DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

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

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:
```
#Importing essential libraries & packages
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [2]:
```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [3]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:
```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[4]:

	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

Preprocessing Categorical Data

1.2 preprocessing of `Teacher prefix`

In [5]:
```
# check if we have any nan values are there in the column
print(project_data['teacher_prefix'].isnull().values.any())
print("number of nan values",project_data['teacher_prefix'].isnull().values.sum())
```

```
True
number of nan values 3
```

In [6]:
```
#Replacing the Nan values with most frequent value in the column
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna('Mrs.')
```

In [7]:
```
#Converting teacher prefix text into smaller case
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.lower()
project_data['teacher_prefix'].value_counts()
```

Out[7]:
```
mrs.       57272
ms.        38955
mr.        10648
teacher     2360
dr.           13
Name: teacher_prefix, dtype: int64
```

1.3 preprocessing of `project_subject_categories`

In [8]:
```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

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

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

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

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

1.4 preprocessing of project_subject_subcategories

In [9]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

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

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

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

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

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

1.5 Preprocessing project grade category

In [10]:
```
#Replacing spaces & hyphens in the text of project grade category with underscore
#converting Capital letters in the string to smaller letters
#Performing avalue count of project grade category
# https://stackoverflow.com/questions/36383821/pandas-dataframe-apply-function-to-column-strings-based-on-
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace(' ','_')
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace('-','_')
project_data['project_grade_category'] = project_data['project_grade_category'].str.lower()
project_data['project_grade_category'].value_counts()
```

Out[10]:
```
grades_prek_2    44225
grades_3_5       37137
grades_6_8       16923
grades_9_12      10963
Name: project_grade_category, dtype: int64
```

1.6 Combining 4 essays column into one

In [11]:
```
# 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)
```

1.7 Test Train split

In [12]:
```
data = project_data[:50001]
data.head(5)
```

Out[12]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	mrs.	IN	2016-12-05 13:43:57	grade
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	mr.	FL	2016-10-25 09:22:10	gr
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	ms.	AZ	2016-08-31 12:03:56	gr
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	mrs.	KY	2016-10-06 21:16:17	grade
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	mrs.	TX	2016-07-11 01:10:09	grade

In [13]:
```
y = data['project_is_approved'].values
data.drop(['project_is_approved'], axis=1, inplace=True)
data.head(1)
```

Out[13]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_c
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	mrs.	IN	2016-12-05 13:43:57	grades

In [14]:
```
X = data
```

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

In [15]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
#Splitting data into test & train set
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
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)
```

In [16]:
```
#Splitting training data into training & cross validation sets
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,
                                                stratify= y_train,
                                                test_size = 0.33)
```

Text Preprocessing

In [17]:
```
# printing some random reviews
print(X_train['essay'].values[0])
print("="*50)
print(X_train['essay'].values[100])
print("="*50)
print(X_train['essay'].values[300])
print("="*50)
print(X_train['essay'].values[5000])
print("="*50)
print(X_train['essay'].values[20000])
print("="*50)
```

```
My students come in the classroom everyday eager to learn. This is my second year teaching and first y
ear teaching second grade. My students are flexible learners and love when we try new things. \r\n\r\n
One of the things I love most about my students is that they are always eager to push themselves and a
chieve new goals. My goal for the classroom is to make and keep learning fun. I want to inspire my stu
dents to be creative, flexible, problem solvers, and excited about gaining knowledge. \r\n\r\nI cannot
express in these 2 to 3 paragraphs the potential all of my students possess. They deserve every resour
ce I can possibly offer them. I am excited to utilize Donors Choose to help build and improve their le
arning experience and environment!I am a firm believer that students need to be in control of their le
arning. They truly value the work and effort that goes into learning if they feel like they are a part
of making it happen. \r\n\r\nMy goal is to encourage student collaboration, critical thinking, communi
cation, and creative thinking. I have researched flexible seating and believe that my students would g
reatly benefit from the type of atmosphere it would create in my classroom.\r\n\r\nRecently, I have ch
anged up my classroom seating the best I can but do not have the supplies I need to fully implement fl
exible seating. The supplies I have asked for will allow students to choose where and how they learn b
est.\r\n\r\nnannan
==================================================
We are a very energetic and motivated group of third graders, determined to get the most out of each s
chool day.  We have learned to work hard to reap the rewards of success in our learning, and it shows
in our busy classroom.  If you walk into our classroom on any given day, you will see students activel
```

In [18]:
```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [19]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays, cognitive del
ays, gross/fine motor delays, to autism. They are eager beavers and always strive to work their hardest
working past their limitations. \r\n\r\nThe materials we have are the ones I seek out for my students. I
teach in a Title I school where most of the students receive free or reduced price lunch.  Despite their
disabilities and limitations, my students love coming to school and come eager to learn and explore.Have
you ever felt like you had ants in your pants and you needed to groove and move as you were in a meetin
g? This is how my kids feel all the time. The want to be able to move as they learn or so they say.Wobbl
e chairs are the answer and I love then because they develop their core, which enhances gross motor and
in Turn fine motor skills. \r\nThey also want to learn through games, my kids do not want to sit and do
worksheets. They want to learn to count by jumping and playing. Physical engagement is the key to our su
ccess. The number toss and color and shape mats can make that happen. My students will forget they are d
oing work and just have the fun a 6 year old deserves.nannan
==================================================
```

In [20]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays, cognitive del
ays, gross/fine motor delays, to autism. They are eager beavers and always strive to work their hardest
working past their limitations.     The materials we have are the ones I seek out for my students. I tea
ch in a Title I school where most of the students receive free or reduced price lunch.  Despite their di
sabilities and limitations, my students love coming to school and come eager to learn and explore.Have y
ou ever felt like you had ants in your pants and you needed to groove and move as you were in a meeting?
This is how my kids feel all the time. The want to be able to move as they learn or so they say.Wobble c
hairs are the answer and I love then because they develop their core, which enhances gross motor and in
Turn fine motor skills.   They also want to learn through games, my kids do not want to sit and do works
heets. They want to learn to count by jumping and playing. Physical engagement is the key to our succes
s. The number toss and color and shape mats can make that happen. My students will forget they are doing
work and just have the fun a 6 year old deserves.nannan
```

In [21]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and language delays cognitive dela
ys gross fine motor delays to autism They are eager beavers and always strive to work their hardest work
ing past their limitations The materials we have are the ones I seek out for my students I teach in a Ti
tle I school where most of the students receive free or reduced price lunch Despite their disabilities a
nd limitations my students love coming to school and come eager to learn and explore Have you ever felt
like you had ants in your pants and you needed to groove and move as you were in a meeting This is how m
y kids feel all the time The want to be able to move as they learn or so they say Wobble chairs are the
answer and I love then because they develop their core which enhances gross motor and in Turn fine motor
skills They also want to learn through games my kids do not want to sit and do worksheets They want to l
earn to count by jumping and playing Physical engagement is the key to our success The number toss and c
olor and shape mats can make that happen My students will forget they are doing work and just have the f
un a 6 year old deserves nannan
```

In [22]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'thei
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 't
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while',
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'agai
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't",
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'n
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'we
            'won', "won't", 'wouldn', "wouldn't"]
```

Preprocessing for Train Data

In [23]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays_xtr = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays_xtr.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████| 22445/22445 [00:14<00:0
0, 1527.14it/s]
```

In [24]:
```
# after preprocesing
preprocessed_essays_xtr[300]
```

Out[24]:
```
'students not hungry hungry learn create daily basis title school students receive free breakfast lunch
daily basis students face many challenges many students families not afford basic school supplies childr
en pencils paper even materials needed complete classroom reports despite challenges eager ready learn s
tudents struggles home job make sure able accomplish wish school one thing would like year give students
families much support providing basic supplies many not able bring school building toys simple plaything
s deluxe logs classic teach basic engineering skills students design bring life objects imagination zoob
s offer imaginative creative way build students follow blue print create object magformers teach student
s design construction students opportunity use materials throughout year finishing activity students fre
e choice complete steam activity activities promote collaboration creativity students offering steam ite
ms classroom give students opportunity apply many key principles fields study children learn learn makin
g mistakes solving problems play knows might inspiring future architect scientist nannan'
```

In [25]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays_xcv = []
# tqdm is for printing the status bar
for sentance in tqdm(X_cv['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays_xcv.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████| 11055/11055 [00:07<00:0
0, 1545.37it/s]
```

In [26]:
```
# after preprocesing
preprocessed_essays_xcv[300]
```

Out[26]:
```
'school bell rings hear excited voices head hall classroom 5th grade hall way filled 125 students high e
nergy enthusiastic full personality teach title 1 school 75 students receive free reduced lunch come div
erse backgrounds families come united states learning english others parents jail living extended family
members one thing certain want learn active throughout day students love move daily ask use options flex
ible seating room dance sing lessons times need seated table desk students begging hokki stools stabilit
y balls use seated able move learning increases engagement focus retention material traditional classroo
ms many students required sit desk 4 6 hours day hokki stools stability balls allow students sit still m
ove students choice seats classroom move different places pillows floor working according needs need bou
nce students may choose stability ball hokki stools allow sway involuntary movement calms many students
allows use energy focus school work instead need fidget get seat school day not mean sit still options m
ovement throughout day increases brain activity keeps minds engaged learning added health benefits worki
ng core muscles creating stronger healthier kids engaging brains well asked feel seating options class s
tudents replied using ball sit keeps trouble not seat concentrate work hokki stool best wiggle not fall
chair please get stools balls need nannan'
```

In [27]:
```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays_xte = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays_xte.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████| 16501/16501 [00:10<00:0
0, 1584.65it/s]
```

In [28]:
```
# after preprocesing
preprocessed_essays_xte[300]
```

Out[28]:
```
'students mostly highly motivated perform well school also face high pressure perform highest level ther
efore find stressed overwhelmed max teach 31 students block students 18 weeks teaching class size challe
nging especially attempting track individual student progress standard goal create environment classroom
gets away one size fits approach desks rows teacher front rather rotates students around room student ge
ts one one time day not waste time group activities might not even helpful e reading around room providi
ng flexible seating tables chairs couches classroom engage students various activities complete centers
day one center reading one center technology research one center writing collaboration additional center
small group instruction horseshoe desk rectangular table also provide way spend one one time students gi
ve individualize instruction rest class comfortable space diligent focused desks overbearing difficult m
aneuver tables allow students relax enjoy learning process nannan'
```

1.4 Preprocessing of `project_title`

In [29]:
```
#printing random titles
print(data['project_title'].values[49])
print("="*50)
print(data['project_title'].values[89])
print("="*50)
print(data['project_title'].values[999])
print("="*50)
print(data['project_title'].values[11156])
print("="*50)
print(data['project_title'].values[20000])
print("="*50)
```

```
Rainy Day Run Around!
==================================================
Education Through Technology
==================================================
Focus Pocus
==================================================
Making Math Interactive!
==================================================
We Need To Move It While We Input It!
==================================================
```

In [30]:
```
preprocessed_titles_xtr = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles_xtr.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████| 22445/22445 [00:00<00:0
0, 28921.74it/s]
```

In [31]:
```
print(preprocessed_titles_xtr[89])
print("="*50)
```

```
reading with support
==================================================
```

In [32]:
```
preprocessed_titles_xcv = []
# tqdm is for printing the status bar
for sentance in tqdm(X_cv['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles_xcv.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████| 11055/11055 [00:00<00:0
0, 28343.01it/s]
```

In [33]:
```
print(preprocessed_titles_xcv[89])
print("="*50)
```

```
technology help learning science math
==================================================
```

In [34]:
```
preprocessed_titles_xte = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles_xte.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████| 16501/16501 [00:00<00:0
0, 27730.60it/s]
```

In [35]:
```
print(preprocessed_titles_xte[89])
print("="*50)
```

```
sounds like music ears
==================================================
```

1.5 Preparing data for models

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

2.2.1 One hot encoding categorical feature : State

In [36]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

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

#We use fit only for train data
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

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

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*75)
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16501, 51) (16501,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks',
'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'n
y', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
===========================================================================
```

2.2.2 One hot encoding the categorical features : teacher_prefix

In [37]:
```
#We use fit only for train data
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

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

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*50)
```

```
After vectorizations
(22445, 5) (22445,)
(11055, 5) (11055,)
(16501, 5) (16501,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
==================================================
```

2.2.3 One hot encoding the categorical features : grades

In [38]:
```
#We use fit only for train data
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

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

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*70)
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16501, 4) (16501,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
======================================================================
```

2.2.4 One hot encoding the categorical features : project subject category

In [39]:
```

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_cat_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_cat_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_cat_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_cat_ohe.shape, y_train.shape)
print(X_cv_cat_ohe.shape, y_cv.shape)
print(X_test_cat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*70)
```

```
After vectorizations
(22445, 9) (22445,)
(11055, 9) (11055,)
(16501, 9) (16501,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_scienc
e', 'music_arts', 'specialneeds', 'warmth']
======================================================================
```

2.2.5 One hot encoding the categorical features : project subject sub-category

In [40]:
```
#We use fit only for train data
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcat_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_subcat_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcat_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subcat_ohe.shape, y_train.shape)
print(X_cv_subcat_ohe.shape, y_cv.shape)
print(X_test_subcat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*70)
```

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16501, 30) (16501,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'com
munityservice', 'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'fina
ncialliteracy', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geo
graphy', 'literacy', 'literature_writing', 'mathematics', 'music', 'nutritioneducation', 'other', 'paren
tinvolvement', 'performingarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
======================================================================
```

In [41]:
```
project_data.columns
```

Out[41]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', '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_approved',
       'clean_categories', 'clean_subcategories', 'essay'],
      dtype='object')
```

1.5.2 Vectorizing Text data

i) BoW encoding

1.5.2.1 Bag of words on Essay feature

In [42]:
```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
```

In [43]:
```
#Applying BoW on essays feature
#Considering only the words which appear atleast in 10 documents or reviews
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)



from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(preprocessed_essays_xtr) # fiting only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(preprocessed_essays_xtr)
X_cv_essay_bow = vectorizer.transform(preprocessed_essays_xcv)
X_test_essay_bow = vectorizer.transform(preprocessed_essays_xte)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
```

```
(22445, 17) (22445,)
(11055, 17) (11055,)
(16501, 17) (16501,)
====================================================================================================
After vectorizations
(22445, 8789) (22445,)
(11055, 8789) (11055,)
(16501, 8789) (16501,)
====================================================================================================
```

1.5.2.2 Bag of words on Project Title feature

In [44]:
```
#Applying BoW on project titles feature
#Considering only the words which appear atleast in 10 documents or reviews
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)



from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(preprocessed_titles_xtr) # fiting only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_bow = vectorizer.transform(preprocessed_titles_xtr)
X_cv_titles_bow = vectorizer.transform(preprocessed_titles_xcv)
X_test_titles_bow = vectorizer.transform(preprocessed_titles_xte)

print("After vectorizations")
print(X_train_titles_bow.shape, y_train.shape)
print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)
print("="*100)
```

```
(22445, 17) (22445,)
(11055, 17) (11055,)
(16501, 17) (16501,)
====================================================================================================
After vectorizations
(22445, 1236) (22445,)
(11055, 1236) (11055,)
(16501, 1236) (16501,)
====================================================================================================
```

ii) TFIDF Vectorization

TFIDF on Essay feature

In [45]:
```
#Applying TF-IDF on essays feature
#Considering only the words which appear atleast in 10 documents or reviews
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)



from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(preprocessed_essays_xtr) # fiting only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(preprocessed_essays_xtr)
X_cv_essay_tfidf = vectorizer.transform(preprocessed_essays_xcv)
X_test_essay_tfidf = vectorizer.transform(preprocessed_essays_xte)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
(22445, 17) (22445,)
(11055, 17) (11055,)
(16501, 17) (16501,)
====================================================================================================
After vectorizations
(22445, 8789) (22445,)
(11055, 8789) (11055,)
(16501, 8789) (16501,)
====================================================================================================
```

TFIDF on Project Title feature

In [46]:
```
#Applying Tfidf on project titles feature
#Considering only the words which appear atleast in 10 documents or reviews
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)


from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(preprocessed_titles_xtr) # fiting only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_tfidf = vectorizer.transform(preprocessed_titles_xtr)
X_cv_titles_tfidf = vectorizer.transform(preprocessed_titles_xcv)
X_test_titles_tfidf = vectorizer.transform(preprocessed_titles_xte)

print("After vectorizations")
print(X_train_titles_tfidf.shape, y_train.shape)
print(X_cv_titles_tfidf.shape, y_cv.shape)
print(X_test_titles_tfidf.shape, y_test.shape)
print("="*100)
```

```
(22445, 17) (22445,)
(11055, 17) (11055,)
(16501, 17) (16501,)
====================================================================================================
After vectorizations
(22445, 1236) (22445,)
(11055, 1236) (11055,)
(16501, 1236) (16501,)
====================================================================================================
```

iii) Using Pretrained Models : AvgW2V

In [47]:
```
# 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 [48]:
```
model = loadGloveModel('glove.42B.300d.txt')
```

```
Loading Glove Model

1917495it [06:37, 4829.01it/s]

Done. 1917495  words loaded!
```

In [49]:
```
words = []
for i in preprocessed_essays_xtr:
    words.extend(i.split(' '))

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

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

```
all the words in the corpus 3192824
the unique words in the corpus 31521
The number of words that are present in both glove vectors and our corpus 29595 ( 93.89 %)
```

In [50]:
```
words_corpus_preprocessed_essays_xtr = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_corpus_preprocessed_essays_xtr[i] = model[i]
print("word 2 vec length", len(words_corpus_preprocessed_essays_xtr))
```

```
word 2 vec length 29595
```

In [51]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-l

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

In [52]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-l

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

Applying to Train set for Essay feature

In [53]:
```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_extr = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_xtr): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_extr.append(vector)

print(len(avg_w2v_vectors_extr))
print(len(avg_w2v_vectors_extr[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 22445/22445 [00:10<00:0
0, 2096.74it/s]

22445
300
```

Applying to Cross validation set for Essay feature

In [54]:
```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_excv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_xcv): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_excv.append(vector)

print(len(avg_w2v_vectors_excv))
print(len(avg_w2v_vectors_excv[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 11055/11055 [00:09<00:0
0, 1163.19it/s]

11055
300
```

Applying to test set for Essay feature

In [55]:
```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_exte = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_xte): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_exte.append(vector)

print(len(avg_w2v_vectors_exte))
print(len(avg_w2v_vectors_exte[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 16501/16501 [00:09<00:0
0, 1713.40it/s]

16501
300
```

Applying to Train set for Project title feature

In [56]:
```
# Vectorizing project_title using avgw2v method
avg_w2v_vectors_txtr = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_xtr): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_txtr.append(vector)

print(len(avg_w2v_vectors_txtr))
print(len(avg_w2v_vectors_txtr[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 22445/22445 [00:01<00:0
0, 15948.85it/s]

22445
300
```

Applying to Cross validation set for Project title feature

In [57]:
```
# Vectorizing project_title using avgw2v method
avg_w2v_vectors_txcv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_xcv): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_txcv.append(vector)

print(len(avg_w2v_vectors_txcv))
print(len(avg_w2v_vectors_txcv[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 11055/11055 [00:00<00:0
0, 19505.44it/s]

11055
300
```

In [58]:
```
# Vectorizing project_title using avgw2v method
avg_w2v_vectors_txte = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_xte): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_txte.append(vector)

print(len(avg_w2v_vectors_txte))
print(len(avg_w2v_vectors_txte[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 16501/16501 [00:00<00:0
0, 19642.40it/s]

16501
300
```

iv) Using Pretrained Models: TFIDF weighted W2V

Applying on Training set of essays feature

In [59]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays_xtr)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [60]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_extr = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_xtr): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/l
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf va
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_extr.append(vector)

print(len(tfidf_w2v_vectors_extr))
print(len(tfidf_w2v_vectors_extr[0]))
```

```
100%|████████████████████████████████████████████████████████████| 22445/22445 [01:31<00:
00, 246.08it/s]

22445
300
```

Applying on Cross validation set of essays feature

In [61]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_excv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_xcv): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/l
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf va
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_excv.append(vector)

print(len(tfidf_w2v_vectors_excv))
print(len(tfidf_w2v_vectors_excv[0]))
```

```
100%|████████████████████████████████████████████████████████████| 11055/11055 [00:45<00:
00, 240.83it/s]

11055
300
```

Applying on test set of essays feature

In [62]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_exte = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_xte): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/l
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf val
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_exte.append(vector)

print(len(tfidf_w2v_vectors_exte))
print(len(tfidf_w2v_vectors_exte[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 16501/16501 [01:10<00:
00, 232.50it/s]

16501
300
```

Applying on Training set of project title feature

In [63]:
```
# Similarly you can vectorize for title also
# vectorizing project_title using TFIDF weighted W2V pretrained model
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_titles_xtr)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [64]:
```
#compute tfidf w2v for project titles in train set

tfidf_w2v_vectors_txtr = []; # the avg-w2v for each sentence/review is stored in this list

for sentence in tqdm(preprocessed_titles_xtr): # for each review/sentence in Xtrain
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words and (word in tfidf_words):
            vector = model[word]

            tf_idf = dictionary[word]*(sentance.count(word)/len(sentance.split()))
            vector += (vector * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_txtr.append(vector)

print(len(tfidf_w2v_vectors_txtr))
print(len(tfidf_w2v_vectors_txtr[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 22445/22445 [00:00<00:0
0, 22802.50it/s]

22445
300
```

Applying on Cross validation set of project title feature

In [65]:
```
#compute tfidf w2v for project titles in cv set

tfidf_w2v_vectors_txcv = []; # the avg-w2v for each sentence/review is stored in this list

for sentence in tqdm(preprocessed_titles_xcv): # for each review/sentence in Xtrain
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words and (word in tfidf_words):
            vector = model[word]

            tf_idf = dictionary[word]*(sentance.count(word)/len(sentance.split()))
            vector += (vector * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_txcv.append(vector)

print(len(tfidf_w2v_vectors_txcv))
print(len(tfidf_w2v_vectors_txcv[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 11055/11055 [00:00<00:0
0, 20470.56it/s]

11055
300
```

Applying on Cross test set of project title feature

In [66]:
```
#compute tfidf w2v for project titles in test set

tfidf_w2v_vectors_txte = []; # the avg-w2v for each sentence/review is stored in this list

for sentence in tqdm(preprocessed_titles_xte): # for each review/sentence in Xtrain
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words and (word in tfidf_words):
            vector = model[word]

            tf_idf = dictionary[word]*(sentance.count(word)/len(sentance.split()))
            vector += (vector * tf_idf)
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_txte.append(vector)

print(len(tfidf_w2v_vectors_txte))
print(len(tfidf_w2v_vectors_txte[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 16501/16501 [00:00<00:0
0, 16800.11it/s]

16501
300
```

1.5.3 Vectorizing Numerical features

For Price

In [67]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [68]:
```
# join two dataframes in python:
X_train = pd.merge(X_train, price_data, on='id', how='left')
X_test = pd.merge(X_test, price_data, on='id', how='left')
X_cv = pd.merge(X_cv, price_data, on='id', how='left')
```

In [69]:
```
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.
normalizer.fit(X_train['price'].values.reshape(-1,1))

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

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16501, 1) (16501,)
====================================================================================================
```

For Quantity

In [70]:
```
#Normalizing quantity
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.
normalizer.fit(X_train['quantity'].values.reshape(-1,1))

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

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_cv_quantity_norm.shape, y_cv.shape)
print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16501, 1) (16501,)
====================================================================================================
```

In [71]:
```
# Normalizing teacher previously posted projects
#Normalizing quantity
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].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.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_tpp_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.res
X_cv_tpp_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(
X_test_tpp_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.resha

print("After vectorizations")
print(X_train_tpp_norm.shape, y_train.shape)
print(X_cv_tpp_norm.shape, y_cv.shape)
print(X_test_tpp_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16501, 1) (16501,)
====================================================================================================
```

1.5.4 Merging numerical & categorical featueres

- we need to merge all the numerical vectors & catogorical vectors

In [72]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_numcat = hstack((X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_norm, X_tra
X_cv_numcat = hstack((X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe, X_cv_price_norm, X_cv_cat_ohe, X_(
X_te_numcat = hstack((X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe, X_test_price_norm, X_test_ca

print("Final Data matrix")
print(X_tr_numcat.shape, y_train.shape)
print(X_cv_numcat.shape, y_cv.shape)
print(X_te_numcat.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 102) (22445,)
(11055, 102) (11055,)
(16501, 102) (16501,)
====================================================================================================
```

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/)](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)](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/)](http://zetcode.com/python/prettytable/)

2. K Nearest Neighbor

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

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

In [ ]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code

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

2.4.1 Applying KNN brute force on BOW, SET 1

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

In [74]:
```
 # merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_set1 = hstack((X_train_essay_bow, X_train_titles_bow, X_tr_numcat )).tocsr()
X_cv_set1 = hstack((X_cv_essay_bow, X_cv_titles_bow, X_cv_numcat)).tocsr()
X_te_set1 = hstack((X_test_essay_bow, X_test_titles_bow, X_te_numcat )).tocsr()

print("Final Data matrix")
print(X_tr_set1.shape, y_train.shape)
print(X_cv_set1.shape, y_cv.shape)
print(X_te_set1.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 10128) (22445,)
(11055, 10128) (11055,)
(16501, 10128) (16501,)
====================================================================================================
```

In [73]:
```
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 cr_loop will be 49041 - 49041%1000 = 49000
# in this for loop we will iterate unti 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
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [99]:
```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
a = []
b = []
K = [1, 5, 10, 15, 21, 25, 31, 35, 41, 51, 55, 57, 60]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr_set1, y_train)
    y_train_pred = batch_predict(neigh, X_tr_set1)
    y_cv_pred = batch_predict(neigh, X_cv_set1)
# 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))
    a.append(y_train_pred)
    b.append(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()
```

```
  0%|                                                                                                 | 0/13
[00:00<?, ?it/s]

  8%|██████                                                                                  | 1/13 [01:23<1
6:43, 83.65s/it]

 15%|███████████                                                                             | 2/13 [02:50<1
5:31, 84.73s/it]

 23%|█████████████████                                                                       | 3/13 [04:15<1
4:07, 84.79s/it]

 31%|███████████████████████                                                                 | 4/13 [05:40<1
2:42, 84.78s/it]

 38%|████████████████████████████                                                            | 5/13 [07:06<1
1:21, 85.21s/it]

 46%|██████████████████████████████████                                                      | 6/13 [08:31<0
9:56, 85.17s/it]

 54%|████████████████████████████████████████                                                | 7/13 [09:56<0
8:30, 85.05s/it]

 62%|██████████████████████████████████████████████                                          | 8/13 [11:17<0
6:59, 83.91s/it]

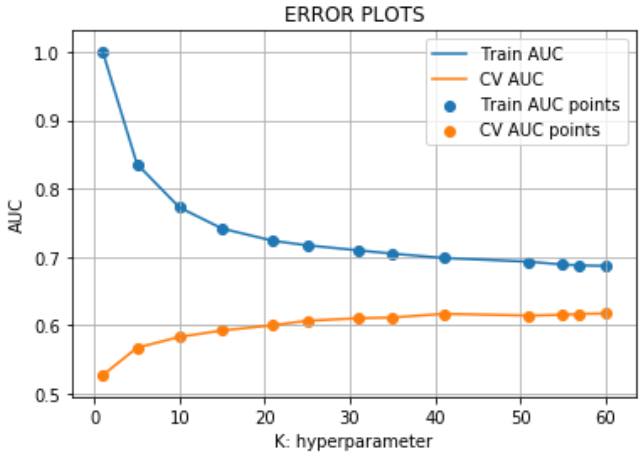
 69%|████████████████████████████████████████████████████                                    | 9/13 [12:45<0
5:39, 84.96s/it]

 77%|██████████████████████████████████████████████████████████                              | 10/13 [14:13<0
4:17, 85.83s/it]

 85%|████████████████████████████████████████████████████████████████                        | 11/13 [15:46<0
2:56, 88.08s/it]

 92%|████████████████████████████████████████████████████████████████████████                | 12/13 [17:16<0
1:28, 88.76s/it]
```

```
100%|██████████████████████████████████████████████████████████████████████████| 13/13 [18:47<0
0:00, 89.31s/it]
```



In [76]:
```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between the trai
best_k = 59
```

In [74]:
```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
```

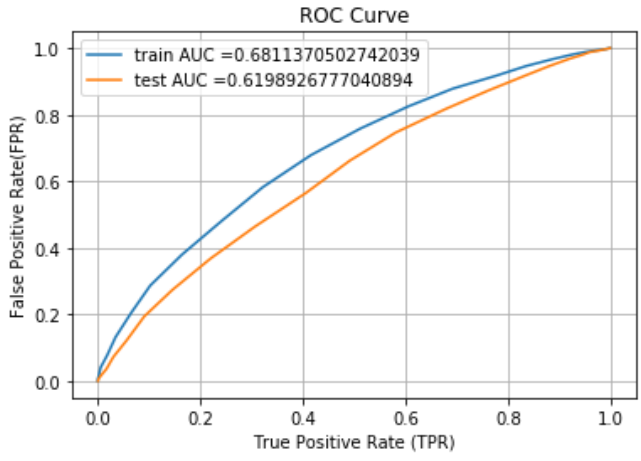
In [79]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_cu
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_tr_set1, 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_set1)
y_test_pred = batch_predict(neigh, X_te_set1)

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("ROC Curve")
plt.grid()
plt.show()
```



In [79]:
```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

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

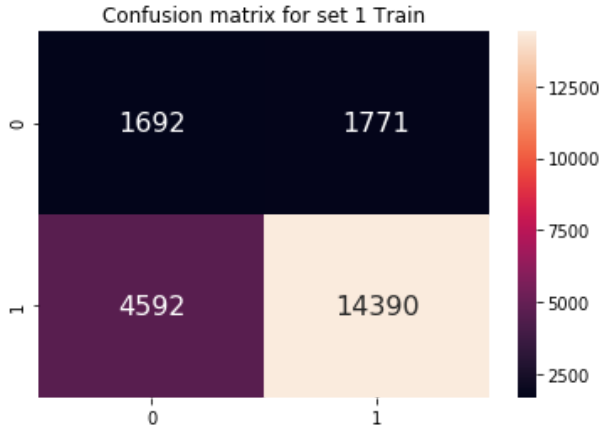
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [81]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24986989643163915 for threshold 0.78
[[ 1692  1771]
 [ 4592 14390]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24993196666026954 for threshold 0.797
[[1294 1252]
 [4711 9244]]
```

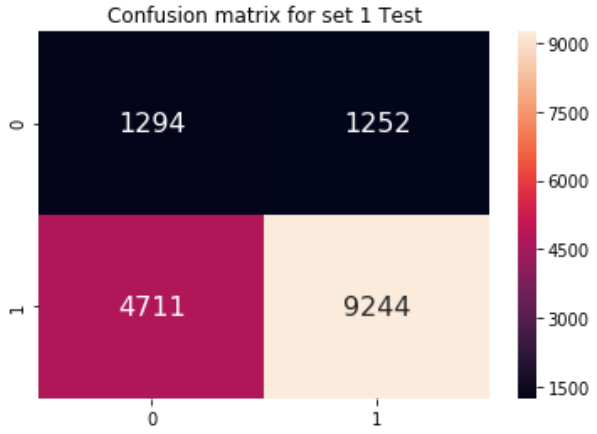
In [89]:
```
#How to plot confusion matrix using heat map - https://stackoverflow.com/questions/35572000/how-can-i-plot
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
array = [[1692,1771],
        [4592,14390]]
df_cm = pd.DataFrame(array, index = [i for i in "01"],
                  columns = [i for i in "01"])
plt.figure
plt.title('Confusion matrix for set 1 Train')
sn.heatmap(df_cm, annot=True, annot_kws={"size":16}, fmt='g')
```

Out[89]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1607d63c6a0>
```



In [90]:
```
#How to plot confusion matrix using heat map - https://stackoverflow.com/questions/35572000/how-can-i-plot
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
array = [[1294,1252],
        [4711,9244]]
df_cm = pd.DataFrame(array, index = [i for i in "01"],
                  columns = [i for i in "01"])
plt.figure
plt.title('Confusion matrix for set 1 Test')
sn.heatmap(df_cm, annot=True, annot_kws={"size":16}, fmt='g')
```

Out[90]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1607d4e2630>
```



2.4.2 Applying KNN brute force on TFIDF, SET 2

Preparing Set 2 - categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)

In [84]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_set2 = hstack((X_train_essay_tfidf, X_train_titles_tfidf, X_tr_numcat)).tocsr()
X_cv_set2 = hstack((X_cv_essay_tfidf, X_cv_titles_tfidf, X_cv_numcat)).tocsr()
X_te_set2 = hstack((X_test_essay_tfidf, X_test_titles_tfidf, X_te_numcat )).tocsr()

print("Final Data matrix")
print(X_tr_set2.shape, y_train.shape)
print(X_cv_set2.shape, y_cv.shape)
print(X_te_set2.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 10127) (22445,)
(11055, 10127) (11055,)
(16501, 10127) (16501,)
====================================================================================================
```

In [105]:
```
train_auc = []
cv_auc = []
a = []
b = []
K = [1, 5, 10, 15, 21, 25, 31, 35, 41, 51, 55, 57, 60]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr_set2, y_train)
    y_train_pred = batch_predict(neigh, X_tr_set2)
    y_cv_pred = batch_predict(neigh, X_cv_set2)
# 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))
    a.append(y_train_pred)
    b.append(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()
```

```
  0%|                                                                                                 | 0/13
[00:00<?, ?it/s]

  8%|██████                                                                                          | 1/13 [01:25<1
7:06, 85.57s/it]

 15%|███████████                                                                                     | 2/13 [02:56<1
5:58, 87.13s/it]

 23%|█████████████████                                                                               | 3/13 [04:28<1
4:46, 88.68s/it]

 31%|███████████████████████                                                                         | 4/13 [05:53<1
3:07, 87.54s/it]

 38%|████████████████████████████                                                                    | 5/13 [07:20<1
1:38, 87.29s/it]

 46%|██████████████████████████████████                                                              | 6/13 [08:45<1
0:06, 86.67s/it]

 54%|████████████████████████████████████████                                                        | 7/13 [10:17<0
8:49, 88.28s/it]

 62%|██████████████████████████████████████████████                                                  | 8/13 [11:53<0
7:32, 90.55s/it]

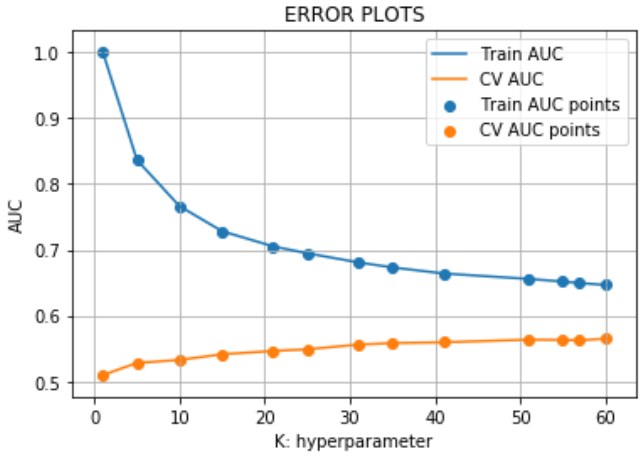
 69%|████████████████████████████████████████████████████                                            | 9/13 [13:25<0
6:03, 90.89s/it]

 77%|██████████████████████████████████████████████████████████                                     | 10/13 [14:58<0
4:34, 91.53s/it]

 85%|████████████████████████████████████████████████████████████████                               | 11/13 [16:26<0
3:01, 90.52s/it]

 92%|████████████████████████████████████████████████████████████████████                           | 12/13 [17:47<0
1:27, 87.70s/it]

100%|██████████████████████████████████████████████████████████████████████████████████████████████| 13/13 [19:03<0
0:00, 84.35s/it]
```



In [83]:
```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between the trai
best_k = 59
```

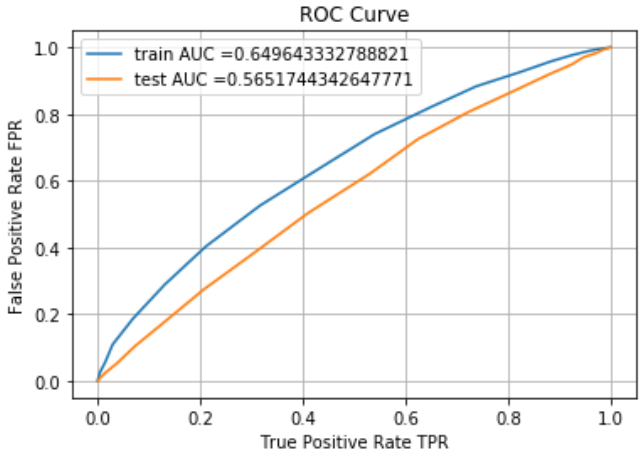
In [84]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_cu
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_tr_set2, 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_set2)
y_test_pred = batch_predict(neigh, X_te_set2)

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("ROC Curve")
plt.grid()
plt.show()
```

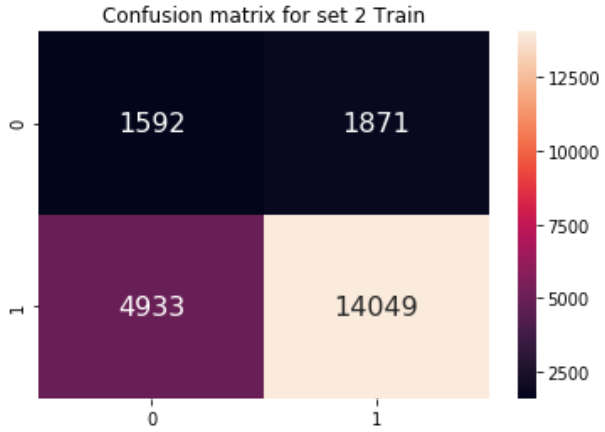


In [85]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24837728058567912 for threshold 0.831
[[ 1592  1871]
 [ 4933 14049]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2490126680855449 for threshold 0.847
[[1193 1353]
 [5277 8678]]
```

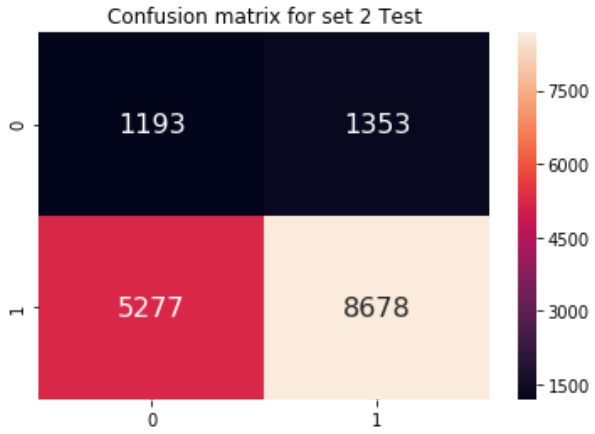
In [91]:
```
#How to plot confusion matrix using heat map - https://stackoverflow.com/questions/35572000/how-can-i-plot
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
array = [[1592,1871],
        [4933,14049]]
df_cm = pd.DataFrame(array, index = [i for i in "01"],
                  columns = [i for i in "01"])
plt.figure
plt.title('Confusion matrix for set 2 Train')
sn.heatmap(df_cm, annot=True, annot_kws={"size":16}, fmt='g')
```

Out[91]: <matplotlib.axes._subplots.AxesSubplot at 0x1607d972160>



In [92]:
```
#How to plot confusion matrix using heat map - https://stackoverflow.com/questions/35572000/how-can-i-plot
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
array = [[1193,1353],
        [5277,8678]]
df_cm = pd.DataFrame(array, index = [i for i in "01"],
                  columns = [i for i in "01"])
plt.figure
plt.title('Confusion matrix for set 2 Test')
sn.heatmap(df_cm, annot=True, annot_kws={"size":16}, fmt='g')
```

Out[92]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1607d82c0f0>
```



2.4.3 Applying KNN brute force on AVG W2V, SET 3

Preparing Set 3 - categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)

In [86]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_set3 = hstack((avg_w2v_vectors_extr, avg_w2v_vectors_txtr, X_tr_numcat)).tocsr()
X_cv_set3 = hstack((avg_w2v_vectors_excv, avg_w2v_vectors_txcv, X_cv_numcat)).tocsr()
X_te_set3 = hstack((avg_w2v_vectors_exte, avg_w2v_vectors_txte, X_te_numcat )).tocsr()

print("Final Data matrix")
print(X_tr_set3.shape, y_train.shape)
print(X_cv_set3.shape, y_cv.shape)
print(X_te_set3.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 701) (22445,)
(11055, 701) (11055,)
(16501, 701) (16501,)
====================================================================================================
```

In [87]:
```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
a = []
b = []
K = [1, 5, 10, 15, 21, 25, 31, 35, 41, 51, 55, 57, 60]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr_set3, y_train)
    y_train_pred = batch_predict(neigh, X_tr_set3)
    y_cv_pred = batch_predict(neigh, X_cv_set3)
# 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))
    a.append(y_train_pred)
    b.append(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()
```

```
  0%|                                                                                                  | 0/13
[00:00<?, ?it/s]

  8%|██████                                                                                  | 1/13 [23:05<4:37:0
6, 1385.55s/it]

 15%|███████████                                                                             | 2/13 [46:10<4:13:5
8, 1385.27s/it]

 23%|████████████████▊                                                                       | 3/13 [1:09:13<3:50:4
7, 1384.73s/it]

 31%|██████████████████████▌                                                                 | 4/13 [1:32:17<3:27:4
0, 1384.55s/it]

 38%|████████████████████████████                                                            | 5/13 [1:55:23<3:04:4
0, 1385.00s/it]

 46%|█████████████████████████████████▊                                                      | 6/13 [2:18:13<2:41:0
3, 1380.48s/it]

 54%|███████████████████████████████████████▌                                                | 7/13 [2:41:10<2:17:5
5, 1379.30s/it]

 62%|█████████████████████████████████████████████▎                                          | 8/13 [3:04:14<1:55:0
4, 1380.90s/it]

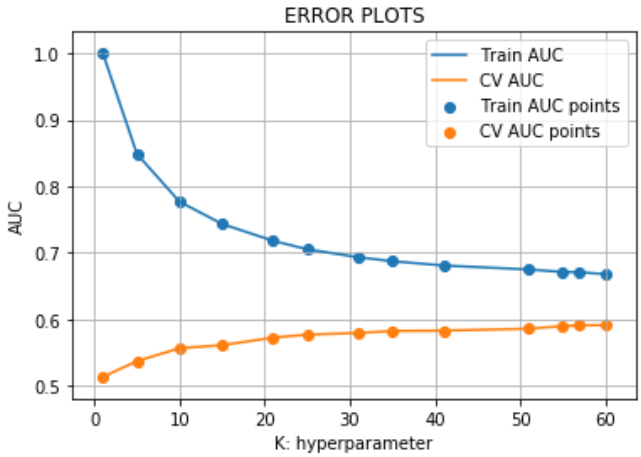
 69%|█████████████████████████████████████████████████                                       | 9/13 [3:27:19<1:32:0
8, 1382.03s/it]

 77%|████████████████████████████████████████████████████████▊                               | 10/13 [3:50:28<1:09:1
2, 1384.18s/it]

 85%|████████████████████████████████████████████████████████████████▌                       | 11/13 [4:14:08<46:2
9, 1394.74s/it]

 92%|████████████████████████████████████████████████████████████████████████▎              | 12/13 [4:37:46<23:2
1, 1401.69s/it]
```

```
100%|██████████████████████████████████████████████████████████████████████████| 13/13 [5:02:34<00:0
0, 1427.85s/it]
```



In [88]:
```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between the trai
best_k = 59
```

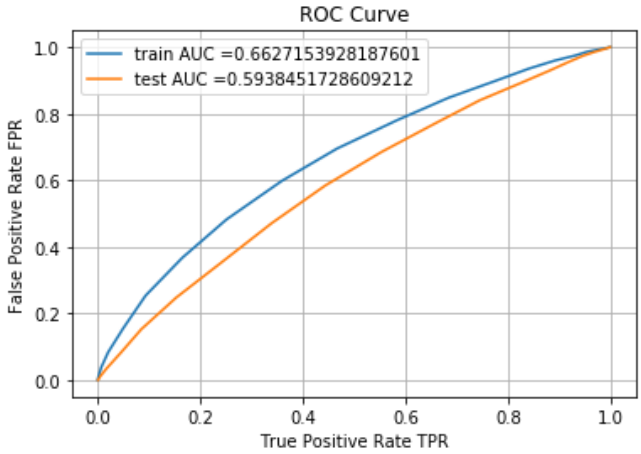
In [89]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_cu
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_tr_set3, 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_set3)
y_test_pred = batch_predict(neigh, X_te_set3)

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("ROC Curve")
plt.grid()
plt.show()
```

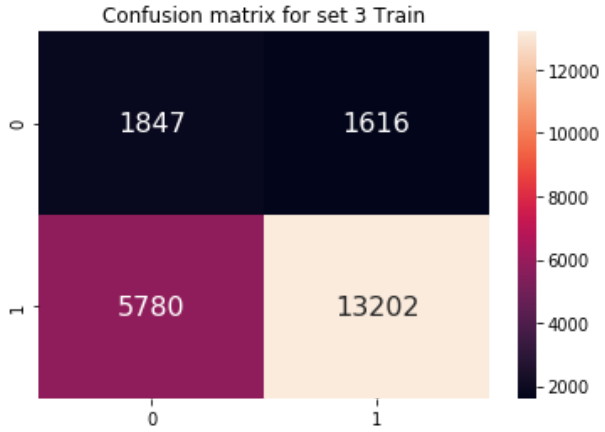


In [90]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24888760510954921 for threshold 0.847
[[ 1847  1616]
 [ 5780 13202]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24718841810297748 for threshold 0.847
[[1138 1408]
 [4396 9559]]
```

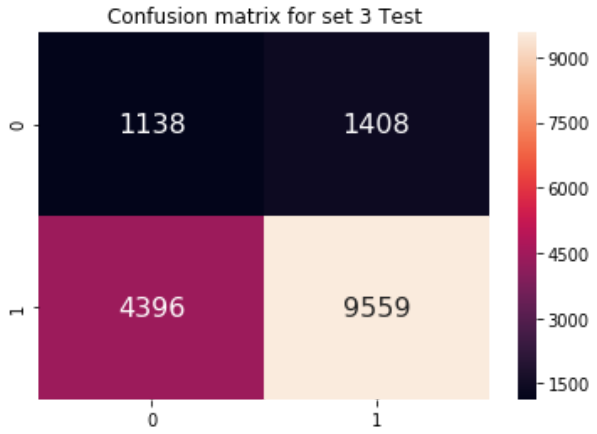
In [93]:
```
#How to plot confusion matrix using heat map - https://stackoverflow.com/questions/35572000/how-can-i-plot
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
array = [[1847,1616],
        [5780,13202]]
df_cm = pd.DataFrame(array, index = [i for i in "01"],
                  columns = [i for i in "01"])
plt.figure
plt.title('Confusion matrix for set 3 Train')
sn.heatmap(df_cm, annot=True, annot_kws={"size":16}, fmt='g')
```

Out[93]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1607d5e0390>
```



In [94]:
```
#How to plot confusion matrix using heat map - https://stackoverflow.com/questions/35572000/how-can-i-plot
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt
array = [[1138,1408],
        [4396,9559]]
df_cm = pd.DataFrame(array, index = [i for i in "01"],
                  columns = [i for i in "01"])
plt.figure
plt.title('Confusion matrix for set 3 Test')
sn.heatmap(df_cm, annot=True, annot_kws={"size":16}, fmt='g')
```

Out[94]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1607d8c50f0>
```



2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

Preparing Set 4 - categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

In [75]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_set4 = hstack((tfidf_w2v_vectors_extr, tfidf_w2v_vectors_txtr, X_tr_numcat)).tocsr()
X_cv_set4 = hstack((tfidf_w2v_vectors_excv, tfidf_w2v_vectors_txcv, X_cv_numcat)).tocsr()
X_te_set4 = hstack((tfidf_w2v_vectors_exte, tfidf_w2v_vectors_txte, X_te_numcat )).tocsr()

print("Final Data matrix")
print(X_tr_set4.shape, y_train.shape)
print(X_cv_set4.shape, y_cv.shape)
print(X_te_set4.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(22445, 702) (22445,)
(11055, 702) (11055,)
(16501, 702) (16501,)
====================================================================================================
```

In [82]:
```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []
a = []
b = []
K = [1, 5, 10, 15, 21, 25, 31, 35, 41, 51, 55, 57, 60]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr_set4, y_train)
    y_train_pred = batch_predict(neigh, X_tr_set4)
    y_cv_pred = batch_predict(neigh, X_cv_set4)
# 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))
    a.append(y_train_pred)
    b.append(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()
```

```
  0%|                                                                                           | 0/13
[00:00<?, ?it/s]

  8%|██████                                                                                 | 1/13 [23:45<4:45:1
0, 1425.86s/it]

 15%|███████████                                                                            | 2/13 [47:25<4:21:0
4, 1424.07s/it]

 23%|█████████████████                                                                      | 3/13 [1:11:28<3:58:1
7, 1429.74s/it]

 31%|███████████████████████                                                                | 4/13 [1:35:30<3:34:5
9, 1433.31s/it]

 38%|████████████████████████████                                                           | 5/13 [2:00:45<3:14:2
3, 1457.98s/it]

 46%|██████████████████████████████████                                                     | 6/13 [2:26:34<2:53:1
5, 1485.11s/it]

 54%|████████████████████████████████████████                                               | 7/13 [2:51:13<2:28:1
9, 1483.29s/it]

 62%|██████████████████████████████████████████████                                         | 8/13 [3:14:59<2:02:1
0, 1466.15s/it]

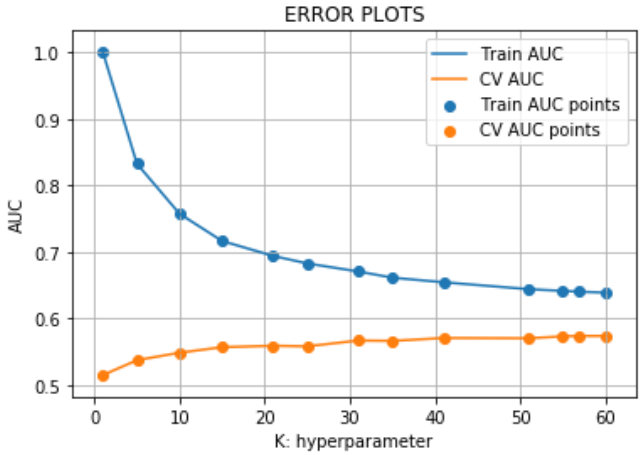
 69%|███████████████████████████████████████████████████                                    | 9/13 [3:38:53<1:37:0
6, 1456.51s/it]

 77%|███████████████████████████████████████████████████████████                            | 10/13 [4:02:24<1:12:0
8, 1442.70s/it]

 85%|███████████████████████████████████████████████████████████████                        | 11/13 [4:25:48<47:4
2, 1431.24s/it]

 92%|███████████████████████████████████████████████████████████████████████                | 12/13 [4:48:27<23:2
9, 1409.55s/it]
```

```
100%|██████████████████████████████████████████████████████████████████████████| 13/13 [5:12:01<00:0
0, 1410.79s/it]
```



In [76]:
```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between the trai
best_k = 59
```

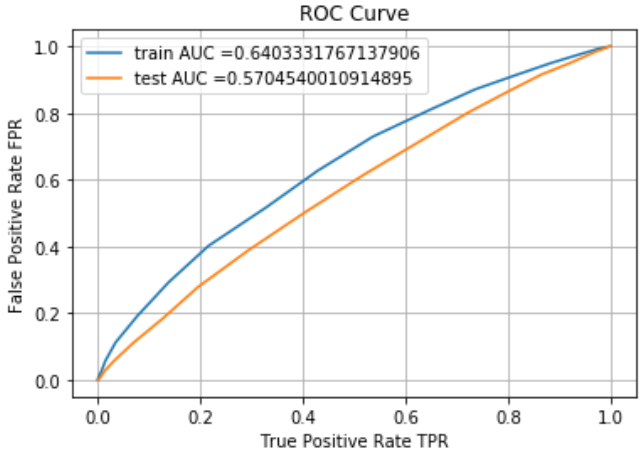
In [77]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_cu
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_tr_set4, 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_set4)
y_test_pred = batch_predict(neigh, X_te_set4)

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("ROC Curve")
plt.grid()
plt.show()
```

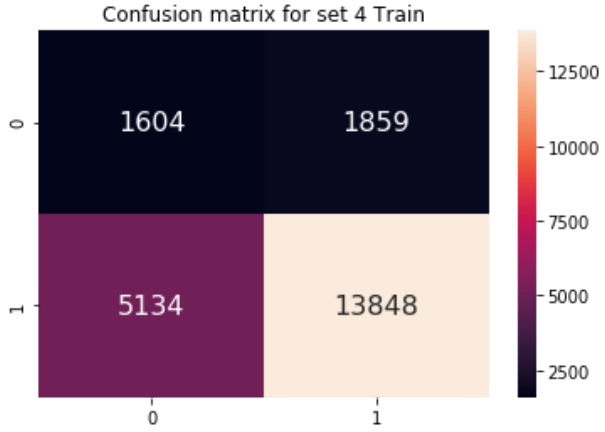


In [80]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24864445048346995 for threshold 0.831
[[ 1604  1859]
 [ 5134 13848]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24964456051079617 for threshold 0.847
[[1225 1321]
 [5372 8583]]
```

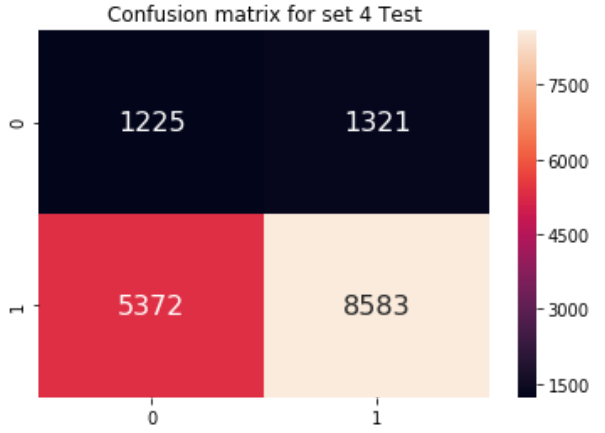
In [95]:
```
#How to plot confusion matrix using heat map - https://stackoverflow.com/questions/35572000/how-can-i-plot
array = [[1604,1859],
        [5134,13848]]
df_cm = pd.DataFrame(array, index = [i for i in "01"],
                  columns = [i for i in "01"])
plt.figure
plt.title('Confusion matrix for set 4 Train')
sn.heatmap(df_cm, annot=True, annot_kws={"size":16}, fmt='g')
```

Out[95]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1607d5cf940>
```



In [96]:
```
#How to plot confusion matrix using heat map - https://stackoverflow.com/questions/35572000/how-can-i-plot
array = [[1225,1321],
        [5372,8583]]
df_cm = pd.DataFrame(array, index = [i for i in "01"],
                  columns = [i for i in "01"])
plt.figure
plt.title('Confusion matrix for set 4 Test')
sn.heatmap(df_cm, annot=True, annot_kws={"size":16}, fmt='g')
```

Out[96]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1607d967cf8>
```



2.5 Feature selection with `SelectKBest`

In [85]:
```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code

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

from sklearn.feature_selection import SelectKBest, chi2
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

best_2k = SelectKBest(chi2, k=2000).fit(X_tr_set2, y_train)

X_tr_set2_2k = best_2k.transform(X_tr_set2)
X_cv_set2_2k = best_2k.transform(X_cv_set2)
X_te_set2_2k = best_2k.transform(X_te_set2)

print("Final Data matrix for best 2k features")
print(X_tr_set2_2k.shape, y_train.shape)
print(X_cv_set2_2k.shape, y_cv.shape)
print(X_te_set2_2k.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix for best 2k features
(22445, 2000) (22445,)
(11055, 2000) (11055,)
(16501, 2000) (16501,)
====================================================================================================
```

In [118]:
```
train_auc = []
cv_auc = []
a = []
b = []
K = [1, 5, 10, 15, 21, 25, 31, 35, 41, 51, 55, 57, 60]

for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_tr_set2_2k, y_train)
    y_train_pred = batch_predict(neigh, X_tr_set2_2k)
    y_cv_pred = batch_predict(neigh, X_cv_set2_2k)
# 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))
    a.append(y_train_pred)
    b.append(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()
```

```
  0%|                                                                                                 | 0/13
[00:00<?, ?it/s]

  8%|██████                                                                                          | 1/13 [00:42<0
8:29, 42.47s/it]

 15%|███████████                                                                                     | 2/13 [01:31<0
8:08, 44.40s/it]

 23%|█████████████████                                                                               | 3/13 [02:20<0
7:37, 45.73s/it]

 31%|███████████████████████                                                                         | 4/13 [03:14<0
7:13, 48.19s/it]

 38%|████████████████████████████                                                                    | 5/13 [04:06<0
6:35, 49.42s/it]

 46%|██████████████████████████████████                                                              | 6/13 [04:57<0
5:48, 49.78s/it]

 54%|████████████████████████████████████████                                                        | 7/13 [05:43<0
4:52, 48.82s/it]

 62%|██████████████████████████████████████████████                                                  | 8/13 [06:32<0
4:04, 48.91s/it]

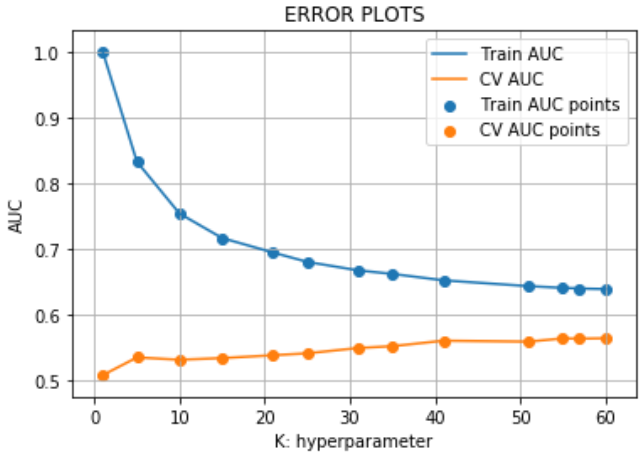
 69%|████████████████████████████████████████████████████                                            | 9/13 [07:23<0
3:17, 49.46s/it]

 77%|██████████████████████████████████████████████████████████                                      | 10/13 [08:13<0
2:28, 49.58s/it]

 85%|████████████████████████████████████████████████████████████████                                | 11/13 [09:02<0
1:39, 49.57s/it]

 92%|████████████████████████████████████████████████████████████████████                            | 12/13 [09:53<0
0:49, 49.86s/it]

100%|████████████████████████████████████████████████████████████████████████| 13/13 [10:42<0
0:00, 49.74s/it]
```



In [86]:
```
# from the error plot we choose K such that, we will have maximum AUC on cv data and gap between the trai
best_k = 59
```

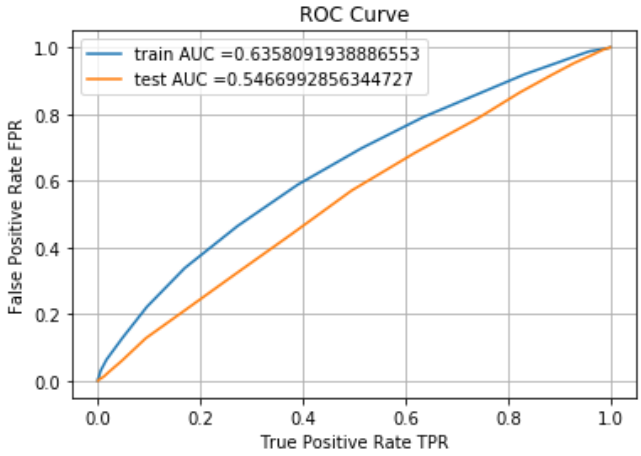
In [87]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_cu
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k)
neigh.fit(X_tr_set2_2k, 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_set2_2k)
y_test_pred = batch_predict(neigh, X_te_set2_2k)

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("ROC Curve")
plt.grid()
plt.show()
```

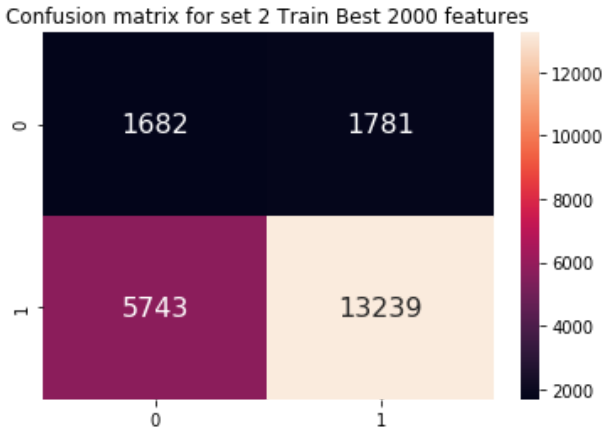


In [88]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr)))
```

```
====================================================================================================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2497956825711417 for threshold 0.831
[[ 1682  1781]
 [ 5743 13239]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24997778503192472 for threshold 0.847
[[1285 1261]
 [5989 7966]]
```

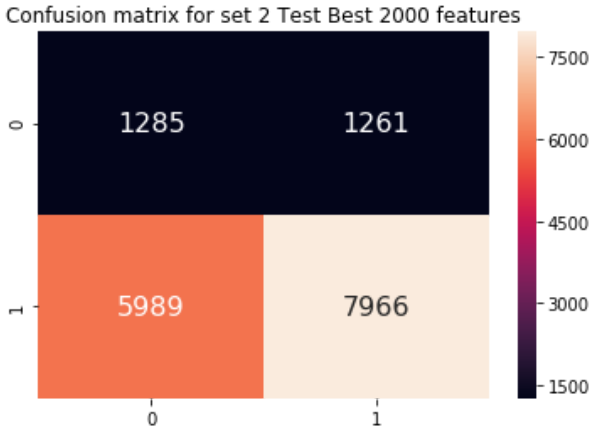
In [97]:
```
#How to plot confusion matrix using heat map - https://stackoverflow.com/questions/35572000/how-can-i-plot
array = [[1682,1781],
        [5743,13239]]
df_cm = pd.DataFrame(array, index = [i for i in "01"],
                  columns = [i for i in "01"])
plt.figure
plt.title('Confusion matrix for set 2 Train Best 2000 features')
sn.heatmap(df_cm, annot=True, annot_kws={"size":16}, fmt='g')
```

Out[97]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1607d4fa908>
```



In [98]:
```
#How to plot confusion matrix using heat map - https://stackoverflow.com/questions/35572000/how-can-i-plot
array = [[1285,1261],
        [5989,7966]]
df_cm = pd.DataFrame(array, index = [i for i in "01"],
                  columns = [i for i in "01"])
plt.figure
plt.title('Confusion matrix for set 2 Test Best 2000 features')
sn.heatmap(df_cm, annot=True, annot_kws={"size":16}, fmt='g')
```

Out[98]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1607d7b2470>
```



3. Conclusions

Please compare all your models using Prettytable library

In [99]:
```
from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]
x.add_row(["BOW", "Brute", 59, 0.61])
x.add_row(["TFIDF", "Brute", 59, 0.56])
x.add_row(["AVG W2V", "Brute", 59, 0.59])
x.add_row(["TFIDF W2V", "Brute", 59, 0.57])
x.add_row(["TFIDF", "Top 2000", 59, 0.54])
print(x)
```

```
+------------+----------+-----------------+------+
| Vectorizer |  Model   | Hyper Parameter | AUC  |
+------------+----------+-----------------+------+
|    BOW     |  Brute   |        59       | 0.61 |
|   TFIDF    |  Brute   |        59       | 0.56 |
|  AVG W2V   |  Brute   |        59       | 0.59 |
| TFIDF W2V  |  Brute   |        59       | 0.57 |
|   TFIDF    | Top 2000 |        59       | 0.54 |
+------------+----------+-----------------+------+
```

In [ ]:
```
```