social skills and I tried to make an a
ngry face as an example, but they all started laughing because I just ca
n't be angry with all of those smiling faces looking back!In my Elementary
PE classes we focus on fun, friendship, and fitness and we have changed th
e name from Physical Education to Play Exercise. We love Adventure Challen
ges and I try to create a variety of new experiences for my students so th
at they can find aspects of PE that they truly enjoy. When the weather is
poor we need small space activities which will promote motor skill practic
e, cooperation, social skills, fitness, and critical thinking. \r\nSpeed s
tacking and scarf juggling are challenging movement activities which all s
tudents can succeed at in a small space. \r\nThese activities will test th
eir hand eye coordination and a full set will allow the whole class to pra
ctice at the same time and to challenge each other.nannan
==========
==========

In [17]:

```
# https://stackoverflow.com/a/47091490/4084039
# removing and replacing decontracted phrases.

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 [18]:

```
sent = decontracted(X_train['essay'].values[10000])
print(sent)
print("="*127)
```

```
My students come from a variety of diverse backgrounds and experiences. So
me live middle-class lives enjoying the love of both their parents, partic
ipating in extra-curricular activities, and focusing on school and their f
uture academics or careers. Others have had a very different experience, e
nduring the challenges of poverty and struggle. Often, these young people
know all too well that the odds are stacked against them, and they must wo
rk harder than their middle-class counterparts to overcome scarcity, negle
ct, cultural and linguistic barriers, or the academic and social challenge
s they experience every day.\r\n\r\nDespite these great advantages or obst
acles, each and every one of my beautiful students comes with the desire t
o improve their lives, as well as the lives of others. They know the impor
tance of a strong education, and have repeatedly demonstrated their abilit
ies as scholars to grow and explore during their time here at MLA. I am co
ntinually impressed with their passion for learning, and their concern for
the world around them. Their creativity, curiosity, and and persistence wi
ll guide them to any place they choose to go in their spectacular futures!
Students first will examine historical case studies, focusing on the role
of government or leadership, and analyzing the effects of dramatic change
on these communities. They will then read dystopian novels that connect to
what they have studied in an effort to examine similar events and experien
ces in the present.\r\n\r\nAs part of our studies of United States Histor
y, we are examining both past and present events that change the lives of
the participants, and alter their future in an often uncontrollable, and s
ometimes horrifying way. Through this study, my students can determine wha
t the problems are, and try to construct a solution to the situation, much
like what is needed in their real lives.nannan
===============================================================================
=======================================================
```

In [19]:

```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/

sent = sent.replace('\\r',' ')
sent = sent.replace('\\"',' ')
sent = sent.replace('\\n',' ')
print(sent)
```

```
My students come from a variety of diverse backgrounds and experiences. So
me live middle-class lives enjoying the love of both their parents, partic
ipating in extra-curricular activities, and focusing on school and their f
uture academics or careers. Others have had a very different experience, e
nduring the challenges of poverty and struggle. Often, these young people
know all too well that the odds are stacked against them, and they must wo
rk harder than their middle-class counterparts to overcome scarcity, negle
ct, cultural and linguistic barriers, or the academic and social challenge
s they experience every day.    Despite these great advantages or obstacle
s, each and every one of my beautiful students comes with the desire to im
prove their lives, as well as the lives of others. They know the importanc
e of a strong education, and have repeatedly demonstrated their abilities
as scholars to grow and explore during their time here at MLA. I am contin
ually impressed with their passion for learning, and their concern for the
world around them. Their creativity, curiosity, and and persistence will g
uide them to any place they choose to go in their spectacular futures!Stud
ents first will examine historical case studies, focusing on the role of g
overnment or leadership, and analyzing the effects of dramatic change on t
hese communities. They will then read dystopian novels that connect to wha
t they have studied in an effort to examine similar events and experiences
in the present.    As part of our studies of United States History, we are
examining both past and present events that change the lives of the partic
ipants, and alter their future in an often uncontrollable, and sometimes h
orrifying way. Through this study, my students can determine what the prob
lems are, and try to construct a solution to the situation, much like what
is needed in their real lives.nannan
```

In [20]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039

sent = re.sub('[^A-Za-z0-9]+',' ',sent)
print(sent)
```

```
My students come from a variety of diverse backgrounds and experiences Som
e live middle class lives enjoying the love of both their parents particip
ating in extra curricular activities and focusing on school and their futu
re academics or careers Others have had a very different experience enduri
ng the challenges of poverty and struggle Often these young people know al
l too well that the odds are stacked against them and they must work harde
r than their middle class counterparts to overcome scarcity neglect cultur
al and linguistic barriers or the academic and social challenges they expe
rience every day Despite these great advantages or obstacles each and ever
y one of my beautiful students comes with the desire to improve their live
s as well as the lives of others They know the importance of a strong educ
ation and have repeatedly demonstrated their abilities as scholars to grow
and explore during their time here at MLA I am continually impressed with
their passion for learning and their concern for the world around them The
ir creativity curiosity and and persistence will guide them to any place t
hey choose to go in their spectacular futures Students first will examine
historical case studies focusing on the role of government or leadership a
nd analyzing the effects of dramatic change on these communities They will
then read dystopian novels that connect to what they have studied in an ef
fort to examine similar events and experiences in the present As part of o
ur studies of United States History we are examining both past and present
events that change the lives of the participants and alter their future in
an often uncontrollable and sometimes horrifying way Through this study my
students can determine what the problems are and try to construct a soluti
on to the situation much like what is needed in their real lives nannan
```

In [21]:

```
from nltk.corpus import stopwords
stopwords = stopwords.words('english')   # To remove words that comes under the stopwor
d.
print(stopwords)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "yo
u're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselv
es', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'hersel
f', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'the
mselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'thes
e', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'hav
e', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'th
e', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'a
t', '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', 'on
ce', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'no
t', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can',
'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll',
'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't",
'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',
"mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't",
'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "would
n't"]
```

1.3.2 Preprocessing of train data: (X_train['essay'])

In [22]:

```
# Combining all the above statements.

from tqdm import tqdm
preprocessed_essay_train = []

for sentence in tqdm(X_train['essay'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\\r',' ')
    sent = sent.replace('\\"',' ')
    sent = sent.replace('\\n',' ')
    sent = re.sub('[^A-Za-z0-9]+',' ',sent)
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essay_train.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████
██████| 22445/22445 [00:22<00:00, 981.06it/s]
```

In [23]:

```
# After preprocessing
preprocessed_essay_train[10000]
```

Out[23]:

```
'my students come variety diverse backgrounds experiences some live middle
class lives enjoying love parents participating extra curricular activitie
s focusing school future academics careers others different experience end
uring challenges poverty struggle often young people know well odds stacke
d must work harder middle class counterparts overcome scarcity neglect cul
tural linguistic barriers academic social challenges experience every day
despite great advantages obstacles every one beautiful students comes desi
re improve lives well lives others they know importance strong education r
epeatedly demonstrated abilities scholars grow explore time mla i continua
lly impressed passion learning concern world around their creativity curio
sity persistence guide place choose go spectacular futures students first
examine historical case studies focusing role government leadership analyz
ing effects dramatic change communities they read dystopian novels connect
studied effort examine similar events experiences present as part studies
united states history examining past present events change lives participa
nts alter future often uncontrollable sometimes horrifying way through stu
dy students determine problems try construct solution situation much like
needed real lives nannan'
```

1.3.3 Preprocessing of cross validation data: (X_cv['essay'])

In [24]:

```
preprocessed_essay_cv = []

for sentence in tqdm(X_cv['essay'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\\r',' ')
    sent = sent.replace('\\"',' ')
    sent = sent.replace('\\n',' ')
    sent = re.sub('[^A-Za-z0-9]+',' ',sent)
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essay_cv.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████
██████| 11055/11055 [00:11<00:00, 980.90it/s]
```

In [25]:

```
# After preprocessing:
preprocessed_essay_cv[10000]
```

Out[25]:

```
'my students come school excited learn they love come front room really ta
ke i teaching i believe creating safe comfortable environment i teach area
70 population speaks second language come low income families my students
loving kind one another they work together help despite difficulties face
home i teaching 5th grade year title i school i want make sure students ne
eded succeed classroom i always encouraging students take ownership learni
ng flexible seating allows students choice sit based individual preference
needs our classroom culture based upon independence self directed learning
being able choose seat matches learn best promote self evaluation students
collaboration communication critical thinking also improve classroom conve
rts flexible seating arrangement flexible seating decreases fidgety behavi
or improves focus increases attention behavior donations towards project h
elp get yoga balls yoga mats lap desks students students allowed move arou
nd freely try different seating based upon needs i believe flexible seatin
g allow students become motivated engaged learning nannan'
```

1.3.4 Preprocessing of test data: (X_test['essay'])

In [26]:

```
preprocessed_essay_test = []

for sentence in tqdm(X_test['essay'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\\r',' ')
    sent = sent.replace('\\"',' ')
    sent = sent.replace('\\n',' ')
    sent = re.sub('[^A-Za-z0-9]+',' ',sent)
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essay_test.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████
██████| 16500/16500 [00:17<00:00, 940.29it/s]
```

In [27]:

```
# After preprocessing
preprocessed_essay_test[10000]
```

Out[27]:

```
'today challenge teacher keeping ever changing technology helping students
learn use adding challenge fact students computer tablet home our school 9
0 95 free reduced lunch consists mostly minority students they come school
little background knowledge formal daycare preschool program students also
come limited resources available lack financial capabilities a lot kids sp
end hours week playing ipad much time spent collaborative activities encou
rage learning creativity the osmo game system ipad accessory consists stan
d camera attachment specially designed games my students use osmo system l
earn coding work words solve puzzles kids work collaborative groups find w
ays become avid programmers solve problems numbers words this something 2n
d grade students would never exposed without gaming system nannan'
```

1.4 Preprocessing of Project titles :

In [28]:

```
# Printing some random project titles:
print(X_train['project_title'].values[0])
print("="*60)
print(X_train['project_title'].values[500])
print("="*60)
print(X_train['project_title'].values[1000])
print("="*60)
print(X_train['project_title'].values[10000])
print("="*60)
print(X_train['project_title'].values[20000])
print("="*60)
```

```
Can You Hear Me Now?
============================================================
STEM to Life With Wonder Workshop
============================================================
Bluestreak College Decision Day!
============================================================
Dystopian Literature Circles
============================================================
Indoor Adventures for PE (Play Exercise)!
============================================================
```

1.4.1 Preprocessing project title of train data: (X_train['project_title'])

In [29]:

```
preprocessed_title_train = []

for sentence in tqdm(X_train['project_title'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\\r',' ')
    sent = sent.replace('\\"',' ')
    sent = sent.replace('\\n',' ')
    sent = re.sub('[^A-Za-z0-9]+',' ',sent)
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_title_train.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████
████| 22445/22445 [00:01<00:00, 21380.31it/s]
```

In [30]:

```
# After preprocessing:
preprocessed_title_train[20000]
```

Out[30]:

```
'indoor adventures pe play exercise'
```

1.4.2 Preprocessing project title of Cross validation data: (X_cv['project_title'])

In [31]:

```
preprocessed_title_cv = []

for sentence in tqdm(X_cv['project_title'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\\r',' ')
    sent = sent.replace('\\"',' ')
    sent = sent.replace('\\n',' ')
    sent = re.sub('[^A-Za-z0-9]+',' ',sent)
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_title_cv.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████
████| 11055/11055 [00:00<00:00, 20989.43it/s]
```

In [32]:

```
# After preprocessing :
preprocessed_title_cv[10000]
```

Out[32]:

```
'flexible seating active learners'
```

1.4.3 Preprocessing project title of test data: (X_test['project_title'])

In [33]:

```
preprocessed_title_test = []

for sentence in tqdm(X_test['project_title'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\\r',' ')
    sent = sent.replace('\\"',' ')
    sent = sent.replace('\\n',' ')
    sent = re.sub('[^A-Za-z0-9]+',' ',sent)
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_title_test.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████
████| 16500/16500 [00:00<00:00, 21514.10it/s]
```

In [34]:

```
# After preprocessing :
preprocessed_title_test[10000]
```

Out[34]:

```
'coding second'
```

1.5 Preparing Data for Models :

In [35]:

```
project_data.columns
```

Out[35]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_
1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       'clean_categories', 'clean_subcategories', 'essay'],
      dtype='object')
```

we are going to consider

- school_state - Categorical data
- clean_categories - Categorical data
- clean_subcategories - Categorical data
- project_grade_category - Categorical data
- teacher_prefix - Categorical data

- project_essay - Text data
- project_title - Text data
- project_resource_summary - Text data (optional)

- quantity - numerical (optional)
- teacher_number_of_previously_posted_projects - numerical
- price - numerical

1.5.1 Vectorizing Categorical data:

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

a. One Hot Encoding of School State :

In [36]:

```
# Using CountVectorizer to convert the categorical data into one hote encoders:
my_counter = Counter()
for state in project_data['school_state'].values:
    my_counter.update(state.split())

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

In [37]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary = list(sorted_dict_state_cat.items()), lowercase=False, binary=True)
vectorizer.fit(X_train['school_state'].values)

school_state_one_hot_train = vectorizer.transform(X_train['school_state'].values)
school_state_one_hot_cv = vectorizer.transform(X_cv['school_state'].values)
school_state_one_hot_test = vectorizer.transform(X_test['school_state'].values)

print(vectorizer.get_feature_names())
print("="*90)

print("The shape of school_state_one_hot_train : ",school_state_one_hot_train.shape)
print("The shape of school_state_one_hot_cv    : ",school_state_one_hot_cv.shape)
print("The shape of school_state_one_hot_test  : ",school_state_one_hot_test.shape)
```

```
[('VT', 32), ('WY', 51), ('ND', 63), ('MT', 106), ('RI', 126), ('NH', 14
1), ('SD', 142), ('NE', 144), ('AK', 153), ('DE', 155), ('WV', 218), ('M
E', 222), ('NM', 236), ('HI', 239), ('DC', 247), ('KS', 285), ('ID', 302),
('IA', 306), ('AR', 446), ('CO', 538), ('MN', 556), ('OR', 577), ('MS', 59
8), ('KY', 614), ('NV', 665), ('MD', 668), ('TN', 774), ('CT', 774), ('A
L', 790), ('UT', 792), ('WI', 833), ('VA', 916), ('AZ', 994), ('NJ', 100
5), ('OK', 1074), ('MA', 1076), ('LA', 1094), ('WA', 1103), ('MO', 1166),
('IN', 1171), ('OH', 1180), ('PA', 1419), ('MI', 1468), ('GA', 1828), ('S
C', 1830), ('IL', 1967), ('NC', 2340), ('FL', 2839), ('TX', 3320), ('NY',
3393), ('CA', 7024)]
==========================================================================
================
The shape of school_state_one_hot_train :  (22445, 51)
The shape of school_state_one_hot_cv    :  (11055, 51)
The shape of school_state_one_hot_test  :  (16500, 51)
```

b. One Hot Encoding of clean_categories :

In [38]:

```
vectorizer = CountVectorizer(vocabulary = list(sorted_cat_dict.items()), lowercase = False, binary = True)
vectorizer.fit(X_train['clean_categories'].values)

categories_one_hot_train = vectorizer.transform(X_train['clean_categories'].values)
categories_one_hot_cv = vectorizer.transform(X_cv['clean_categories'].values)
categories_one_hot_test = vectorizer.transform(X_test['clean_categories'].values)

print(vectorizer.get_feature_names())
print("="*90)

print("The shape of categories_one_hot_train: ", categories_one_hot_train.shape)
print("The shape of categories_one_hot_cv   : ", categories_one_hot_cv.shape)
print("The shape of categories_one_hot_test : ", categories_one_hot_test.shape)
```

```
[('Warmth', 643), ('Care_Hunger', 643), ('History_Civics', 2689), ('Music_
Arts', 4699), ('AppliedLearning', 5569), ('SpecialNeeds', 6233), ('Health_
Sports', 6538), ('Math_Science', 18874), ('Literacy_Language', 23998)]
==========================================================================
================
The shape of categories_one_hot_train:  (22445, 9)
The shape of categories_one_hot_cv   :  (11055, 9)
The shape of categories_one_hot_test :  (16500, 9)
```

c. One Hot Encoding of clean_subcategories :

In [39]:

```
vectorizer = CountVectorizer(vocabulary = list(sorted_sub_cat_dict.items()), lowercase
= False, binary = True)
vectorizer.fit(X_train['clean_subcategories'].values)

subcategories_one_hot_train = vectorizer.transform(X_train['clean_subcategories'].value
s)
subcategories_one_hot_cv = vectorizer.transform(X_cv['clean_subcategories'].values)
subcategories_one_hot_test = vectorizer.transform(X_test['clean_subcategories'].values)

print(vectorizer.get_feature_names())
print("="*90)

print("The shape of subcategories_one_hot_train: ", subcategories_one_hot_train.shape)
print("The shape of subcategories_one_hot_cv   : ", subcategories_one_hot_cv.shape)
print("The shape of subcategories_one_hot_test : ", subcategories_one_hot_test.shape)
```

```
[('Economics', 127), ('CommunityService', 214), ('FinancialLiteracy', 25
3), ('ParentInvolvement', 302), ('Extracurricular', 373), ('Civics_Governm
ent', 380), ('ForeignLanguages', 388), ('NutritionEducation', 617), ('Warm
th', 643), ('Care_Hunger', 643), ('SocialSciences', 864), ('PerformingArt
s', 910), ('CharacterEducation', 958), ('TeamSports', 995), ('Other', 112
8), ('College_CareerPrep', 1168), ('Music', 1432), ('History_Geography', 1
433), ('Health_LifeScience', 1876), ('EarlyDevelopment', 1937), ('ESL', 19
99), ('Gym_Fitness', 2068), ('EnvironmentalScience', 2533), ('VisualArts',
2865), ('Health_Wellness', 4732), ('AppliedSciences', 4901), ('SpecialNeed
s', 6233), ('Literature_Writing', 10127), ('Mathematics', 12832), ('Litera
cy', 15611)]
==========================================================================
================
The shape of subcategories_one_hot_train:  (22445, 30)
The shape of subcategories_one_hot_cv   :  (11055, 30)
The shape of subcategories_one_hot_test :  (16500, 30)
```

d. One Hot Encoding of project_grade_category :

In [40]:

```
# Setting project_data['project_grade_category'].column :-
project_grade_category = []

for i in range(len(project_data)):
    x = project_data['project_grade_category'][i].replace(' ','_')
    project_grade_category.append(x)

project_grade_category[0:5]
```

Out[40]:

```
['Grades_PreK-2', 'Grades_6-8', 'Grades_6-8', 'Grades_PreK-2', 'Grades_Pre
K-2']
```

In [41]:

```
project_data['project_grade_category'] = project_grade_category

project_data['project_grade_category'].head(5)
```

Out[41]:

```
473      Grades_PreK-2
41558       Grades_6-8
29891       Grades_6-8
23374    Grades_PreK-2
49228    Grades_PreK-2
Name: project_grade_category, dtype: object
```

In [42]:

```
my_counter = Counter()

for grades in project_data['project_grade_category'].values:
    my_counter.update(grades.split())

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

In [43]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_category.items()), lo
wercase=False, binary=True)
vectorizer.fit(X_train['project_grade_category'].values)

project_grade_category_one_hot_train = vectorizer.transform(X_train['project_grade_cate
gory'].values)
project_grade_category_one_hot_cv = vectorizer.transform(X_cv['project_grade_category']
.values)
project_grade_category_one_hot_test = vectorizer.transform(X_test['project_grade_catego
ry'].values)

print(vectorizer.get_feature_names())
print("="*80)


print("The shape of project_grade_category_one_hot_train: ", project_grade_category_one
_hot_train.shape)
print("The shape of project_grade_category_one_hot_cv   : ", project_grade_category_one
_hot_cv.shape)
print("The shape of project_grade_category_one_hot_test : ", project_grade_category_one
_hot_test.shape)
```

```
[('Grades_9-12', 4966), ('Grades_6-8', 7750), ('Grades_3-5', 16968), ('Gra
des_PreK-2', 20316)]
================================================================================
======
The shape of project_grade_category_one_hot_train:  (22445, 4)
The shape of project_grade_category_one_hot_cv   :  (11055, 4)
The shape of project_grade_category_one_hot_test :  (16500, 4)
```

e. One Hot Encoding of teacher_prefix :

In [44]:

```
my_counter = Counter()

for prefix in project_data['teacher_prefix'].values:
    prefix = str(prefix)
    my_counter.update(prefix.split())

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

In [45]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_prefix.items()), lowercase=
False, binary=True)
vectorizer.fit(X_train['teacher_prefix'].values.astype('unicode'))

teacher_prefix_one_hot_train = vectorizer.transform(X_train['teacher_prefix'].values.as
type('unicode'))
teacher_prefix_one_hot_cv = vectorizer.transform(X_cv['teacher_prefix'].values.astype(
'unicode'))
teacher_prefix_one_hot_test = vectorizer.transform(X_test['teacher_prefix'].values.asty
pe('unicode'))

print(vectorizer.get_feature_names())
print("="*80)

print("The shape of teacher_prefix_one_hot_train: ",teacher_prefix_one_hot_train.shape)
print("The shape of teacher_prefix_one_hot_cv   : ",teacher_prefix_one_hot_cv.shape)
print("The shape of teacher_prefix_one_hot_test : ",teacher_prefix_one_hot_test.shape)
```

```
[('nan', 2), ('Dr.', 2), ('Teacher', 1061), ('Mr.', 4859), ('Ms.', 17936),
('Mrs.', 26140)]
================================================================================
======
The shape of teacher_prefix_one_hot_train:  (22445, 6)
The shape of teacher_prefix_one_hot_cv   :  (11055, 6)
The shape of teacher_prefix_one_hot_test :  (16500, 6)
```

1.5.2 Vectorizing Textual data: (project_essay & project_title)

1.5.2.1 Bag of Words :(project_essay)

a) Bag of word - X_train

In [46]:

```
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(preprocessed_essay_train)

project_essay_BOW_train = vectorizer.transform(preprocessed_essay_train)
print("The shape of project_essay_BOW_train after processing: ", project_essay_BOW_trai
n.shape)
```

```
The shape of project_essay_BOW_train after processing:  (22445, 8905)
```

b) Bag of word - X_cv

In [47]:

```
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).

project_essay_BOW_cv = vectorizer.transform(preprocessed_essay_cv)
print("The shape of project_essay_BOW_cv after processing: ", project_essay_BOW_cv.shap
e)
```

```
The shape of project_essay_BOW_cv after processing:  (11055, 8905)
```

b) Bag of word - X_test

In [48]:

```
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects).

project_essay_BOW_test = vectorizer.transform(preprocessed_essay_test)
print("The shape of project_essay_BOW_test after processing: ", project_essay_BOW_test.
shape)
```

```
The shape of project_essay_BOW_test after processing:  (16500, 8905)
```

Bag of Words :(project_title)

a) X_train

In [49]:

```
vectorizer.fit(preprocessed_title_train)
project_title_BOW_train = vectorizer.transform(preprocessed_title_train)
print("The shape of project_title_BOW_train: ",project_title_BOW_train.shape)
```

```
The shape of project_title_BOW_train:  (22445, 1238)
```

b) X_cv

In [50]:

```
project_title_BOW_cv = vectorizer.transform(preprocessed_title_cv)
print("The shape of project_title_cv: ",project_title_BOW_cv.shape)
```

```
The shape of project_title_cv:  (11055, 1238)
```

c) X_test

In [51]:

```
project_title_BOW_test = vectorizer.transform(preprocessed_title_test)
print("The shape of project_title_test: ",project_title_BOW_test.shape)
```

```
The shape of project_title_test:  (16500, 1238)
```

1.5.2.2 TFIDF :(project_essay)

a) X_train

In [52]:

```
# We are considering only the words which appeared in at least 10 documents(rows or pro
jects)

from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vector = TfidfVectorizer(min_df=10)

tfidf_vector.fit(preprocessed_essay_train)

project_essay_tfidf_train = tfidf_vector.transform(preprocessed_essay_train)
print("The shape of project_essay_tfidf_train: ", project_essay_tfidf_train.shape)
```

```
The shape of project_essay_tfidf_train:  (22445, 8905)
```

b) X_cv

In [53]:

```
project_essay_tfidf_cv = tfidf_vector.transform(preprocessed_essay_cv)
print("The shape of project_essay_tfidf_cv: ", project_essay_tfidf_cv.shape)
```

```
The shape of project_essay_tfidf_cv:  (11055, 8905)
```

c) X_test

In [54]:

```
project_essay_tfidf_test = tfidf_vector.transform(preprocessed_essay_test)
print("The shape of project_essay_tfidf_test: ", project_essay_tfidf_test.shape)
```

```
The shape of project_essay_tfidf_test:  (16500, 8905)
```

TFIDF :(project_title)

a) X_train

In [55]:

```
tfidf_vector.fit(preprocessed_title_train)
project_title_tfidf_train = tfidf_vector.transform(preprocessed_title_train)

print("The shape of project_title_tfidf_train: ",project_title_tfidf_train.shape)
```

```
The shape of project_title_tfidf_train:  (22445, 1238)
```

b) X_cv

In [56]:

```
project_title_tfidf_cv = tfidf_vector.transform(preprocessed_title_cv)

print("The shape of project_title_tfidf_cv: ",project_title_tfidf_cv.shape)
```

```
The shape of project_title_tfidf_cv:  (11055, 1238)
```

c) X_test

In [57]:

```
project_title_tfidf_test = tfidf_vector.transform(preprocessed_title_test)

print("The shape of project_title_tfidf_test: ",project_title_tfidf_test.shape)
```

```
The shape of project_title_tfidf_test:  (16500, 1238)
```

1.5.2.3 Using pre-trained model : avg w2v

In [58]:

```
# 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
```

In [59]:

```
model = loadGloveModel('glove.42B.300d.txt')
```

```
Loading Glove Model

1917495it [09:15, 3450.34it/s]

Done. 1917495  words loaded!
```

In [60]:

```
# Find the total number of words in the Train data of Essays.
total_words_train_essay = []

for word in preprocessed_essay_train:
    total_words_train_essay.extend(word.split(' '))

print("Total words in the train essay corpus: ", len(total_words_train_essay))
```

```
Total words in the train essay corpus:  3366602
```

In [61]:

```
# Finding unique words in train essay corpus.

words_unique_train_essay = set(total_words_train_essay)    # set contains only unique w
ords
print("Unique words in train essay corpus ", len(words_unique_train_essay))
```

```
Unique words in train essay corpus  30508
```

In [62]:

```
# Finding number of words present both in  glove vector and in our corpus.
common_word = set(model.keys()).intersection(words_unique_train_essay)

print("The number of words present in Glove vectors and in our corpus ",len(common_word
),"& is {} of train essay corpus"
      .format(np.round(len(common_word)/len(words_unique_train_essay)*100,3)))
```

```
The number of words present in Glove vectors and in our corpus  28809 & is
94.431 of train essay corpus
```

In [63]:

```
words_corpus_train = {}

words_glove = set(model.keys())

for i in words_unique_train_essay:
    if i in words_glove:
        words_corpus_train[i] = model[i]

print("Word2Vec length is ", len(words_corpus_train))
```

```
Word2Vec length is  28809
```

In [64]:

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

import pickle
with open('glove_vectors','wb') as f :       # glove_vectors is our filename
    pickle.dump(words_corpus_train,f)
```

In [65]:

```
# stronging variables into pickle files python:
# http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file

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

a) Avg w2v X_train

In [66]:

```
# Computing average w2v of train essay
avg_w2v_vectors_essay_train = []

for sentence in tqdm(preprocessed_essay_train):  # for each sentence
    vector = np.zeros(300)  # as word vectors are of zero length  # 300 dimensions.
    count_words = 0         # num of words with a valid vector in the sentence.

    for word in sentence.split(): # for each word in a sentence
        if word in glove_words:
            vector = vector + model[word]
            count_words = count_words + 1
    if count_words != 0:
        vector = vector/count_words

    avg_w2v_vectors_essay_train.append(vector)

print(len(avg_w2v_vectors_essay_train))
print(len(avg_w2v_vectors_essay_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█████| 22445/22445 [00:10<00:00, 2053.88it/s]

22445
300
```

b) Avg w2v X_cv

In [67]:

```
# Computing average w2v of cross validation essay
avg_w2v_vectors_essay_cv = []

for sentence in tqdm(preprocessed_essay_cv):  # for each sentence
    vector = np.zeros(300)  # as word vectors are of zero length  # 300 dimensions.
    count_words = 0         # num of words with a valid vector in the sentence.

    for word in sentence.split(): # for each word in a sentence
        if word in glove_words:
            vector = vector + model[word]
            count_words = count_words + 1
    if count_words != 0:
        vector = vector/count_words

    avg_w2v_vectors_essay_cv.append(vector)

print(len(avg_w2v_vectors_essay_cv))
print(len(avg_w2v_vectors_essay_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█████| 11055/11055 [00:08<00:00, 1330.77it/s]

11055
300
```

c) Avg w2v X_test

In [68]:

```
# Computing average w2v of test essay
avg_w2v_vectors_essay_test = []

for sentence in tqdm(preprocessed_essay_test):  # for each sentence
    vector = np.zeros(300)  # as word vectors are of zero length  # 300 dimensions.
    count_words = 0         # num of words with a valid vector in the sentence.

    for word in sentence.split(): # for each word in a sentence
        if word in glove_words:
            vector = vector + model[word]
            count_words = count_words + 1
    if count_words != 0:
        vector = vector/count_words

    avg_w2v_vectors_essay_test.append(vector)

print(len(avg_w2v_vectors_essay_test))
print(len(avg_w2v_vectors_essay_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
█████| 16500/16500 [00:08<00:00, 1888.96it/s]

16500
300
```

Avg w2v of project title:

a) avg w2v X_train

In [69]:

```
# Computing average w2v of train title
avg_w2v_vectors_title_train = []

for sentence in tqdm(preprocessed_title_train):  # for each sentence
    vector = np.zeros(300)  # as word vectors are of zero length  # 300 dimensions.
    count_words = 0         # num of words with a valid vector in the sentence.

    for word in sentence.split(): # for each word in a sentence
        if word in glove_words:
            vector = vector + model[word]
            count_words = count_words + 1
    if count_words != 0:
        vector = vector/count_words

    avg_w2v_vectors_title_train.append(vector)

print(len(avg_w2v_vectors_title_train))
print(len(avg_w2v_vectors_title_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
████| 22445/22445 [00:00<00:00, 28820.52it/s]

22445
300
```

b) avg w2v X_cv

In [70]:

```
# Computing average w2v of cross validation title
avg_w2v_vectors_title_cv = []

for sentence in tqdm(preprocessed_title_cv):  # for each sentence
    vector = np.zeros(300)  # as word vectors are of zero length  # 300 dimensions.
    count_words = 0         # num of words with a valid vector in the sentence.

    for word in sentence.split(): # for each word in a sentence
        if word in glove_words:
            vector = vector + model[word]
            count_words = count_words + 1
    if count_words != 0:
        vector = vector/count_words

    avg_w2v_vectors_title_cv.append(vector)

print(len(avg_w2v_vectors_title_cv))
print(len(avg_w2v_vectors_title_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
████| 11055/11055 [00:00<00:00, 30947.69it/s]

11055
300
```

c) avg w2v X_test

In [71]:

```
# Computing average w2v of test title
avg_w2v_vectors_title_test = []

for sentence in tqdm(preprocessed_title_test):  # for each sentence
    vector = np.zeros(300)  # as word vectors are of zero length  # 300 dimensions.
    count_words = 0         # num of words with a valid vector in the sentence.

    for word in sentence.split(): # for each word in a sentence
        if word in glove_words:
            vector = vector + model[word]
            count_words = count_words + 1
    if count_words != 0:
        vector = vector/count_words

    avg_w2v_vectors_title_test.append(vector)

print(len(avg_w2v_vectors_title_test))
print(len(avg_w2v_vectors_title_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
████| 16500/16500 [00:00<00:00, 30503.21it/s]

16500
300
```

1.5.2.4 Using pre-trained model :TFIDF weighted avg w2v

In [72]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]

tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essay_train)
# we are converting a dictionary with features 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())
```

TFIDF weighted avg w2v of project essay

a) X_train

In [73]:

```
# TFIDF weighted W2W
# computing TFIDF weighted for project essay train
tfidf_w2v_project_essay_train = []   # to store the tfidf w2v of each sentence in list
 format.

for sentence in tqdm(preprocessed_essay_train):
    vector = np.zeros(300)       # length of word vectors
    tf_idf_weight = 0            # num of words with a valid vector in the sentence.

    for word in sentence.split():  # for each word in the 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((sent
ence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # tf
idf values of each word.
            vector = vector + (vec * tf_idf)
            tf_idf_weight = tf_idf_weight + tf_idf
    if tf_idf_weight != 0:
        vector = vector/tf_idf_weight
    tfidf_w2v_project_essay_train.append(vector)

print(len(tfidf_w2v_project_essay_train))
print(len(tfidf_w2v_project_essay_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
██████| 22445/22445 [01:34<00:00, 238.36it/s]

22445
300
```

b) X_cv

In [74]:

```
# TFIDF weighted W2W
# computing TFIDF weighted for project essay cross validation
tfidf_w2v_project_essay_cv = []   # to store the tfidf w2v of each sentence in list for
mat.

for sentence in tqdm(preprocessed_essay_cv):
    vector = np.zeros(300)       # length of word vectors
    tf_idf_weight = 0            # num of words with a valid vector in the sentence.

    for word in sentence.split():  # for each word in the 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((sent
ence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # tf
idf values of each word.
            vector = vector + (vec * tf_idf)
            tf_idf_weight = tf_idf_weight + tf_idf
    if tf_idf_weight != 0:
        vector = vector/tf_idf_weight
    tfidf_w2v_project_essay_cv.append(vector)

print(len(tfidf_w2v_project_essay_cv))
print(len(tfidf_w2v_project_essay_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████
██████| 11055/11055 [00:45<00:00, 242.97it/s]

11055
300
```

c) X_test

In [75]:

```
# TFIDF weighted W2W
# computing TFIDF weighted for project essay test
tfidf_w2v_project_essay_test = []   # to store the tfidf w2v of each sentence in list f
ormat.

for sentence in tqdm(preprocessed_essay_test):
    vector = np.zeros(300)       # length of word vectors
    tf_idf_weight = 0            # num of words with a valid vector in the sentence.

    for word in sentence.split():  # for each word in the 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((sent
ence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # tf
idf values of each word.
            vector = vector + (vec * tf_idf)
            tf_idf_weight = tf_idf_weight + tf_idf
    if tf_idf_weight != 0:
        vector = vector/tf_idf_weight
    tfidf_w2v_project_essay_test.append(vector)

print(len(tfidf_w2v_project_essay_test))
print(len(tfidf_w2v_project_essay_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████
██████| 16500/16500 [01:08<00:00, 240.51it/s]

16500
300
```

TFIDF weighted avg w2v of project title

In [76]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]

tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_title_train)
# we are converting a dictionary with features 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())
```

a) X_train

In [77]:

```
# TFIDF weighted W2W
# computing TFIDF weighted for project title train
tfidf_w2v_project_title_train = []   # to store the tfidf w2v of each sentence in list
 format.

for sentence in tqdm(preprocessed_title_train):
    vector = np.zeros(300)       # length of word vectors
    tf_idf_weight = 0            # num of words with a valid vector in the sentence.

    for word in sentence.split():  # for each word in the 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((sent
ence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # tf
idf values of each word.
            vector = vector + (vec * tf_idf)
            tf_idf_weight = tf_idf_weight + tf_idf
    if tf_idf_weight != 0:
        vector = vector/tf_idf_weight
    tfidf_w2v_project_title_train.append(vector)

print(len(tfidf_w2v_project_title_train))
print(len(tfidf_w2v_project_title_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
████| 22445/22445 [00:01<00:00, 14642.13it/s]

22445
300
```

b) X_cv

In [78]:

```
# TFIDF weighted W2W
# computing TFIDF weighted for project title cross validation
tfidf_w2v_project_title_cv = []   # to store the tfidf w2v of each sentence in list for
mat.

for sentence in tqdm(preprocessed_title_cv):
    vector = np.zeros(300)       # length of word vectors
    tf_idf_weight = 0            # num of words with a valid vector in the sentence.

    for word in sentence.split():  # for each word in the 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((sent
ence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # tf
idf values of each word.
            vector = vector + (vec * tf_idf)
            tf_idf_weight = tf_idf_weight + tf_idf
    if tf_idf_weight != 0:
        vector = vector/tf_idf_weight
    tfidf_w2v_project_title_cv.append(vector)

print(len(tfidf_w2v_project_title_cv))
print(len(tfidf_w2v_project_title_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████
████| 11055/11055 [00:00<00:00, 15115.34it/s]

11055
300
```

c) X_test

In [79]:

```
# TFIDF weighted W2W
# computing TFIDF weighted for project title test
tfidf_w2v_project_title_test = []   # to store the tfidf w2v of each sentence in list f
ormat.

for sentence in tqdm(preprocessed_title_test):
    vector = np.zeros(300)       # length of word vectors
    tf_idf_weight = 0            # num of words with a valid vector in the sentence.

    for word in sentence.split():  # for each word in the 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((sent
ence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # tf
idf values of each word.
            vector = vector + (vec * tf_idf)
            tf_idf_weight = tf_idf_weight + tf_idf
    if tf_idf_weight != 0:
        vector = vector/tf_idf_weight
    tfidf_w2v_project_title_test.append(vector)

print(len(tfidf_w2v_project_title_test))
print(len(tfidf_w2v_project_title_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████
████| 16500/16500 [00:01<00:00, 14978.59it/s]

16500
300
```

1.5.3 Encoding Numerical Features:

a) Price

In [80]:

```
resource_data.head()
```

Out[80]:

	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
2	p069063	Cory Stories: A Kid's Book About Living With Adhd	1	8.45
3	p069063	Dixon Ticonderoga Wood-Cased #2 HB Pencils, Bo...	2	13.59
4	p069063	EDUCATIONAL INSIGHTS FLUORESCENT LIGHT FILTERS...	3	24.95

In [81]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in-one-step
price_data = resource_data.groupby('id').agg({'price' : 'sum', 'quantity' : 'sum'}).reset_index()

price_data.head()
```

Out[81]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21
2	p000003	298.97	4
3	p000004	1113.69	98
4	p000005	485.99	8

In [82]:

```
# We need to join the X_train, X_cv and X_test with price_data to proceed further.

X_train = pd.merge(X_train,price_data, on = 'id', how = 'left')
X_cv    = pd.merge(X_cv,   price_data, on = 'id', how = 'left')
X_test  = pd.merge(X_test, price_data, on = 'id', how = 'left')
```

In [83]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
```

In [84]:

```
normalizer.fit(X_train['price'].values.reshape(-1,1))

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


print("After vectorizations")
print(price_train.shape, y_train.shape)
print(price_cv.shape, y_cv.shape)
print(price_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
==========================
```

b) Quantity:

In [85]:

```
normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
```

In [86]:

```
normalizer.fit(X_train['quantity'].values.reshape(-1,1))

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


print("After vectorizations")
print(quantity_train.shape, y_train.shape)
print(quantity_cv.shape, y_cv.shape)
print(quantity_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================================
==========================
```

c) Number of previously posted projects by Teachers:

In [87]:

```
normalizer = Normalizer()

# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
```

In [88]:

```
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-
1,1))

previously_posted_projects_train = normalizer.transform(X_train['teacher_number_of_prev
iously_posted_projects'].values.reshape(-1,1))
previously_posted_projects_cv = normalizer.transform(X_cv['teacher_number_of_previously
_posted_projects'].values.reshape(-1,1))
previously_posted_projects_test = normalizer.transform(X_test['teacher_number_of_previo
usly_posted_projects'].values.reshape(-1,1))


print("After vectorizations")
print(previously_posted_projects_train.shape, y_train.shape)
print(previously_posted_projects_cv.shape, y_cv.shape)
print(previously_posted_projects_test.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
====================================================================================
==========================
```

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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
   - Use gridsearch-cv or randomsearch-cv or 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

4. **[Task-2]**

   - Select top 2000 features from feature Set 2 using `SelectKBest` (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) 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 (http://zetcode.com/python/prettytable/)

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. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

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

Merging all the features for the final Data matrix

In [89]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((school_state_one_hot_train,categories_one_hot_train,subcategories_one_ho
t_train,
project_grade_category_one_hot_train,teacher_prefix_one_hot_train,price_train,quantity_
train,
previously_posted_projects_train,project_essay_BOW_train,project_title_BOW_train)).tocs
r()

X_cr = hstack((school_state_one_hot_cv,categories_one_hot_cv,subcategories_one_hot_cv,
project_grade_category_one_hot_cv,teacher_prefix_one_hot_cv,price_cv,quantity_cv,
previously_posted_projects_cv,project_essay_BOW_cv,project_title_BOW_cv)).tocsr()

X_te = hstack((school_state_one_hot_test,categories_one_hot_test,subcategories_one_hot_
test,
project_grade_category_one_hot_test,teacher_prefix_one_hot_test,price_test,quantity_tes
t,
previously_posted_projects_test,project_essay_BOW_test,project_title_BOW_test)).tocsr()



print("The 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('='*50)
```

```
The Final Data Matrix :----
(22445, 10246) (22445,)
(11055, 10246) (11055,)
(16500, 10246) (16500,)
==================================================
```

i) Finding the best hyper parameter which results in the maximum AUC value :

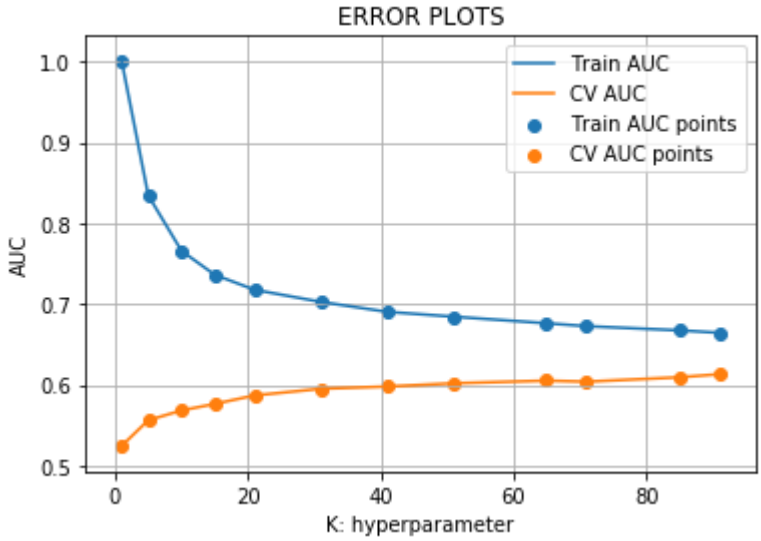
In [109]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
 of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 =
 49000
    # in this for loop we will iterate untill the last 1000 multiplier
    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 [112]:

```
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 va
lues, or non-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, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]
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 positive 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()
```

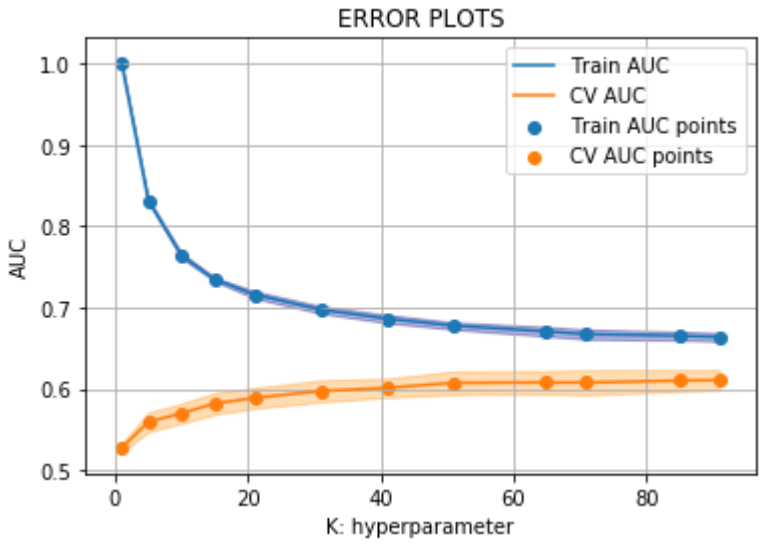
```
100%|██████████| 12/12 [1:13:41<00:00, 350.18s/it]
```



ii) GridsearchCV

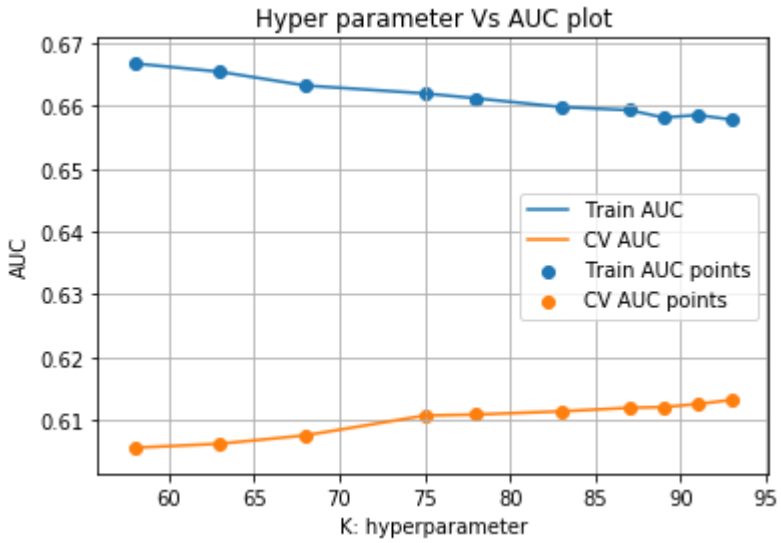
In [111]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

neigh = KNeighborsClassifier()
parameters = {'n_neighbors':[1, 5, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]}
clf = GridSearchCV(neigh, parameters, cv= 5, scoring='roc_auc')
clf.fit(X_tr, y_train)
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.3,color='darkblue')
plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc + cv_auc_st
d,alpha=0.3,
color='darkorange')
plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



In [95]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV

neigh = KNeighborsClassifier(n_jobs=-1)
parameters = {'n_neighbors':sp_randint(50, 100)}
clf = RandomizedSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
clf.fit(X_tr, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_n_neighbors'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_n_neighbors']

plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='darkblue')

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='darkorange')

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("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



Out[95]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors
4	0.083334	0.007364	23.593047	1.138842	58
5	0.088543	0.014732	25.150839	1.792233	63
1	0.119781	0.029453	15.114131	0.978608	68
8	0.108859	0.021368	29.011097	2.329769	75
0	8.807448	7.011351	129.640104	26.567274	78

In [98]:

```
results.head(30)
```

Out[98]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors
4	0.083334	0.007364	23.593047	1.138842	58
5	0.088543	0.014732	25.150839	1.792233	63
1	0.119781	0.029453	15.114131	0.978608	68
8	0.108859	0.021368	29.011097	2.329769	75
0	8.807448	7.011351	129.640104	26.567274	78
6	0.109375	0.025516	29.268212	3.147103	83
2	0.078124	0.000003	16.322430	0.820485	87
7	0.083333	0.007363	25.119042	2.828643	89
3	0.078124	0.000001	19.067139	1.971594	91
9	0.130209	0.007367	29.158710	1.158546	93

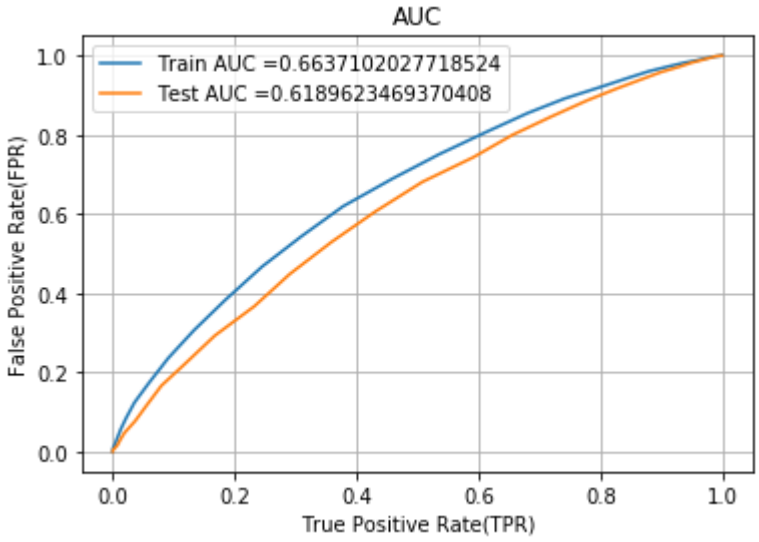
In [104]:

```
# We find that the best value for k is 93.
best_k_1 = 93
```

iii) Training the model using the best hyper parameter

In [105]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k_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.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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



iv) Confusion Matrix

In [119]:

```
# 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.rou
nd(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

a) Confusion matrix for Train Data

In [138]:

```
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_f
pr)))
```

```
====================================================================================
==========================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2474608644880757 for threshold 0.835
[[ 1906  1557]
 [ 5839 13143]]
```

In [126]:

```
df_train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred
, tr_thresholds,
train_fpr, train_fpr)))
```

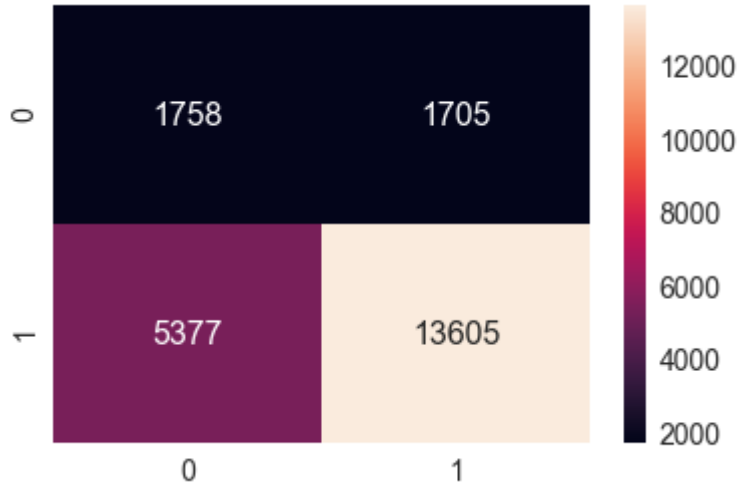
```
the maximum value of tpr*(1-fpr) 0.24878920920462003 for threshold 0.785
```

In [127]:

```
sns.set(font_scale=1.4)     #for label size
sns.heatmap(df_train_confusion_matrix, annot=True, annot_kws={"size": 16}, fmt='g')
```

Out[127]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c6775360f0>
```



b) Confusion Matrix for test data

In [129]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_fpr
)))
```

```
====================================================================================
==========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24993829175534654 for threshold 0.796
[[1253 1293]
 [4451 9503]]
```

In [130]:

```
df_test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, t
e_thresholds,
test_fpr, test_fpr)))
```

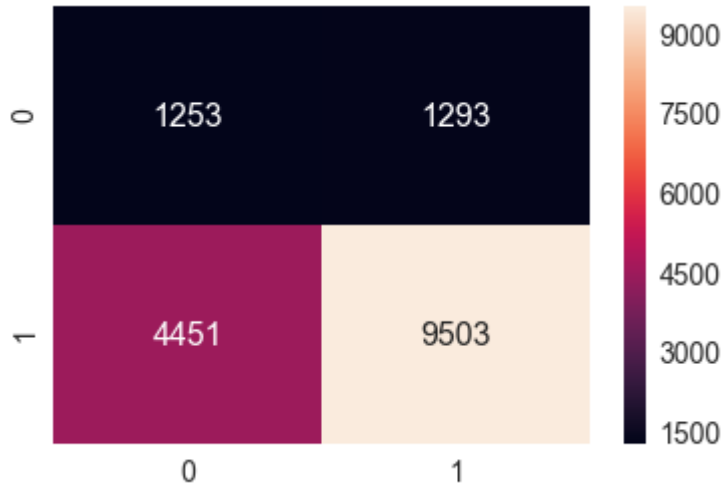
```
the maximum value of tpr*(1-fpr) 0.24993829175534654 for threshold 0.796
```

In [131]:

```
sns.set(font_scale=1.4)     #for label size
sns.heatmap(df_test_confusion_matrix, annot=True, annot_kws={"size": 16}, fmt='g')
```

Out[131]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c677428b38>
```



Set 2: categorical, numerical features + project_title(Tfidf) +preprocessed_essay (Tfidf)

In [132]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr = hstack((school_state_one_hot_train,categories_one_hot_train,subcategories_one_ho
t_train,
project_grade_category_one_hot_train,teacher_prefix_one_hot_train,price_train,quantity_
train,
previously_posted_projects_train,project_essay_tfidf_train,project_title_tfidf_train)).
tocsr()

X_cr = hstack((school_state_one_hot_cv,categories_one_hot_cv,subcategories_one_hot_cv,
project_grade_category_one_hot_cv,teacher_prefix_one_hot_cv,price_cv,quantity_cv,
previously_posted_projects_cv,project_essay_tfidf_cv,project_title_tfidf_cv)).tocsr()

X_te = hstack((school_state_one_hot_test,categories_one_hot_test,subcategories_one_hot_
test,
project_grade_category_one_hot_test,teacher_prefix_one_hot_test,price_test,quantity_tes
t,
previously_posted_projects_test,project_essay_tfidf_test,project_title_tfidf_test)).toc
sr()



print("The 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('='*50)
```

```
The Final Data Matrix :----
(22445, 10246) (22445,)
(11055, 10246) (11055,)
(16500, 10246) (16500,)
==================================================
```

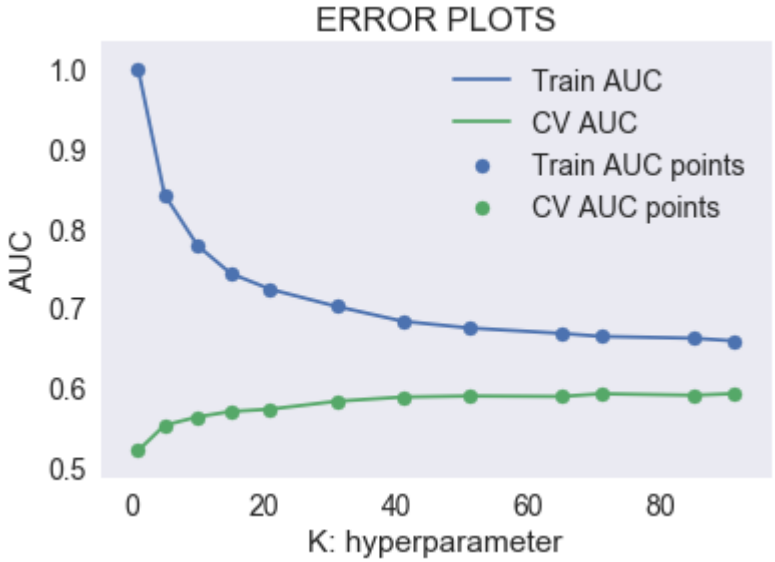
i) Finding the best hyper parameter which results in the maximum AUC value :

- Simple for loop

In [133]:

```
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]
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 positive 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()
```

```
100%|██████████| 12/12 [1:01:50<00:00, 358.03s/it]
```



In [136]:

```
# By looking at the plot we can say that, the best value of k is 85
best_k_2 = 85
```

ii) RandomizedSearchCV:

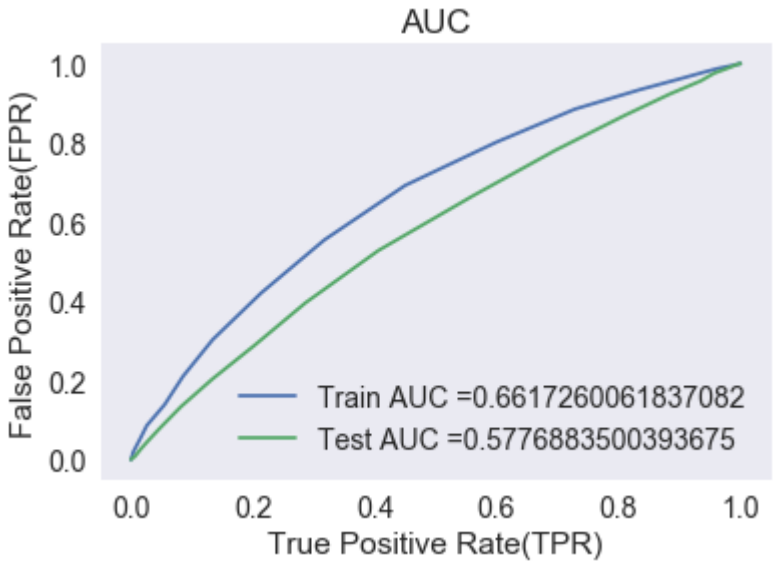
In [ ]:

```
# This will take much time to run, So not running this code.
#neigh = KNeighborsClassifier(n_jobs=-1)
#clf = RandomizedSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
#clf.fit(X_tr, y_train)

#results = pd.DataFrame.from_dict(clf.cv_results_)
#results = results.sort_values(['param_n_neighbors'])

#train_auc= results['mean_train_score']
#train_auc_std= results['std_train_score']
#cv_auc = results['mean_test_score']
#cv_auc_std= results['std_test_score']
#K =  results['param_n_neighbors']

#plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_std,alpha=
0.2,color='darkblue')

#plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2,color='da
rkorange')

#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("Hyper parameter Vs AUC plot")
#plt.grid()
#plt.show()
```

iii) Training the model using the best hyper parameter

In [137]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k_2)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he 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.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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



iv) Confusion Matrix

a) Confusion matrix for Train Data

In [139]:

```
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_f
pr)))
```

```
====================================================================================
==========================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2474608644880757 for threshold 0.835
[[ 1906  1557]
 [ 5839 13143]]
```

In [140]:

```
df_train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred
, tr_thresholds,
train_fpr, train_fpr)))
```

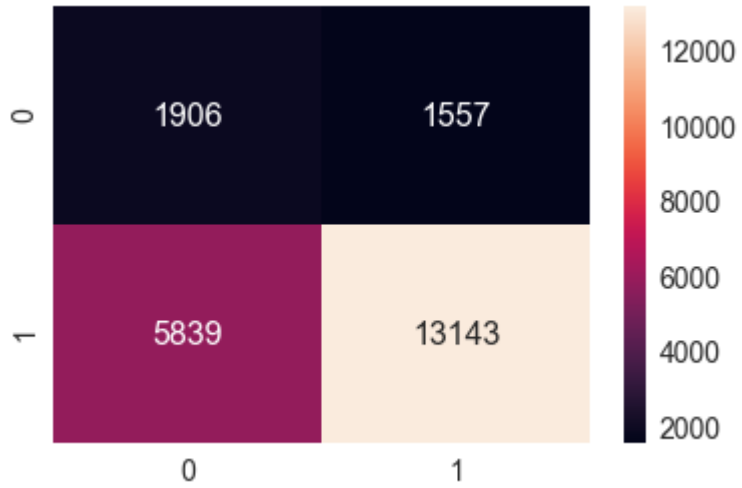
```
the maximum value of tpr*(1-fpr) 0.2474608644880757 for threshold 0.835
```

In [141]:

```
sns.set(font_scale=1.4)     #for label size
sns.heatmap(df_train_confusion_matrix, annot=True, annot_kws={"size": 16}, fmt='g')
```

Out[141]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c677616940>
```



b) Confusion Matrix for test data

In [142]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_fpr
)))
```

```
====================================================================================
==========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24638867925226887 for threshold 0.835
[[1120 1426]
 [4667 9287]]
```

In [143]:

```
df_test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, t
e_thresholds,
test_fpr, test_fpr)))
```

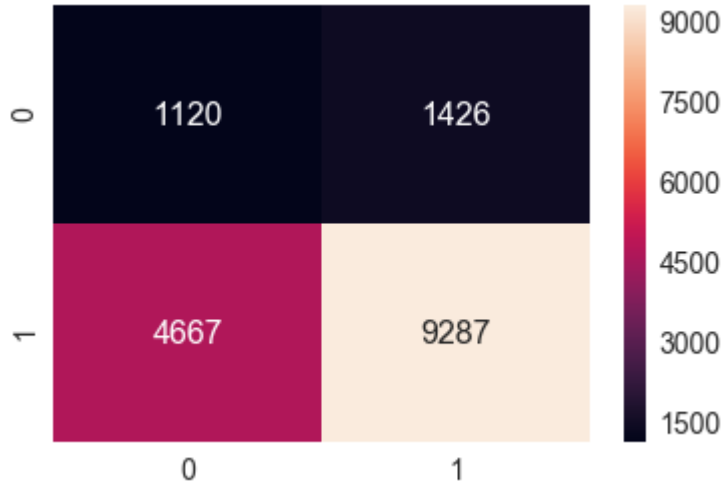
```
the maximum value of tpr*(1-fpr) 0.24638867925226887 for threshold 0.835
```

In [144]:

```
sns.set(font_scale=1.4)     #for label size
sns.heatmap(df_test_confusion_matrix, annot=True, annot_kws={"size": 16}, fmt='g')
```

Out[144]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c677b50240>
```



Set 3: categorical, numerical features + project_title(avg w2v) +preprocessed_essay (avg w2v)

In [145]:

```
X_tr = hstack((school_state_one_hot_train,categories_one_hot_train,subcategories_one_ho
t_train,
project_grade_category_one_hot_train,teacher_prefix_one_hot_train,price_train,quantity_
train,
previously_posted_projects_train,avg_w2v_vectors_essay_train,avg_w2v_vectors_title_trai
n)).tocsr()

X_cr = hstack((school_state_one_hot_cv,categories_one_hot_cv,subcategories_one_hot_cv,
project_grade_category_one_hot_cv,teacher_prefix_one_hot_cv,price_cv,quantity_cv,
previously_posted_projects_cv,avg_w2v_vectors_essay_cv,avg_w2v_vectors_title_cv)).tocsr
()

X_te = hstack((school_state_one_hot_test,categories_one_hot_test,subcategories_one_hot_
test,
project_grade_category_one_hot_test,teacher_prefix_one_hot_test,price_test,quantity_tes
t,
previously_posted_projects_test,avg_w2v_vectors_essay_test,avg_w2v_vectors_title_test))
.tocsr()



print("The 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('='*50)
```

```
The Final Data Matrix :----
(22445, 703) (22445,)
(11055, 703) (11055,)
(16500, 703) (16500,)
==================================================
```

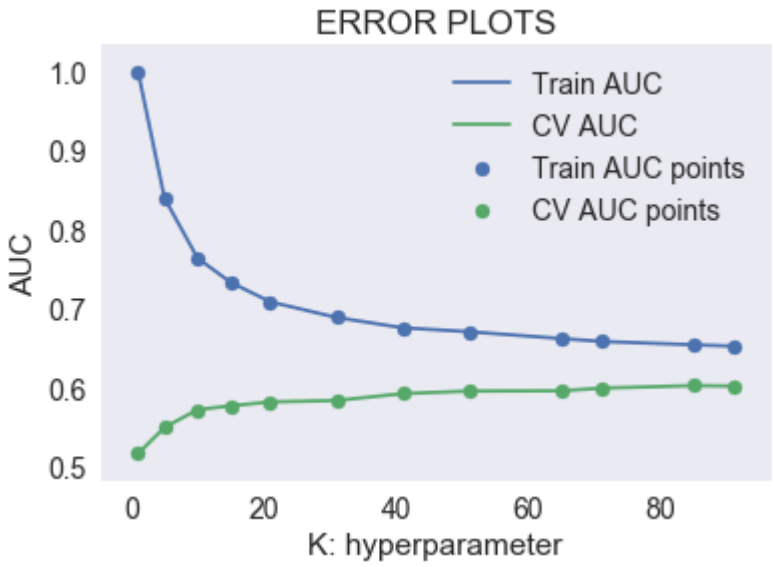
i) Finding the best hyper parameter which results in the maximum AUC value :

- simple for loop

In [146]:

```
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]
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 positive 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()
```

```
100%|██████████| 12/12 [4:32:14<00:00, 1224.08s/it]
```



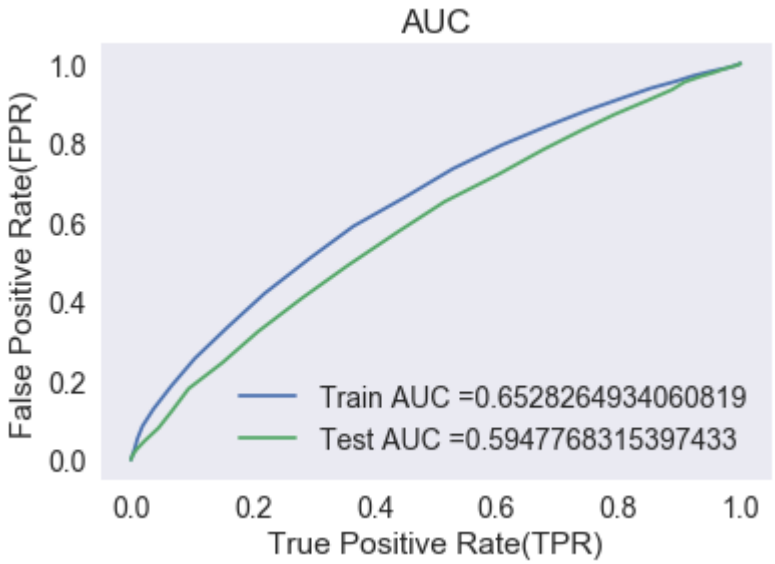
In [147]:

```
# we can assume the best k to be 91.
best_k_3 = 91
```

ii) Training the model using the best hyper parameter

In [149]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k_3)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he 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.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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



iii) Confusion Matrix

a) Confusion Matrix of train data

In [151]:

```
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_f
pr)))
```

```
====================================================================================
==========================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2491909646876276 for threshold 0.835
[[ 1633  1830]
 [ 5031 13951]]
```

In [152]:

```
df_train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred
, tr_thresholds,
train_fpr, train_fpr)))
```

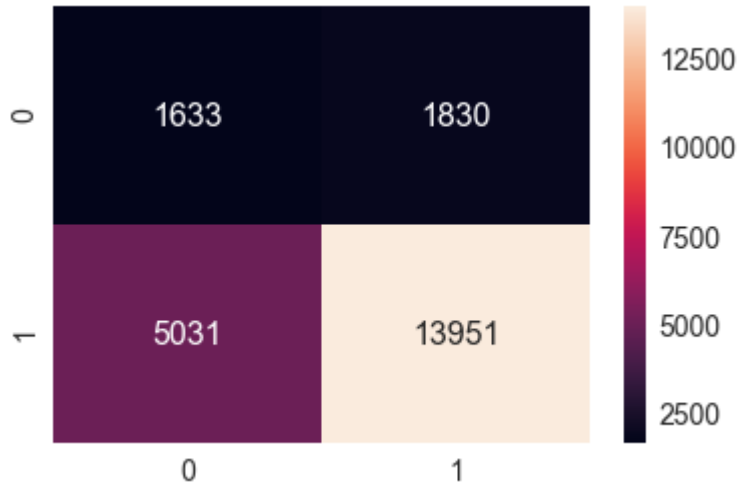
```
the maximum value of tpr*(1-fpr) 0.2491909646876276 for threshold 0.835
```

In [153]:

```
sns.set(font_scale=1.4)     #for label size
sns.heatmap(df_train_confusion_matrix, annot=True, annot_kws={"size": 16}, fmt='g')
```

Out[153]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c6773616d8>
```



b) Confusion Matrix of test data

In [154]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_fpr
)))
```

```
====================================================================================
==========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24978880353267358 for threshold 0.846
[[1236 1310]
 [4861 9093]]
```

In [155]:

```
df_test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, t
e_thresholds,
test_fpr, test_fpr)))
```

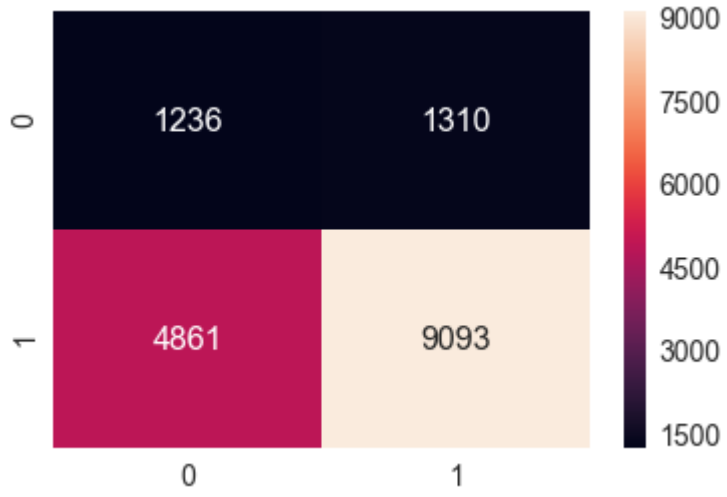
```
the maximum value of tpr*(1-fpr) 0.24978880353267358 for threshold 0.846
```

In [156]:

```
sns.set(font_scale=1.4)     #for label size
sns.heatmap(df_test_confusion_matrix, annot=True, annot_kws={"size": 16}, fmt='g')
```

Out[156]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c677be4710>
```



Set 4: categorical, numerical features + project_title(Tfidf w2v) +preprocessed_essay (Tfidf w2v)

In [157]:

```
X_tr = hstack((school_state_one_hot_train,categories_one_hot_train,subcategories_one_ho
t_train,
project_grade_category_one_hot_train,teacher_prefix_one_hot_train,price_train,quantity_
train,
previously_posted_projects_train,tfidf_w2v_project_essay_train,tfidf_w2v_project_title_
train)).tocsr()

X_cr = hstack((school_state_one_hot_cv,categories_one_hot_cv,subcategories_one_hot_cv,
project_grade_category_one_hot_cv,teacher_prefix_one_hot_cv,price_cv,quantity_cv,
previously_posted_projects_cv,tfidf_w2v_project_essay_cv,tfidf_w2v_project_title_cv)).t
ocsr()

X_te = hstack((school_state_one_hot_test,categories_one_hot_test,subcategories_one_hot_
test,
project_grade_category_one_hot_test,teacher_prefix_one_hot_test,price_test,quantity_tes
t,
previously_posted_projects_test,tfidf_w2v_project_essay_test,tfidf_w2v_project_title_te
st)).tocsr()



print("The 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('='*50)
```

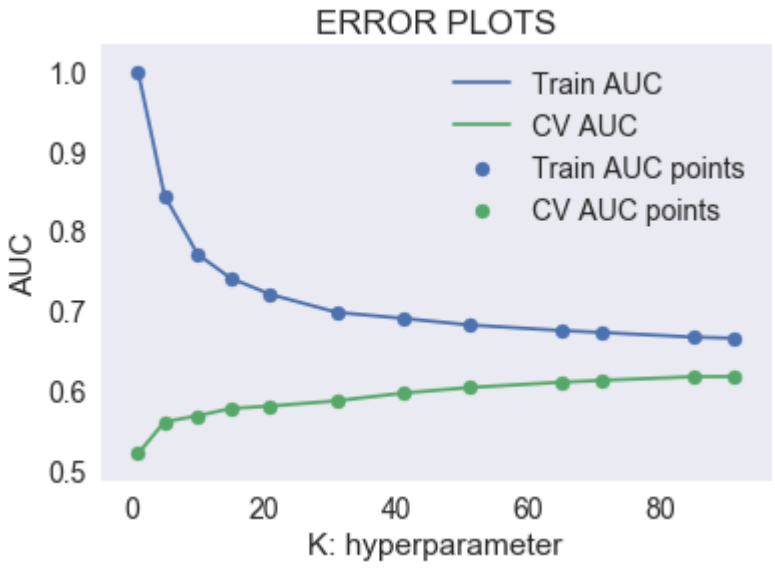
```
The Final Data Matrix :----
(22445, 703) (22445,)
(11055, 703) (11055,)
(16500, 703) (16500,)
==================================================
```

i) Finding the best hyper parameter which results in the maximum AUC value :

In [158]:

```
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]
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 positive 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()
```

```
100%|██████████| 12/12 [6:17:54<00:00, 1647.49s/it]
```



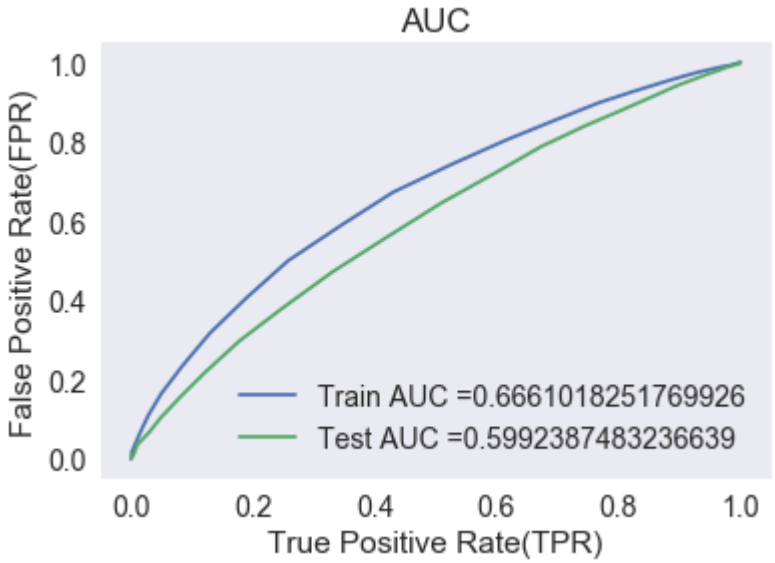
In [159]:

```
best_k_4 = 85
```

ii) Training the model using the best hyper parameter

In [160]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k_4)
neigh.fit(X_tr, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he 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.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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



iii) Confusion Matrix

a) Confusion Matrix of train data

In [161]:

```
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_f
pr)))
```

```
====================================================================================
==========================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24923949554921135 for threshold 0.835
[[ 1636  1827]
 [ 4854 14128]]
```

In [162]:

```
df_train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred
, tr_thresholds,
train_fpr, train_fpr)))
```

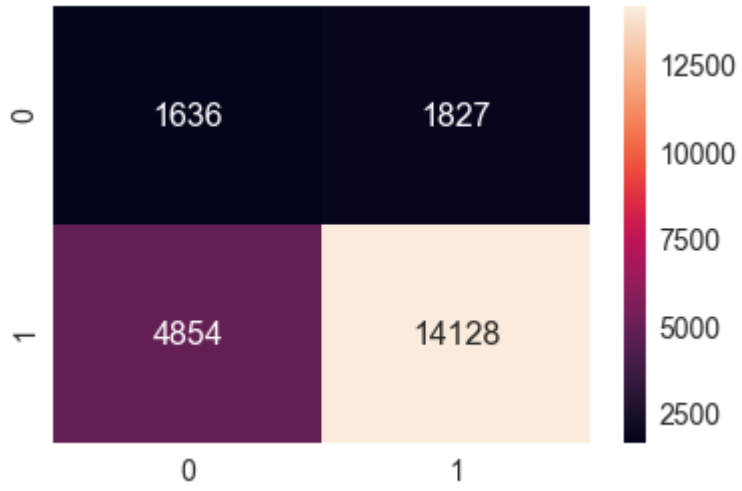
```
the maximum value of tpr*(1-fpr) 0.24923949554921135 for threshold 0.835
```

In [163]:

```
sns.set(font_scale=1.4)     #for label size
sns.heatmap(df_train_confusion_matrix, annot=True, annot_kws={"size": 16}, fmt='g')
```

Out[163]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c677ad61d0>
```



b) Confusion Matrix of test data

In [164]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_fpr
)))
```

```
====================================================================================
==========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24991114012769905 for threshold 0.847
[[1249 1297]
 [4942 9012]]
```

In [165]:

```
df_test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, t
e_thresholds,
test_fpr, test_fpr)))
```

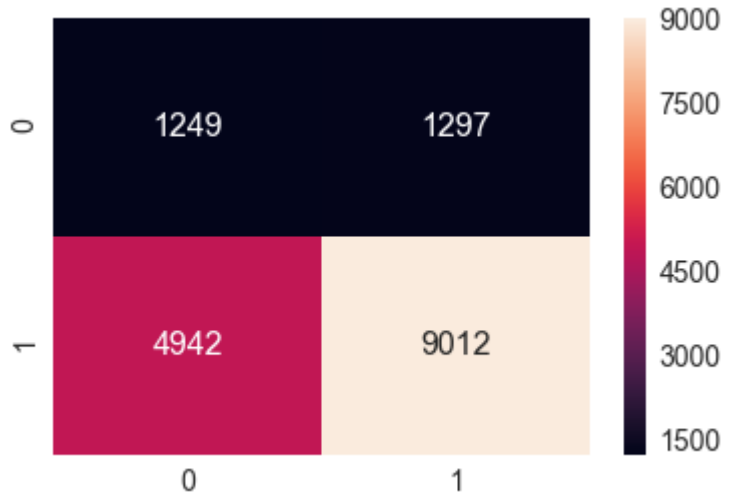
```
the maximum value of tpr*(1-fpr) 0.24991114012769905 for threshold 0.847
```

In [166]:

```
sns.set(font_scale=1.4)     #for label size
sns.heatmap(df_test_confusion_matrix, annot=True, annot_kws={"size": 16}, fmt='g')
```

Out[166]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c6773df278>
```



Task 2: Feature selection of Set 2 using selectkbest

In [168]:

```
X_tr = hstack((school_state_one_hot_train,categories_one_hot_train,subcategories_one_ho
t_train,
project_grade_category_one_hot_train,teacher_prefix_one_hot_train,price_train,quantity_
train,
previously_posted_projects_train,project_essay_tfidf_train,project_title_tfidf_train)).
tocsr()

X_cr = hstack((school_state_one_hot_cv,categories_one_hot_cv,subcategories_one_hot_cv,
project_grade_category_one_hot_cv,teacher_prefix_one_hot_cv,price_cv,quantity_cv,
previously_posted_projects_cv,project_essay_tfidf_cv,project_title_tfidf_cv)).tocsr()

X_te = hstack((school_state_one_hot_test,categories_one_hot_test,subcategories_one_hot_
test,
project_grade_category_one_hot_test,teacher_prefix_one_hot_test,price_test,quantity_tes
t,
previously_posted_projects_test,project_essay_tfidf_test,project_title_tfidf_test)).toc
sr()



print("The Data Matrix :----")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print('='*50)
```

```
The Data Matrix :----
(22445, 10246) (22445,)
(11055, 10246) (11055,)
(16500, 10246) (16500,)
==================================================
```

In [172]:

```
from sklearn.feature_selection import SelectKBest,chi2

X_tr_new = SelectKBest(chi2, k=2000).fit_transform(X_tr, y_train)
X_cr_new = SelectKBest(chi2, k=2000).fit_transform(X_cr, y_cv)
X_te_new = SelectKBest(chi2, k=2000).fit_transform(X_te, y_test)


print("The Final Data Matrix :----")
print(X_tr_new.shape, y_train.shape)
print(X_cr_new.shape, y_cv.shape)
print(X_te_new.shape, y_test.shape)
print('='*50)
```

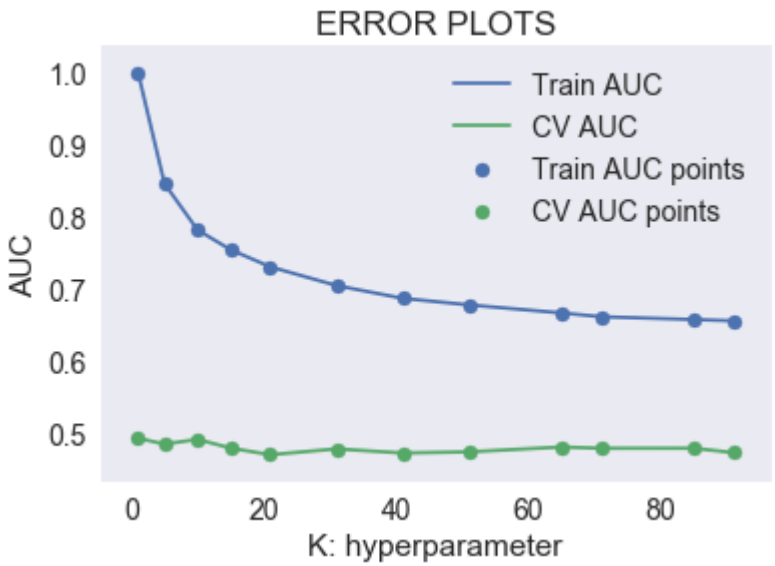
```
The Final Data Matrix :----
(22445, 2000) (22445,)
(11055, 2000) (11055,)
(16500, 2000) (16500,)
==================================================
```

i) Finding the best hyper parameter which results in the maximum AUC value :

In [173]:

```
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 65, 71, 85, 91]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    neigh.fit(X_tr_new, y_train)

    y_train_pred = batch_predict(neigh, X_tr_new)
    y_cv_pred = batch_predict(neigh, X_cr_new)

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
 of the positive 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()
```

```
100%|██████████| 12/12 [1:03:36<00:00, 347.67s/it]
```



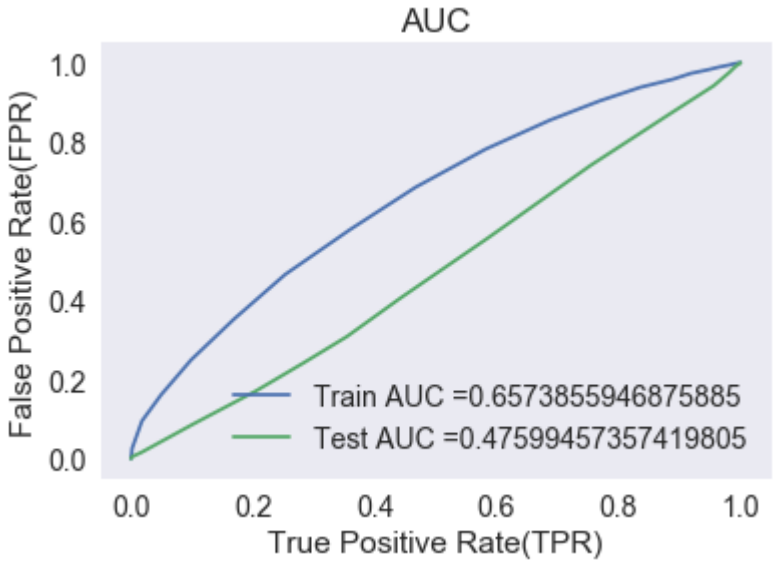
In [174]:

```
# By looking at the above curve we can say that k is nearly equal to 85
best_k_5 = 85
```

ii) Training the model using the best hyper parameter

In [184]:

```
neigh = KNeighborsClassifier(n_neighbors=best_k_5)
neigh.fit(X_tr_new, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_tr_new)
y_test_pred = batch_predict(neigh, X_te_new)

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

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



iii) Confusion Matrix

a) Confusion Matrix of train data

In [176]:

```
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_f
pr)))
```

```
====================================================================================
==========================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24890678397237445 for threshold 0.847
[[ 1846  1617]
 [ 5971 13011]]
```

In [177]:

```
df_train_confusion_matrix = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred
, tr_thresholds,
train_fpr, train_fpr)))
```

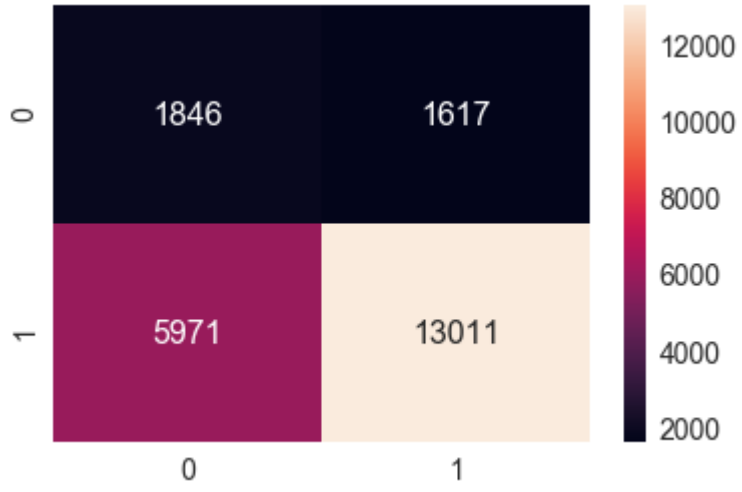
```
the maximum value of tpr*(1-fpr) 0.24890678397237445 for threshold 0.847
```

In [178]:

```
sns.set(font_scale=1.4)     #for label size
sns.heatmap(df_train_confusion_matrix, annot=True, annot_kws={"size": 16}, fmt='g')
```

Out[178]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c677b38c88>
```



b) Confusion Matrix of test data

In [179]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, te_thresholds, test_fpr, test_fpr
)))
```

```
====================================================================================
==========================
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24662085652277743 for threshold 0.859
[[1421 1125]
 [8297 5657]]
```

In [180]:

```
df_test_confusion_matrix = pd.DataFrame(confusion_matrix(y_test, predict(y_test_pred, t
e_thresholds,
test_fpr, test_fpr)))
```

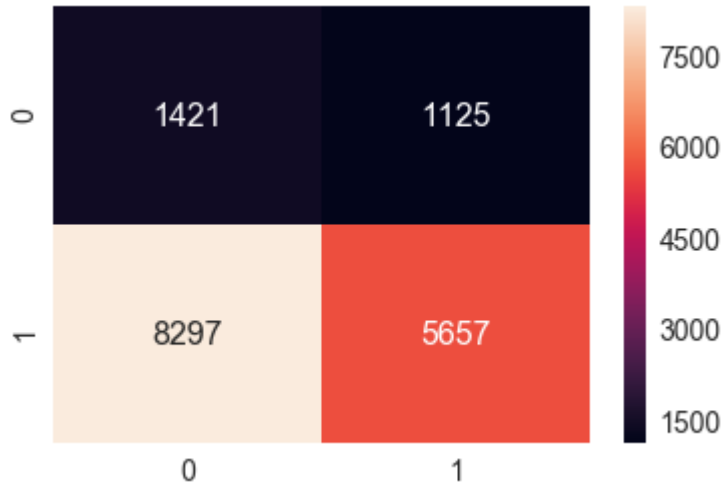
```
the maximum value of tpr*(1-fpr) 0.24662085652277743 for threshold 0.859
```

In [181]:

```
sns.set(font_scale=1.4)     #for label size
sns.heatmap(df_test_confusion_matrix, annot=True, annot_kws={"size": 16}, fmt='g')
```

Out[181]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c677b39828>
```



Conclusions :

In [185]:

```
# Constructing Prettytable to compare all the performance measures.
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prett
ytable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]
x.add_row(["BOW", "Brute", 93, 0.66])
x.add_row(["TFIDF", "Brute", 85, 0.57])
x.add_row(["AVG W2V", "Brute", 91, 0.59])
x.add_row(["TFIDF W2V", "Brute", 85, 0.59])
x.add_row(["TFIDF", "Top 2000", 85, 0.47])

print(x)
```

```
+------------+----------+-----------------+------+
| Vectorizer |  Model   | Hyper Parameter | AUC  |
+------------+----------+-----------------+------+
|    BOW     |  Brute   |        93       | 0.66 |
|   TFIDF    |  Brute   |        85       | 0.57 |
|  AVG W2V   |  Brute   |        91       | 0.59 |
| TFIDF W2V  |  Brute   |        85       | 0.59 |
|   TFIDF    | Top 2000 |        85       | 0.47 |
+------------+----------+-----------------+------+
```