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	
`project_id`	A unique identifier for the proposed project
`project_title`	Title of tl • `Art Wil` •
`project_grade_category`	Grade level of students for which the project is targetec • • • •

Feature	
project_subject_categories	One or more (comma-separated) subject categories f following enur • Li • Literacy & Languag
school_state	State where school is located (Two- (https://en.wikipedia.org/wiki/List_of_U.S._state_abbrevia
project_subject_subcategories	One or more (comma-separated) subject subcate • Literature & Writing
project_resource_summary	An explanation of the resources needed for t • My students need hands on literacy ma sen
project_essay_1	F
project_essay_2	Sec
project_essay_3	Tl
project_essay_4	Fou
project_submitted_datetime	Datetime when project application was submitted. **Ex**
teacher_id	A unique identifier for the teacher of the propo bdf8baa8fedef6b
teacher_prefix	Teacher's title. One of the following
teacher_number_of_previously_posted_projects	Number of project applications previously submitte

* 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 [3]:
```
import os
os.chdir('F:/f/MY____/AAIC/AssignmentS/2DonarChoose/Assignments_DonorsChoose
os.getcwd()
```

Out[3]: `'F:\\f\\MY____\\AAIC\\AssignmentS\\2DonarChoose\\Assignments_DonorsChoose_2018'`

In [7]:
```
%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 sea
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 sklearn.metrics import roc_auc_score
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 [8]:
```
project_data = pd.read_csv('train_data.csv', nrows=50000)
resource_data = pd.read_csv('resources.csv')
```

In [9]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
print(type(project_data))
print(type(resource_data))
```

```
Number of data points in train data (50000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'scho
ol_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']
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

In [10]:
```
print("Number of data Points in Resources Data ",resource_data.shape)
print("-"*50)
print("The Features of resources data ", resource_data.columns.values)
```

```
Number of data Points in Resources Data  (1541272, 4)
--------------------------------------------------
The Features of resources data  ['id' 'description' 'quantity' 'price']
```

In [11]:
```
print("Number of data points in resource data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[11]:

	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

In [ ]:
```

```

1.2 preprocessing of `project_subject_categories`

In [12]:

```
categories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverfl

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-str
cat_list = []
for i in categories:
    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 & Scienc
        if 'The' in j.split(): # this will split each of the catogory based
            j=j.replace('The','') # if we have the words "The" we are going
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(e
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the t
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

```

1.3 preprocessing of `project_subject_subcategories`

In [13]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverfl

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

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 & Scienc
        if 'The' in j.split(): # this will split each of the catogory based
            j=j.replace('The','') # if we have the words "The" we are going
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(e
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the t
        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/22898
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

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

1.3 Text preprocessing

In [14]:
```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [15]:
```
project_data.head(2)
```

Out[15]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_sul
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	20
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	20

In [11]:
```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [9]:
```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
#print(project_data['essay'].values[99999])
print("="*50)
```

```
My students are English learners that are working on English as their second or
third languages. We are a melting pot of refugees, immigrants, and native-born
Americans bringing the gift of language to our school. \r\n\r\n We have over 24
languages represented in our English Learner program with students at every lev
el of mastery.  We also have over 40 countries represented with the families wi
thin our school.  Each student brings a wealth of knowledge and experiences to
us that open our eyes to new cultures, beliefs, and respect.\"The limits of you
r language are the limits of your world.\"-Ludwig Wittgenstein  Our English lea
rner's have a strong support system at home that begs for more resources.  Many
times our parents are learning to read and speak English along side of their ch
ildren.  Sometimes this creates barriers for parents to be able to help their c
hild learn phonetics, letter recognition, and other reading skills.\r\n\r\nBy p
roviding these dvd's and players, students are able to continue their mastery o
f the English language even if no one at home is able to assist.  All families
with students within the Level 1 proficiency status, will be a offered to be a
part of this program.  These educational videos will be specially chosen by the
English Learner Teacher and will be sent home regularly to watch.  The videos a
re to help the child develop early reading skills.\r\n\r\nParents that do not h
ave access to a dvd player will have the opportunity to check out a dvd player
to use for the year.  The plan is to use these videos and educational dvd's for
the years to come for other EL students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this year all
love learning, at least most of the time. At our school, 97.3% of the students
receive free or reduced price lunch. Of the 560 students, 97.3% are minority st
udents. \r\nThe school has a vibrant community that loves to get together and c
elebrate. Around Halloween there is a whole school parade to show off the beaut
iful costumes that students wear. On Cinco de Mayo we put on a big festival wit
h crafts made by the students, dances, and games. At the end of the year the sc
hool hosts a carnival to celebrate the hard work put in during the school year,
with a dunk tank being the most popular activity.My students will use these fiv
e brightly colored Hokki stools in place of regular, stationary, 4-legged chair
s. As I will only have a total of ten in the classroom and not enough for each
student to have an individual one, they will be used in a variety of ways. Duri
ng independent reading time they will be used as special chairs students will e
ach use on occasion. I will utilize them in place of chairs at my small group t
ables during math and reading times. The rest of the day they will be used by t
he students who need the highest amount of movement in their life in order to s
tay focused on school.\r\n\r\nWhenever asked what the classroom is missing, my
students always say more Hokki Stools. They can't get their fill of the 5 stool
s we already have. When the students are sitting in group with me on the Hokki
Stools, they are always moving, but at the same time doing their work. Anytime
the students get to pick where they can sit, the Hokki Stools are the first to
be taken. There are always students who head over to the kidney table to get on
e of the stools who are disappointed as there are not enough of them. \r\n\r\nW
e ask a lot of students to sit for 7 hours a day. The Hokki stools will be a co
mpromise that allow my students to do desk work and move at the same time. Thes
e stools will help students to meet their 60 minutes a day of movement by allow
```

```
ing them to activate their core muscles for balance while they sit. For many of
my students, these chairs will take away the barrier that exists in schools for
a child who can't sit still.nannan
==================================================
How do you remember your days of school? Was it in a sterile environment with p
lain walls, rows of desks, and a teacher in front of the room? A typical day in
our room is nothing like that. I work hard to create a warm inviting themed roo
m for my students look forward to coming to each day.\r\n\r\nMy class is made u
p of 28 wonderfully unique boys and girls of mixed races in Arkansas.\r\nThey a
ttend a Title I school, which means there is a high enough percentage of free a
nd reduced-price lunch to qualify. Our school is an \"open classroom\" concept,
which is very unique as there are no walls separating the classrooms. These 9 a
nd 10 year-old students are very eager learners; they are like sponges, absorbi
ng all the information and experiences and keep on wanting more.With these reso
urces such as the comfy red throw pillows and the whimsical nautical hanging de
cor and the blue fish nets, I will be able to help create the mood in our class
room setting to be one of a themed nautical environment. Creating a classroom e
nvironment is very important in the success in each and every child's educatio
n. The nautical photo props will be used with each child as they step foot into
our classroom for the first time on Meet the Teacher evening. I'll take picture
s of each child with them, have them developed, and then hung in our classroom
ready for their first day of 4th grade.  This kind gesture will set the tone be
fore even the first day of school! The nautical thank you cards will be used th
roughout the year by the students as they create thank you cards to their team
groups.\r\n\r\nYour generous donations will help me to help make our classroom
a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of mon
ey out of my own pocket on resources to get our classroom ready. Please conside
r helping with this project to make our new school year a very successful one.
Thank you!nannan
==================================================
```

In [22]:

```python
# 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 [23]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
My kindergarten students have varied disabilities ranging from speech and langu
age delays, cognitive delays, gross/fine motor delays, to autism. They are eage
r beavers and always strive to work their hardest working past their limitation
s. \r\n\r\nThe 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 disabilities and limitations, my students love coming to
school and come eager to learn and explore.Have you ever felt like you had ants
in your pants and you needed to groove and move as you were in a meeting? This
is how my kids feel all the time. The want to be able to move as they learn or
so they say.Wobble chairs are the answer and I love then because they develop t
heir core, which enhances gross motor and in Turn fine motor skills. \r\nThey a
lso 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 success. The number toss and color and shape mats can make that happ
en. My students will forget they are doing work and just have the fun a 6 year
old deserves.nannan
==================================================
```

In [19]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remove-lin
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
My kindergarten students have varied disabilities ranging from speech and langu
age delays, cognitive delays, gross/fine motor delays, to autism. They are eage
r beavers and always strive to work their hardest working past their limitation
s.     The materials we have are the ones I seek out for my students. I teach i
n a Title I school where most of the students receive free or reduced price lun
ch.  Despite their disabilities and limitations, my students love coming to sch
ool and come eager to learn and explore.Have you ever felt like you had ants in
your pants and you needed to groove and move as you were in a meeting? This is
how my kids feel all the time. The want to be able to move as they learn or so
they say.Wobble chairs are the answer and I love then because they develop thei
r core, which enhances gross motor and in Turn fine motor skills.   They also w
ant 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 t
o our success. The number toss and color and shape mats can make that happen. M
y students will forget they are doing work and just have the fun a 6 year old d
eserves.nannan
```

In [20]:
```
#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 langu
age delays cognitive delays gross fine motor delays to autism They are eager be
avers and always strive to work their hardest working past their limitations Th
e 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 e
ager to learn and explore Have you ever felt like you had ants in your pants an
d you needed to groove and move as you were in a meeting This is how my kids fe
el 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 enh
ances 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 learn to count
by jumping and playing Physical engagement is the key to our success The number
toss and color and shape mats can make that happen My students will forget they
are doing work and just have the fun a 6 year old deserves nannan
```

In [24]:
```
# 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', 'yo
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'h
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', '
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this',
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'bec
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into'
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 't
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shou
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn'
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't",
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "sho
            'won', "won't", 'wouldn', "wouldn't"]
```

In [18]:
```
# Combining all the above stundents
#from tqdm import tqdm
#preprocessed_essays = []
## tqdm is for printing the status bar
#for sentance in tqdm(project_data['essay'].values):
#    sent = decontracted(sentance)
#    sent = sent.replace('\\r', ' ')
#    sent = sent.replace('\\"', ' ')
#    sent = sent.replace('\\n', ' ')
#    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
#    # https://gist.github.com/sebleier/554280
#    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
#    preprocessed_essays.append(sent.lower().strip())
```

In [19]:
```
# after preprocesing
#preprocessed_essays[20000]
```

1.4 Preprocessing of `project_title`

In [ ]:
```
#sent = decontracted(project_data['project_title'].values[12])
#print(sent)
```

In [ ]:
```
# similarly you can preprocess the titles also
#from tqdm import tqdm
#preprocessed_titles = []
## tqdm is for printing the status bar
#for sentance in tqdm(project_data['project_title'].values):
#    sent = decontracted(sentance)
#    sent = sent.replace('\\r', ' ')
#    sent = sent.replace('\\"', ' ')
#    sent = sent.replace('\\n', ' ')
#    sent = re.sub('[b^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_titles.append(sent.lower().strip())
```

In [ ]:
```

```

1.5 Preparing data for models

Train_Test_Split

In [206]:
```
#Splitting Data into train and Test sklearn https://scikit-learn.org/stable/
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(project_data,
                                                    project_data["project_is
                                                    test_size = 0.33, strati
```

In [207]:
```
x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train,
                                                test_size = 0.33, stratify =
```

In [249]:
```
print(x_test.columns)
print(x_train.columns)
#print(x_cv.columns)
#print(x_train.shape)
#print(x_test.shape)
#print(x_cv.shape)
```

```
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', 'clean_categories',
       'clean_subcategories', 'essay', 'price_x', 'quantity_x', 'price_y',
       'quantity_y'],
      dtype='object')
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', 'clean_categories',
       'clean_subcategories', 'essay', 'price_x', 'quantity_x', 'price_y',
       'quantity_y'],
      dtype='object')
```

In [209]:
```
#https://stackoverflow.com/questions/13411544/delete-column-from-pandas-data
#x_train =
x_train.drop(["project_is_approved"], axis = 1, inplace = True)
#x_test =
x_test.drop(["project_is_approved"], axis = 1, inplace = True)
#x_cv =
x_cv.drop(["project_is_approved"], axis = 1, inplace = True)

```

In [20]:
```
print(x_train.columns)
```

```
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', 'clean_categories',
       'clean_subcategories', 'essay'],
      dtype='object')
```

In [21]:
```
print(x_train['essay'].values[0])
```

```
First grade rocks! My students come into the room each morning with smiles on t
heir faces and an eagerness to learn! They are excited about school and they ma
ke my job fun.\r\n\r\nMy students inspire me daily to be the best teacher I can
be for them. They work hard and they play hard. They deserve an amazing educati
onal experience!\r\n\r\nOur school is a Title I school, with over 75% of studen
ts receiving free or reduced-price lunch. Many of them come from homes where th
eir parents work hard to make ends meet and take care of basic needs. Luxuries
and extras are few and far between, so school is a place where they can experie
nce things that are out-of-reach at home.Technology is a fabulous motivator in
the classroom! My students get excited whenever technology is added to a lesson
or activity. They love the hands-on aspect and enjoy branching out and trying s
omething new.\r\n\r\nThe use of technology in lessons and activities assists me
in reaching the different types of learners in my classroom.\r\nThe addition of
2 HP Chromebooks to our classroom collection will allow students to further dev
elop technological skills as well as enhance learning in all subject areas. We
will use the Chromebooks to play math games, practice math facts, read and list
en to online books, engage in phonics practice, and practice typing and publish
ing stories.nannan
```

In [22]:
```
sent = decontracted(x_train['essay'].values[2000])
print(sent)
print("="*50)
```

```
I am a special education teacher with 20 years of experience, however, I still
need help providing for my students what they need.  My classroom has students
who come in and out all day long.  I work with 26 students throughout the day w
ith Learning Disabilities,  Intellectual Disabilities, Other Health Impairment
s, ADD/ADHD, and Speech and Language Impairments.  The materials I am requestin
g will help my students (who need to work on developing their independent work
skills) increase their academic skills by providing activities that are on thei
r individual level, self-paced and self-checking.I am requesting these material
s to provide my students with opportunities to improve their academic abilities
through independent learning.  I work with small groups of students who need ex
tra help in reading, writing, and/or math.  When students are with me, I strive
to make the best use of every minute we have together by keeping them engaged a
nd learning.  Many of my students struggle to complete work unless I am helping
them along.  I want to provide opportunities for them to increase their indepen
dence while increasing their academic skills.  I see some of these students for
multiple years and I am looking for new and engaging activities for them to com
plete.nannan
==================================================
```

Type *Markdown* and LaTeX: α^2

Preprocessing Train Data

In [25]:
```
# Preprocessing Train Data of Project Essays

from tqdm import tqdm
train_preprocessed_essays = []
# 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)
    train_preprocessed_essays.append(sent.lower().strip())
```

```
100%|
| 22445/22445 [00:19<00:00, 1127.12it/s]
```

In [26]:
```
train_preprocessed_essays[10]
```

Out[26]:
```
'students naturally inquisitive respectful innate concern others community well
outside community 90 love read fun exploring new genres within classroom connec
t thematic elements text outside world oftentimes bring new item piece informat
ion pertaining unit time brilliant give hope future area state country running
place another 20 30 years vital nurture naturally inquisitive minds fullest ext
ent possible help student learning skyrocket tools engage formative assessment
students use markers erasers individual lapboards provide feedback answers ques
tions posed reviewed content also use review games planning group collaboration
inferences within text reading teacher complete view grasped material not mobil
e organizer used organize markers erasers pencils glue classwork better organiz
ation ensure class runs smoothly kids teacher stay focused instruction time hig
her quality nannan'
```

In [210]:
```
#Preprocessing Test Data of Project Essays

# Combining all the above stundents
from tqdm import tqdm
test_preprocessed_essays = []
# 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)
    test_preprocessed_essays.append(sent.lower().strip())
```

```
| 14559/16500 [00:14<00:02, 882.59it/s]

 89%|
| 14650/16500 [00:14<00:02, 888.07it/s]

 89%|
| 14739/16500 [00:14<00:02, 875.58it/s]

 90%|
| 14836/16500 [00:14<00:01, 899.44it/s]

 90%|
| 14929/16500 [00:14<00:01, 905.79it/s]

 91%|
```

In [28]:
```
test_preprocessed_essays[10]
```

Out[28]:
```
'elementary school beautifully old building 91 years old houses many classrooms
ranging kindergarten fifth grade home full adults children love learn new stude
nts joining family year kindergarteners students enter classroom year excited n
ervous unaware fun magic awaits start academic journey many entering school fir
st time students come backgrounds deserving love endless opportunites love disc
overing ways guided encouraged become independent take ownership learning studi
es show music vital part learning certain areas brain activated music help many
different types learners classroom students feel inspired songs carefully choos
e match theme learn active means healthy happy dance fun beats favorite songs n
ot help sing dance along lighting fire learning many different types learners g
ood majority find music great way help remember things somewhat tedious parts d
ay clean become exciting tidy beat music slow fast kindergarten also creating t
hings whether tea party set ready serve dramatic play giant cave shivering bear
construciton area students love take pictures work proudly show classmates fami
ly pictures spark ideas ideas motivate young authors write stories camera help
time travel back small moments learning write possiblities simply endless nanna
n'
```

In [29]:
```
#Preprocessing Cross Validation Data  of Project Essays

# Combining all the above stundents
from tqdm import tqdm
cv_preprocessed_essays = []
# 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)
    cv_preprocessed_essays.append(sent.lower().strip())
```

```
100%|
| 11055/11055 [00:09<00:00, 1157.04it/s]
```

In [33]:
```
cv_preprocessed_essays[10]
```

Out[33]:
```
'students great time using bright colorful glazes expensive not afford please h
elp us get basic colors finish project help us purchase tools create beautiful
works art many first experience fine arts clay high school mix cultures socioec
onomic backgrounds students not speak much english still create clay projects a
lso students learning disabilities ceramics classes school currently california
distinguished school teachers students proud hard work one first projects stude
nts make clay introduced coil building technique well introduced making repeate
d patterns clay shapes students create original pot shows unique personality id
eas learn many artists art elements principles well learn basic ceramic buildin
g techniques influenced anything around pop culture artists even disney movies
influences lives project use common core state visual arts standards also paint
patterned pots bright vivid glaze colors make designs choose stand look complet
e students need coil extruder tool help create coils patterned pots project hel
p students learn think creatively help develop new ideas different anyone else
around also appreciate arts learn create designs fun unique students really use
equipment complete projects not normally afford high cost materials nannan'
```

Preprocessing Project Titles

In [30]:
```
#Preprocessing Train Data for Project Titles
from tqdm import tqdm
train_preprocessed_titles = []
# 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.lower() not in stopwords)
    train_preprocessed_titles.append(sent.lower().strip())
```

```
100%|[redacted]|
22445/22445 [00:00<00:00, 25508.57it/s]
```

In [31]:
```
train_preprocessed_titles[10]
```

Out[31]: 'show work student lap board markers organizer'

In [211]:
```
#Preprocessing Test Data for Project Titles
from tqdm import tqdm
test_preprocessed_titles = []
# 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.lower() not in stopwords)
    test_preprocessed_titles.append(sent.lower().strip())
```

```
| 0/16500 [00:00<?, ?it/s]

 16%|[redacted]
| 2699/16500 [00:00<00:00, 26726.79it/s]

 32%|[redacted]
| 5312/16500 [00:00<00:00, 26465.71it/s]

 49%|[redacted]
| 8105/16500 [00:00<00:00, 26812.34it/s]

 65%|[redacted]
| 10653/16500 [00:00<00:00, 26317.70it/s]

 80%|[redacted]
```

In [33]:
```
test_preprocessed_titles[10]
```

Out[33]: 'need make music capture memories'

In [34]:
```
#Preprocessing CV Data for Project Titles
from tqdm import tqdm
cv_preprocessed_titles = []
# 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.lower() not in stopwords)
    cv_preprocessed_titles.append(sent.lower().strip())
```

```
100%|[redacted]|
11055/11055 [00:00<00:00, 24970.80it/s]
```

In [35]:
```
cv_preprocessed_titles[10]
```

Out[35]: 'creative coil pottery'

In [36]:
```
project_data.columns
```

Out[36]:
```
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')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

In [212]:

```
# we use count vectorizer to convert the values into one
# Vectorizing Clean Categories
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowerc

vectorizer.fit(x_train['clean_categories'].values)
train_categories_one_hot = vectorizer.transform(x_train['clean_categories'].
test_categories_one_hot = vectorizer.transform(x_test['clean_categories'].va
cv_categories_one_hot = vectorizer.transform(x_cv['clean_categories'].values

print(vectorizer.get_feature_names())
print("Shape of  Train matrix after one hot encodig ",train_categories_one_h
print("Shape of  Test matrix after one hot encodig ",test_categories_one_hot
print("Shape of  cv  matrix after one hot encodig ",cv_categories_one_hot.sh
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'S
pecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of  Train matrix after one hot encodig  (22445, 9)
Shape of  Test matrix after one hot encodig  (16500, 9)
Shape of  cv  matrix after one hot encodig  (11055, 9)
```

In [213]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lo
vectorizer.fit(x_train["clean_subcategories"].values)
train_sub_categories_one_hot = vectorizer.transform(x_train['clean_subcatego
test_sub_categories_one_hot = vectorizer.transform(x_test['clean_subcategori
cv_sub_categories_one_hot = vectorizer.transform(x_cv['clean_subcategories']

print(vectorizer.get_feature_names())
print("Shape of Train matrix after  Trainone hot encodig ",train_sub_categor
print("Shape of test matrix after one hot encodig ",test_sub_categories_one_
print("Shape of cv_ matrix after  one hot encodig ",cv_sub_categories_one_ho
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Ex
tracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducation',
'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'CharacterEducatio
n', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_Geography',
'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness', 'EnvironmentalS
cience', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'L
iterature_Writing', 'Mathematics', 'Literacy']
Shape of Train matrix after  Trainone hot encodig  (22445, 30)
Shape of test matrix after one hot encodig  (16500, 30)
Shape of cv_ matrix after  one hot encodig  (11055, 30)
```

In [215]:
```
# you can do the similar thing with state, teacher_prefix and project_grade_
from collections import Counter
my_counter = Counter()
for word in project_data["school_state"].values:
    my_counter.update(word.split())
```

In [216]:
```
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
state_cat_dict = dict(my_counter)
storted_state_cat_dict = dict(sorted(state_cat_dict.items(), key=lambda kv:

```

In [217]:
```
# Please do the similar feature encoding with state, teacher_prefix and proj
#Using Count Vectorizer to convert the state value onto on hot encoded featu
vectorizer = CountVectorizer(vocabulary=list(storted_state_cat_dict.keys()),
vectorizer.fit(project_data['school_state'].values)
print(vectorizer.get_feature_names())


train_state_one_hot = vectorizer.transform(x_train['school_state'].values)
test_state_one_hot = vectorizer.transform(x_test['school_state'].values)
cv_state_one_hot = vectorizer.transform(x_cv['school_state'].values)

print("Shape of Train matrix after one hot encodig ",train_state_one_hot.sha
print("Shape of Test matrix after one hot encodig ",test_state_one_hot.shape
print("Shape of CV matrix after one hot encodig ",cv_state_one_hot.shape)
```

```
['VT', 'WY', 'ND', 'MT', 'RI', 'NH', 'SD', 'NE', 'AK', 'DE', 'WV', 'ME', 'NM',
'HI', 'DC', 'KS', 'ID', 'IA', 'AR', 'CO', 'MN', 'OR', 'MS', 'KY', 'NV', 'MD',
'CT', 'TN', 'AL', 'UT', 'WI', 'VA', 'AZ', 'NJ', 'OK', 'MA', 'LA', 'WA', 'MO',
'IN', 'OH', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'TX', 'NY', 'CA']
Shape of Train matrix after one hot encodig  (22445, 51)
Shape of Test matrix after one hot encodig  (16500, 51)
Shape of CV matrix after one hot encodig  (11055, 51)
```

In [218]:
```
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-ha
project_data['project_grade_category']=project_data['project_grade_category'
my_counter = Counter()
for word in project_data['project_grade_category'].values:
    my_counter.update(word.split())
```

In [219]:
```
project_cat_dict = dict(my_counter)
sorted_project_cat_dict = dict(sorted(project_cat_dict.items(), key=lambda k
```

In [220]:
```
# feature encoding for project_grade_category also
vectorizer = CountVectorizer(vocabulary=list(sorted_project_cat_dict.keys())
vectorizer.fit(project_data['project_grade_category'].values)
print(vectorizer.get_feature_names())

train_grade_one_hot = vectorizer.transform(x_train['project_grade_category']
test_grade_one_hot = vectorizer.transform(x_test['project_grade_category'].v
cv_grade_one_hot = vectorizer.transform(x_cv['project_grade_category'].value

print("Shape of Train matrix after one hot encodig ",train_grade_one_hot.sha
print("Shape of test matrix after one hot encodig ",test_grade_one_hot.shape
print("Shape of cv matrix after one hot encodig ",cv_grade_one_hot.shape)
```

```
['9-12', '6-8', '3-5', 'PreK-2', 'Grades']
Shape of Train matrix after one hot encodig  (22445, 5)
Shape of test matrix after one hot encodig  (16500, 5)
Shape of cv matrix after one hot encodig  (11055, 5)
```

In [221]:
```
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-ha
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna(" ")
```

In [222]:
```

my_counter = Counter()
for word in project_data['teacher_prefix'].values:
    my_counter.update(word.split())
```

In [223]:
```
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
teacher_cat_dict = dict(my_counter)
sorted_teacher_cat_dict = dict(sorted(teacher_cat_dict.items(), key=lambda k
```

In [224]:
```
#Using Count Vectorizer to convert the teacher_prefix value onto on hot enco
#ValueError: np.nan is an invalid document, expected byte or unicode string.
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-lear
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_cat_dict.keys())
vectorizer.fit(project_data['teacher_prefix'].values.astype('U'))
print(vectorizer.get_feature_names())

train_teacher_one_hot = vectorizer.transform(x_train['teacher_prefix'].value
test_teacher_one_hot = vectorizer.transform(x_test['teacher_prefix'].values.
cv_teacher_one_hot = vectorizer.transform(x_cv['teacher_prefix'].values.asty

print("Shape of Train matrix after one hot encodig ",train_teacher_one_hot.s
print("Shape of Test matrix after one hot encodig ",test_teacher_one_hot.sha
print("Shape of CV matrix after one hot encodig ",cv_teacher_one_hot.shape)

```

```
['Dr.', 'Teacher', 'Mr.', 'Ms.', 'Mrs.']
Shape of Train matrix after one hot encodig  (22445, 5)
Shape of Test matrix after one hot encodig  (16500, 5)
Shape of CV matrix after one hot encodig  (11055, 5)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words Project Essays

In [49]:
```
# We are considering only the words which appeared in at least 10 documents(
vectorizer = CountVectorizer(min_df=10)
vectorizer.fit(train_preprocessed_essays)

train_text_bow = vectorizer.transform(train_preprocessed_essays)
print("Shape of matrix after one hot encodig ",train_text_bow.shape)
```

```
Shape of matrix after one hot encodig  (22445, 8772)
```

In [225]:
```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it

test_text_bow = vectorizer.transform(test_preprocessed_essays)
print("Shape of matrix after one hot encodig ",test_text_bow.shape)
```

```
Shape of matrix after one hot encodig  (16500, 5)
```

In [51]:
```
# Vectrozing Cv
cv_text_bow = vectorizer.transform(cv_preprocessed_essays)
print("Shape of matrix after one hot encodig ",cv_text_bow.shape)

```

```
Shape of matrix after one hot encodig  (11055, 8772)
```

Project Titles

In [52]:
```
vectorizer.fit(train_preprocessed_titles)
train_title_bow = vectorizer.fit_transform(train_preprocessed_titles)
print("Shape of matrix after one hot encodig ",train_title_bow.shape)
```

```
Shape of matrix after one hot encodig  (22445, 1151)
```

In [53]:
```
test_title_bow = vectorizer.transform(test_preprocessed_titles)
print("Shape of matrix after one hot encodig ",test_title_bow.shape)
```

```
Shape of matrix after one hot encodig  (16500, 1151)
```

In [54]:
```
cv_title_bow = vectorizer.transform(cv_preprocessed_titles)
print("Shape of matrix after one hot encodig ",cv_title_bow.shape)
```

```
Shape of matrix after one hot encodig  (11055, 1151)
```

1.5.2.2 TFIDF vectorizer Project Essays

In [55]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(train_preprocessed_essays)

train_text_tfidf = vectorizer.transform(train_preprocessed_essays)
print("Shape of matrix after one hot encodig ",train_text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (22445, 8772)
```

In [56]:
```

test_text_tfidf = vectorizer.transform(test_preprocessed_essays)
print("Shape of matrix after one hot encodig ",test_text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (16500, 8772)
```

In [57]:
```

cv_text_tfidf = vectorizer.transform(cv_preprocessed_essays)
print("Shape of matrix after one hot encodig ",cv_text_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (11055, 8772)
```

Project Titles

In [58]:
```

vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(train_preprocessed_titles)

train_title_tfidf = vectorizer.transform(train_preprocessed_titles)
print("Shape of matrix after one hot encodig ",train_title_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (22445, 1151)
```

In [59]:
```
test_title_tfidf = vectorizer.transform(test_preprocessed_titles)
print("Shape of matrix after one hot encodig ",test_title_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (16500, 1151)
```

In [60]:
```
cv_title_tfidf = vectorizer.transform(cv_preprocessed_titles)
print("Shape of matrix after one hot encodig ",cv_title_tfidf.shape)
```

```
Shape of matrix after one hot encodig  (11055, 1151)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [61]:

```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

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

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

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

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

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

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

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


# stronging variables into pickle files python: http://www.jessicayung.com/h

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


'''
```

Out[61]: '\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084

```
039\ndef (https://stackoverflow.com/a/38230349/4084039\ndef) loadGloveModel(glo
veFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\', enco
ding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = lin
e.split()\n        word = splitLine[0]\n        embedding = np.array([float(va
l) for val in splitLine[1:]])\n        model[word] = embedding\n    print ("Don
e.",len(model)," words loaded!")\n    return model\nmodel = loadGloveModel(\'gl
ove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n# =
============================\n\nwords = []\nfor i in preproced_texts:\n    word
s.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n    words.extend(i.spli
t(\' \'))\nprint("all the words in the coupus", len(words))\nwords = set(words)
\nprint("the unique words in the coupus", len(words))\n\ninter_words = set(mode
l.keys()).intersection(words)\nprint("The number of words that are present in b
oth glove vectors and our coupus",       len(inter_words),"(",np.round(len(inte
r_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove = set(mode
l.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i]
 = model[i]\nprint("word 2 vec length", len(words_courpus))\n\n\n# stronging va
riables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-
to-save-and-load-variables-in-python/\n\nimport (http://www.jessicayung.com/how
-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport) pickle\nwith op
en(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [65]:

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

In [66]:

```
# average Word2Vec
# compute average word2vec for each review.
train_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(train_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    train_avg_w2v_vectors.append(vector)

print(len(train_avg_w2v_vectors))
print(len(train_avg_w2v_vectors[0]))
```

```
 75%|
| 16828/22445 [00:09<00:03, 1636.81it/s]
 76%|
| 17004/22445 [00:09<00:03, 1667.24it/s]
 77%|
| 17182/22445 [00:09<00:03, 1694.76it/s]
 77%|
| 17353/22445 [00:09<00:03, 1694.33it/s]
 78%|
| 17537/22445 [00:09<00:02, 1730.74it/s]
 79%|
| 17739/22445 [00:09<00:02, 1803.66it/s]
 80%|
| 17962/22445 [00:10<00:02, 1908.56it/s]
 81%|
| 18163/22445 [00:10<00:02, 1937.89it/s]
 82%|
| 18376/22445 [00:10<00:02, 1986.41it/s]
 83%|
| 18601/22445 [00:10<00:01, 2058.76it/s]
```

In [67]:

```
# average Word2Vec
# compute average word2vec for each review.
test_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(test_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_avg_w2v_vectors.append(vector)

print(len(test_avg_w2v_vectors))
print(len(test_avg_w2v_vectors[0]))
```

```
91%|
| 14976/16500 [00:08<00:00, 1755.78it/s]
 92%|
| 15152/16500 [00:08<00:00, 1751.89it/s]
 93%|
| 15334/16500 [00:08<00:00, 1766.70it/s]
 94%|
| 15531/16500 [00:08<00:00, 1818.18it/s]
 95%|
| 15735/16500 [00:08<00:00, 1874.39it/s]
 97%|
|   | 15924/16500 [00:08<00:00, 1793.56it/s]
 98%|
|   | 16105/16500 [00:08<00:00, 1772.13it/s]
 99%|
| | 16284/16500 [00:08<00:00, 1706.38it/s]
100%|
| | 16456/16500 [00:08<00:00, 1665.80it/s]
100%|
| 16500/16500 [00:08<00:00, 1840.57it/s]
```

In [68]:

```
# average Word2Vec
# compute average word2vec for each review.
cv_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(cv_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_avg_w2v_vectors.append(vector)

print(len(cv_avg_w2v_vectors))
print(len(cv_avg_w2v_vectors[0]))
```

```
 34%|
| 3807/11055 [00:01<00:03, 1924.65it/s]
 36%|
| 4015/11055 [00:02<00:03, 1957.75it/s]
 38%|
| 4225/11055 [00:02<00:03, 1992.76it/s]
 40%|
| 4441/11055 [00:02<00:03, 2034.47it/s]
 42%|
| 4646/11055 [00:02<00:03, 2033.14it/s]
 44%|
| 4869/11055 [00:02<00:02, 2082.69it/s]
 46%|
| 5089/11055 [00:02<00:02, 2110.50it/s]
 48%|
| 5301/11055 [00:02<00:02, 2082.35it/s]
 50%|
| 5512/11055 [00:02<00:02, 2084.48it/s]
 52%|
| 5721/11055 [00:02<00:02, 2002.30it/s]
```

Projects Titles

In [69]:

```
# average Word2Vec
# compute average word2vec for each review.
train_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is
for sentence in tqdm(train_preprocessed_titles): # 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
    train_title_avg_w2v_vectors.append(vector)

print(len(train_title_avg_w2v_vectors))
print(len(train_title_avg_w2v_vectors[0]))
```

```
  0%|
| 0/22445 [00:00<?, ?it/s]
 14%|
| 3190/22445 [00:00<00:00, 31589.51it/s]
 33%|
| 7360/22445 [00:00<00:00, 33985.36it/s]
 51%|
| 11428/22445 [00:00<00:00, 35657.90it/s]
 67%|
| 14958/22445 [00:00<00:00, 35444.39it/s]
 81%|                                                                    |
18159/22445 [00:00<00:00, 34230.85it/s]
 96%|
| 21615/22445 [00:00<00:00, 34228.34it/s]
100%|
22445/22445 [00:00<00:00, 34696.30it/s]

22445
300
```

In [70]:

```
# average Word2Vec
# compute average word2vec for each review.
test_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is s
for sentence in tqdm(test_preprocessed_titles): # 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
    test_title_avg_w2v_vectors.append(vector)

print(len(test_title_avg_w2v_vectors))
print(len(test_title_avg_w2v_vectors[0]))
```

```
  0%|
| 0/16500 [00:00<?, ?it/s]
 25%|
| 4179/16500 [00:00<00:00, 41381.78it/s]
 49%|
| 8028/16500 [00:00<00:00, 40344.39it/s]
 70%|
| 11580/16500 [00:00<00:00, 38640.69it/s]
 85%|
| 14044/16500 [00:00<00:00, 30787.81it/s]
100%|
16500/16500 [00:00<00:00, 33747.67it/s]

16500
300
```

In [71]:
```
# average Word2Vec
# compute average word2vec for each review.
cv_title_avg_w2v_vectors = []; # the avg-w2v for each sentence/review is sto
for sentence in tqdm(cv_preprocessed_titles): # 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
    cv_title_avg_w2v_vectors.append(vector)

print(len(cv_title_avg_w2v_vectors))
print(len(cv_title_avg_w2v_vectors[0]))
```

```
  0%|
| 0/11055 [00:00<?, ?it/s]
 37%|
| 4049/11055 [00:00<00:00, 40095.23it/s]
 69%|
| 7639/11055 [00:00<00:00, 38614.25it/s]
 99%|
| 10955/11055 [00:00<00:00, 36678.44it/s]
100%|
11055/11055 [00:00<00:00, 35552.12it/s]

11055
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [72]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(train_preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_
tfidf_words = set(tfidf_model.get_feature_names())
```

In [73]:

```
# average Word2Vec
# compute average word2vec for each review.
train_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review i
for sentence in tqdm(train_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/rev
    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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spl
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_essay_tfidf_w2v_vectors.append(vector)

print(len(train_essay_tfidf_w2v_vectors))
print(len(train_essay_tfidf_w2v_vectors[0]))
```

```
97%|
 | 21849/22445 [01:13<00:02, 225.50it/s]
97%|
 | 21873/22445 [01:13<00:02, 229.01it/s]
```

In [74]:

```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
test_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is
for sentence in tqdm(test_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/rev
    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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spl
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_essay_tfidf_w2v_vectors.append(vector)

print(len(test_essay_tfidf_w2v_vectors))
print(len(test_essay_tfidf_w2v_vectors[0]))
```

```
| 291/16500 [00:00<00:51, 312.34it/s]
  2%|
| 328/16500 [00:01<00:49, 325.08it/s]
  2%|
| 360/16500 [00:01<00:50, 317.78it/s]
  2%|
| 394/16500 [00:01<00:49, 323.22it/s]
  3%|
| 427/16500 [00:01<00:50, 317.73it/s]
  3%|
| 459/16500 [00:01<00:56, 282.20it/s]
  3%|
| 494/16500 [00:01<00:53, 298.86it/s]
  3%|
| 525/16500 [00:01<00:54, 291.89it/s]
  3%|
| 555/16500 [00:01<00:55, 285.06it/s]
  4%|
| 589/16500 [00:01<00:53, 298.02it/s]
  4%|
```

In [75]:

```
# average Word2Vec
# compute average word2vec for each review.
cv_essay_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is s
for sentence in tqdm(cv_preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/rev
    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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spl
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_essay_tfidf_w2v_vectors.append(vector)

print(len(cv_essay_tfidf_w2v_vectors))
print(len(cv_essay_tfidf_w2v_vectors[0]))
```

```
 95%|
| 10447/11055 [00:36<00:02, 245.62it/s]
 95%|
| 10473/11055 [00:37<00:02, 245.53it/s]
 95%|
|     | 10498/11055 [00:37<00:02, 245.42it/s]
 95%|
|     | 10523/11055 [00:37<00:02, 244.61it/s]
 95%|
| 10548/11055 [00:37<00:02, 235.77it/s]
 96%|
|     | 10574/11055 [00:37<00:01, 241.88it/s]
 96%|
|     | 10600/11055 [00:37<00:01, 245.66it/s]
 96%|
| 10629/11055 [00:37<00:01, 255.44it/s]
 96%|
```

Project Titles

In [76]:

```
# average Word2Vec
# compute average word2vec for each review.
train_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review i
for sentence in tqdm(train_preprocessed_titles): # 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/rev
    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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spl
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    train_title_tfidf_w2v_vectors.append(vector)

print(len(train_title_tfidf_w2v_vectors))
print(len(train_title_tfidf_w2v_vectors[0]))
```

```
  0%|
| 0/22445 [00:00<?, ?it/s]
  7%|
| 1491/22445 [00:00<00:01, 14764.53it/s]
 13%|
| 2849/22445 [00:00<00:01, 14343.19it/s]
 19%|
| 4372/22445 [00:00<00:01, 14557.00it/s]
 26%|
| 5772/22445 [00:00<00:01, 14341.79it/s]
 32%|
| 7268/22445 [00:00<00:01, 14480.29it/s]
 38%|
| 8481/22445 [00:00<00:01, 13639.48it/s]
 44%|
| 9929/22445 [00:00<00:00, 13842.04it/s]
 51%|
11411/22445 [00:00<00:00, 14081.94it/s]
 58%|
| 12972/22445 [00:00<00:00, 14468.33it/s]
 65%|
| 14645/22445 [00:01<00:00, 15039.81it/s]
 72%|
| 16189/22445 [00:01<00:00, 15113.91it/s]
 79%|
17680/22445 [00:01<00:00, 14696.89it/s]
 85%|
| 19182/22445 [00:01<00:00, 14749.40it/s]
 92%|
| 20742/22445 [00:01<00:00, 14952.31it/s]
 99%|
| 22232/22445 [00:01<00:00, 14629.33it/s]
100%|
22445/22445 [00:01<00:00, 14464.26it/s]
```

```
22445
300
```

In [77]:

```python
# average Word2Vec
# compute average word2vec for each review.
test_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is
for sentence in tqdm(test_preprocessed_titles): # 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/rev
    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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spl
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_title_tfidf_w2v_vectors.append(vector)

print(len(test_title_tfidf_w2v_vectors))
print(len(test_title_tfidf_w2v_vectors[0]))
```

```
  0%|
| 0/16500 [00:00<?, ?it/s]
  9%|
| 1451/16500 [00:00<00:01, 14368.53it/s]
 17%|
| 2869/16500 [00:00<00:00, 14268.92it/s]
 26%|
| 4229/16500 [00:00<00:00, 14018.68it/s]
 35%|
| 5742/16500 [00:00<00:00, 14294.51it/s]
 44%|
| 7260/16500 [00:00<00:00, 14508.13it/s]
 52%|
| 8515/16500 [00:00<00:00, 13814.34it/s]
 60%|
| 9950/16500 [00:00<00:00, 13930.80it/s]
 71%|
| 11671/16500 [00:00<00:00, 14738.00it/s]
 81%|                                                                  |
13344/16500 [00:00<00:00, 15242.83it/s]
 90%|                                                                  |
14831/16500 [00:01<00:00, 15120.45it/s]
 99%|
| 16315/16500 [00:01<00:00, 14722.64it/s]
100%|
16500/16500 [00:01<00:00, 14562.80it/s]

16500
300
```

In [78]:

```
# average Word2Vec
# compute average word2vec for each review.
cv_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is s
for sentence in tqdm(cv_preprocessed_titles): # 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/rev
    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
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.spl
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_title_tfidf_w2v_vectors.append(vector)

print(len(cv_title_tfidf_w2v_vectors))
print(len(cv_title_tfidf_w2v_vectors[0]))
```

```
  0%|
| 0/11055 [00:00<?, ?it/s]
 13%|
| 1411/11055 [00:00<00:00, 14040.59it/s]
 26%|
| 2851/11055 [00:00<00:00, 14105.64it/s]
 38%|
| 4252/11055 [00:00<00:00, 14035.12it/s]
 51%|
| 5638/11055 [00:00<00:00, 13940.56it/s]
 63%|
| 6939/11055 [00:00<00:00, 13605.46it/s]
 75%|
| 8278/11055 [00:00<00:00, 13499.71it/s]
 88%|
| 9684/11055 [00:00<00:00, 13623.98it/s]
 99%|
| 10960/11055 [00:00<00:00, 13311.67it/s]
100%|
11055/11055 [00:00<00:00, 13491.80it/s]

11055
300
```

1.5.3 Vectorizing Numerical features

In [80]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum
project_data = pd.merge(project_data, price_data, on='id', how='left')
print(price_data.head())
#print(project_data.columns)
print(x_train.columns)

```

```
        id    price  quantity
0  p000001   459.56         7
1  p000002   515.89        21
2  p000003   298.97         4
3  p000004  1113.69        98
4  p000005   485.99         8
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', 'clean_categories',
       'clean_subcategories', 'essay'],
      dtype='object')
```

In [81]:
```
#   - quantity : numerical (optinal)
#   - teacher_number_of_previously_posted_projects : numerical
#   - price : numerical
x_train = pd.merge(x_train, price_data, on = "id", how = "left")
#print(x_train.columns)
x_test = pd.merge(x_test, price_data, on = "id", how = "left")
x_cv = pd.merge(x_cv, price_data, on = "id", how = "left")
```

Standardize Price

In [ ]:
```
#ValueError: Input contains NaN, infinity or a value too large for dtype('fl
#https://stackoverflow.com/questions/37716699/how-to-hstack-several-sparse-m
#https://datascience.stackexchange.com/questions/11928/valueerror-input-cont
#x_train = x_train['price'].as_matrix().astype(np.float)
#x_train = x_train.fillna(x_train.mean())
```

In [82]:
```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# standardization sklearn: https://scikit-learn.org/stable/modules/generated
from sklearn.preprocessing import StandardScaler

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

price_scalar = StandardScaler()

price_scalar.fit(x_train['price'].values.reshape(-1,1)) # finding the mean a
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_

# Now standardize the data with above maen and variance.
train_price_standar = price_scalar.transform(x_train['price'].values.reshape
train_price_standar
```

```
Mean : 297.71562753397194, Standard deviation : 373.5890845438879
```

Out[82]:
```
array([[-0.37665883],
       [-0.77008574],
       [-0.61491526],
       ...,
       [-0.44617371],
       [-0.58967362],
       [-0.12881968]])
```

In [84]:
```
price_scalar.fit(x_test['price'].values.reshape(-1,1)) # finding the mean an
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_

# Now standardize the data with above maen and variance.
test_price_standar = price_scalar.transform(x_test['price'].values.reshape(-
test_price_standar
```

```
Mean : 301.40470666666664, Standard deviation : 387.6948484063583
```

Out[84]:
```
array([[ 0.09227694],
       [-0.54660181],
       [-0.00364902],
       ...,
       [ 1.88484654],
       [-0.32495842],
       [-0.45085641]])
```

In [85]:
```
price_scalar.fit(x_cv['price'].values.reshape(-1,1)) # finding the mean and
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_

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

```
Mean : 299.52771506105836, Standard deviation : 373.1478347400019
```

Out[85]:
```
array([[ 0.09227694],
       [-0.54660181],
       [-0.00364902],
       ...,
       [ 1.88484654],
       [-0.32495842],
       [-0.45085641]])
```

In [200]:
```
print(train_price_standar.shape, y_train.shape)
print(test_price_standar.shape, y_test.shape)
print(cv_price_standar.shape, y_cv.shape)
```

```
(22445, 1) (22445,)
(16500, 1) (1797,)
(11055, 1) (11055,)
```

Standardize Teacher previously posted Projects

In [87]:
```

warnings.filterwarnings("ignore")
price_scalar.fit(x_train['teacher_number_of_previously_posted_projects'].val
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_

# Now standardize the data with above maen and variance.
train_prev_proj_standar = price_scalar.transform(x_train['teacher_number_of_
train_prev_proj_standar
```

```
Mean : 11.66473602138561, Standard deviation : 29.394684894505467
```

Out[87]:
```
array([[ 0.75984022],
       [-0.22673269],
       [-0.32879196],
       ...,
       [-0.36281171],
       [-0.32879196],
       [-0.32879196]])
```

In [202]:
```
price_scalar.fit(x_test['teacher_number_of_previously_posted_projects'].valu
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_

# Now standardize the data with above maen and variance.
test_prev_proj_standar = price_scalar.transform(x_test['teacher_number_of_pr
test_prev_proj_standar
```

```
Mean : 10.824363636363636, Standard deviation : 26.94848396907338
```

Out[202]:
```
array([[-0.40166874],
       [-0.3645609 ],
       [-0.40166874],
       ...,
       [ 0.22916452],
       [-0.17902171],
       [-0.40166874]])
```

In [89]:
```
price_scalar.fit(x_cv['teacher_number_of_previously_posted_projects'].values
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_

# Now standardize the data with above maen and variance.
cv_prev_proj_standar = price_scalar.transform(x_cv['teacher_number_of_previo
cv_prev_proj_standar
```

```
Mean : 11.040253279059248, Standard deviation : 27.32928192601594
```

Out[89]:
```
array([[ 0.58397973],
       [-0.33079   ],
       [-0.36738079],
       ...,
       [-0.36738079],
       [-0.29419921],
       [ 0.14489026]])
```

In [203]:
```
print(train_prev_proj_standar.shape, y_train.shape)
print(test_prev_proj_standar.shape, y_test.shape)
print(cv_prev_proj_standar.shape, y_cv.shape)

```

```
(22445, 1) (22445,)
(16500, 1) (1797,)
(11055, 1) (11055,)
```

Standardize Quantity

In [91]:
```
price_scalar.fit(x_train['quantity'].values.reshape(-1,1)) # finding the mea
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_

# Now standardize the data with above maen and variance.
train_quantity_standar = price_scalar.transform(x_train['quantity'].values.r
train_quantity_standar
```

```
Mean : 17.001470260637113, Standard deviation : 26.03701886058425
```

Out[91]:
```
array([[-0.57615929],
       [ 0.1151641 ],
       [-0.26890445],
       ...,
       [-0.49934558],
       [-0.19209074],
       [-0.26890445]])
```

In [92]:
```
price_scalar.fit(x_test['quantity'].values.reshape(-1,1)) # finding the mean
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_

# Now standardize the data with above maen and variance.
test_quantity_standar = price_scalar.transform(x_test['quantity'].values.res
test_quantity_standar
```

```
Mean : 17.05290909090909, Standard deviation : 27.494020077501187
```

Out[92]:
```
array([[ 4.32628952],
       [-0.40201139],
       [-0.51112602],
       ...,
       [ 0.17993334],
       [-0.58386911],
       [-0.43838293]])
```

In [93]:
```
price_scalar.fit(x_cv['quantity'].values.reshape(-1,1)) # finding the mean a
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_

# Now standardize the data with above maen and variance.
cv_quantity_standar = price_scalar.transform(x_cv['quantity'].values.reshape
cv_quantity_standar
```

```
Mean : 17.173224785165083, Standard deviation : 27.30705061932881
```

Out[93]:
```
array([[ 1.27537667],
       [-0.04296417],
       [-0.5190317 ],
       ...,
       [-0.44579054],
       [ 0.39648278],
       [ 0.03027699]])
```

In [226]:
```
print(train_quantity_standar.shape, y_train.shape)
print(test_quantity_standar.shape, y_test.shape)
print(cv_quantity_standar.shape, y_cv.shape)

```

```
(22445, 1) (22445,)
(16500, 1) (16500,)
(11055, 1) (11055,)
```

1.5.4 Merging all the above features

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

In [227]:
```
#Categorical
print("Shape of Train matrix after  Trainone hot encodig ",train_categories_
print("Shape of test matrix after one hot encodig ",test_categories_one_hot.
print("Shape of cv_ matrix after  one hot encodig ",cv_categories_one_hot.sh

```

```
Shape of Train matrix after  Trainone hot encodig  (22445, 9)
Shape of test matrix after one hot encodig  (16500, 9)
Shape of cv_ matrix after  one hot encodig  (11055, 9)
```

In [228]:
```
#subcategorical
print("Shape of Train matrix after  Trainone hot encodig ",train_sub_categor
print("Shape of test matrix after one hot encodig ",test_sub_categories_one_
print("Shape of cv_ matrix after  one hot encodig ",cv_sub_categories_one_ho
```

```
Shape of Train matrix after  Trainone hot encodig  (22445, 30)
Shape of test matrix after one hot encodig  (16500, 30)
Shape of cv_ matrix after  one hot encodig  (11055, 30)
```

In [97]:
```
#State
print("Shape of Train matrix after one hot encodig ",train_state_one_hot.sha
print("Shape of Test matrix after one hot encodig ",test_state_one_hot.shape
print("Shape of CV matrix after one hot encodig ",cv_state_one_hot.shape)
```

```
Shape of Train matrix after one hot encodig  (22445, 51)
Shape of Test matrix after one hot encodig  (16500, 51)
Shape of CV matrix after one hot encodig  (11055, 51)
```

In [98]:
```
#Grade
print("Shape of Train matrix after one hot encodig ",train_grade_one_hot.sha
print("Shape of test matrix after one hot encodig ",test_grade_one_hot.shape
print("Shape of cv matrix after one hot encodig ",cv_grade_one_hot.shape)
```

```
Shape of Train matrix after one hot encodig  (22445, 5)
Shape of test matrix after one hot encodig  (16500, 5)
Shape of cv matrix after one hot encodig  (11055, 5)
```

In [99]:
```
#Teacher
print("Shape of Train matrix after one hot encodig ",train_teacher_one_hot.s
print("Shape of Test matrix after one hot encodig ",test_teacher_one_hot.sha
print("Shape of CV matrix after one hot encodig ",cv_teacher_one_hot.shape)
```

```
Shape of Train matrix after one hot encodig  (22445, 5)
Shape of Test matrix after one hot encodig  (16500, 5)
Shape of CV matrix after one hot encodig  (11055, 5)
```

In [100]:
```
#BOW Project Essays
print("Shape of train matrix after one hot encodig ",train_text_bow.shape)
print("Shape of test  matrix after one hot encodig ",test_text_bow.shape)
print("Shape of cv matrix after one hot encodig ",cv_text_bow.shape)
print("- "*50)
#BOW Project Titles
print("Shape of train matrix after one hot encodig ",train_title_bow.shape)
print("Shape of test matrix after one hot encodig ",test_title_bow.shape)
print("Shape of cv matrix after one hot encodig ",cv_title_bow.shape)
```

```
Shape of train matrix after one hot encodig  (22445, 8772)
Shape of test  matrix after one hot encodig  (16500, 8772)
Shape of cv matrix after one hot encodig  (11055, 8772)
- - - -
```

In [104]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
#TypeError: 'coo_matrix' object is not subscriptable https://stackoverflow.c
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a d
X_train1 = hstack((train_categories_one_hot,train_sub_categories_one_hot,tra
                  train_teacher_one_hot,train_text_bow, train_title_bow, tra
                  train_prev_proj_standar, train_price_standar)).tocsr()
print(X_train1.shape, y_train.shape)
print(type(X_train1))
```

```
(22445, 10026) (22445,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [109]:
```
#X_trainn = X_train.todense() #NOT Working
```

In [105]:
```
X_test1 = hstack((test_categories_one_hot,test_sub_categories_one_hot,test_s
                  test_teacher_one_hot, test_text_bow, test_title_bow, test_
                  test_prev_proj_standar, test_price_standar)).tocsr()
print(X_test1.shape, y_test.shape)
print(type(X_test1))
```

```
(16500, 10026) (16500,)
<class 'scipy.sparse.csr.csr_matrix'>
```

In [106]:
```
X_cv1 = hstack((cv_categories_one_hot,cv_sub_categories_one_hot,cv_state_one
                  cv_teacher_one_hot, cv_text_bow, cv_title_bow, cv_quantity
                  cv_prev_proj_standar, cv_price_standar)).tocsr()
print(X_cv1.shape, y_cv.shape)
print(type(X_cv1))
```

```
(11055, 10026) (11055,)
<class 'scipy.sparse.csr.csr_matrix'>
```

Type *Markdown* and LaTeX: α^2

In [ ]:
```
#from sklearn.neighbors import KNeighborsClassifier

#knn = KNeighborsClassifier(n_neighbors=50)
#knn.fit(X_train, y_train)
#pred1 = knn.predict(X_test)
```

In [ ]:
```

#pred = knn.predict(X_test)
#accuracy_score(y_test, pred)
```

Assignment 3: Apply KNN

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

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

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

2. **Hyper paramter tuning to find best K**

- Find the best hyper parameter which results in the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
- Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
- Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task

3. **Representation of results**

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
- Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
- Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

4. **[Task-2]**

- Select top 2000 features from feature Set 2 using `SelectKBest` (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) and then apply KNN on top of these features

```
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, chi2
X, y = load_digits(return_X_y=True)
X.shape
X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
X_new.shape
========
output:
(1797, 64)
(1797, 20)
```

- Repeat the steps 2 and 3 on the data matrix after feature selection

5. **Conclusion**

   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link (http://zetcode.com/python/prettytable/)

Note: Data Leakage

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

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

2.4.1 Applying KNN brute force on BOW_SET-1

In [ ]:
```
#from sklearn.neighbors import KNeighborsClassifier
#knn = KNeighborsClassifier(n_neighbors=50)
#knn.fit(X_train, y_train)
#pred1 = knn.predict(X_test)
```

In [229]:
```
print(X_train1.shape, y_train.shape)
print(X_test1.shape, y_test.shape)
print(X_cv1.shape, y_cv.shape)
```

```
(22445, 10026) (22445,)
(16500, 10026) (16500,)
(11055, 10026) (11055,)
```

In [ ]:

```
#myList = list(range(1,50))

# subsetting just the odd ones
#neighbors = filter(lambda x: x % 2 != 0, myList)

# empty list that will hold cv scores
#cv_scores = []

# perform 10-fold cross validation
#for k in neighbors:
#   knn = KNeighborsClassifier(n_neighbors=k)
#    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accurac
#    cv_scores.append(scores.mean())
```

In [ ]:

```

```

In [108]:

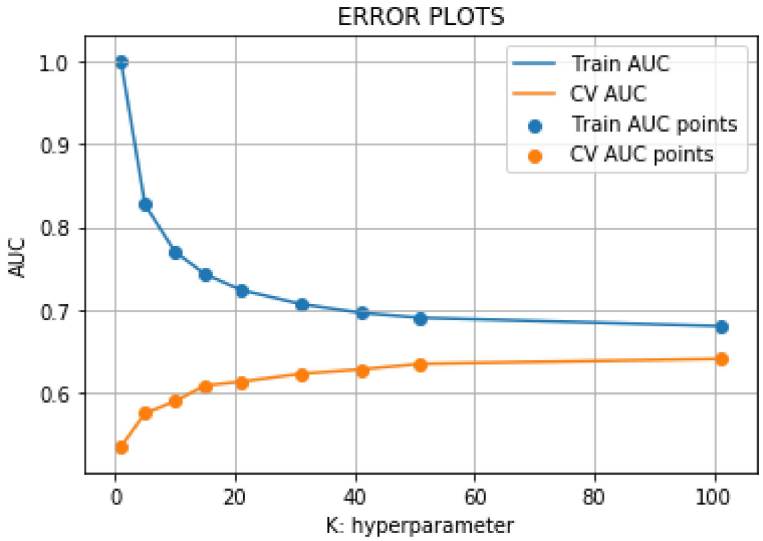
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability
    # not the predicted outputs

    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 - 49
    # 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 [109]:

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

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, co
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with gre

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train1, y_train)

    y_train_pred = batch_predict(neigh, X_train1)
    y_cv_pred = batch_predict(neigh, X_cv1)

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

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

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

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

```
  0%|
| 0/9 [00:00<?, ?it/s]
 11%|
| 1/9 [01:30<12:05, 90.70s/it]
 22%|
| 2/9 [03:09<10:52, 93.20s/it]
 33%|
| 3/9 [04:53<09:38, 96.38s/it]
 44%|
| 4/9 [06:31<08:04, 96.99s/it]
 56%|
| 5/9 [08:10<06:29, 97.41s/it]
 67%|
| 6/9 [09:49<04:53, 97.79s/it]
```

```
 78%|
| 7/9 [11:28<03:16, 98.36s/it]
 89%|
|          | 8/9 [13:06<01:38, 98.23s/it]
100%|
          | 9/9 [14:44<00:00, 98.22s/it]
```



Grid Search

In [137]:

```
from sklearn.model_selection import GridSearchCV

```

In [ ]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.
#from sklearn.model_selection import GridSearchCV
#neigh = KNeighborsClassifier()
#parameters = {'n_neighbors':[1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 107, 12
#clf = GridSearchCV(neigh, parameters, cv= 5, scoring='roc_auc')
#clf.fit(X_train1, y_train)

#train_auc= clf.cv_results_['mean_train_score']
#train_auc_std= clf.cv_results_['std_train_score']
#cv_auc = clf.cv_results_['mean_test_score']
#cv_auc_std= clf.cv_results_['std_test_score']

#plt.plot(parameters['n_neighbors'], train_auc, label='Train AUC')
## this code is copied from here: https://stackoverflow.com/a/48803361/40840
#plt.gca().fill_between(parameters['n_neighbors'],train_auc - train_auc_std,

#plt.plot(parameters['n_neighbors'], cv_auc, label='CV AUC')
## this code is copied from here: https://stackoverflow.com/a/48803361/40840
#plt.gca().fill_between(parameters['n_neighbors'],cv_auc - cv_auc_std,cv_auc

#plt.scatter(parameters['n_neighbors'], train_auc, label='Train AUC points')
#plt.scatter(parameters['n_neighbors'], cv_auc, label='CV AUC points')
#plt.legend()
#plt.xlabel("K: hyperparameter")
#plt.ylabel("AUC")
#plt.title("ERROR PLOTS")
#plt.grid()
#plt.show()
```

Fitting Model to Hyper-Parameter Curve

TO Reduce the noise in the Pridcting Values Here we are Picking the Highest k Values

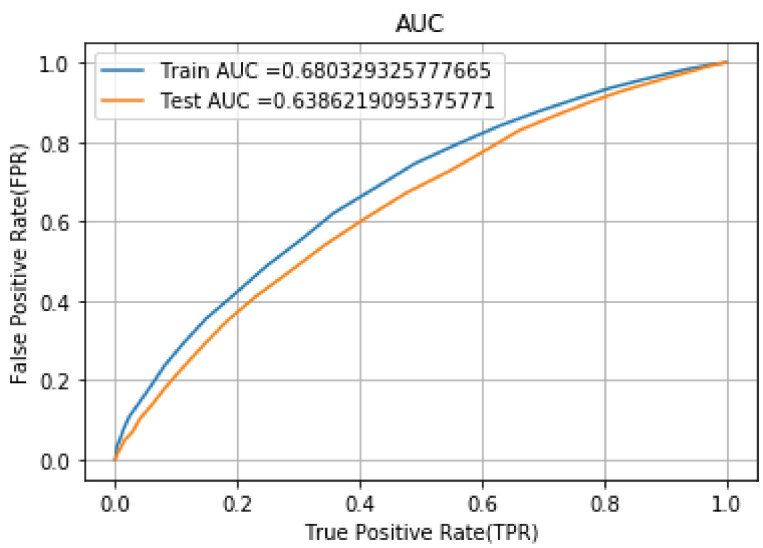
In [122]:

```
best_k_1 = 97
print(X_test1.shape)
```

```
(16500, 10026)
```

In [123]:

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


neigh = KNeighborsClassifier(n_neighbors=best_k_1)
neigh.fit(X_train1, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability est
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train1)
y_test_pred = batch_predict(neigh, X_test1)

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


```



In [ ]:

```

```

In [113]:
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshould, fpr, tpr):

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

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

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

Confusion Matrix

In [114]:
```
print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_f

```

```
====================================================================================
=====================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2498422121600828 for threshold 0.769
[[ 1688  1775]
 [ 4508 14474]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999984572938835 for threshold 0.769
[[ 1099  1447]
 [ 3555 10399]]
```

Test Data

In [115]:
```
#This is Future Unseen Data
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999984572938835 for threshold 0.769
[[ 1099  1447]
 [ 3555 10399]]
```

In [ ]:
```

```

2.4.2 Applying KNN brute force on TFIDF_SET-2

In [120]:

```
X_train2 = hstack((train_categories_one_hot,train_sub_categories_one_hot,tra
                  train_teacher_one_hot,train_text_tfidf, train_title_tfidf,
                  train_prev_proj_standar, train_price_standar)).tocsr()
X_test2 = hstack((test_categories_one_hot,test_sub_categories_one_hot,test_s
                  test_teacher_one_hot, test_text_tfidf, test_title_tfidf, t
                  test_prev_proj_standar, test_price_standar)).tocsr()
X_cv2 = hstack((cv_categories_one_hot,cv_sub_categories_one_hot,cv_state_one
                  cv_teacher_one_hot, cv_text_tfidf, cv_title_tfidf, cv_quan
                  cv_prev_proj_standar, cv_price_standar)).tocsr()
print(X_train2.shape, y_train.shape)
print(X_test2.shape, y_test.shape)
print(X_cv2.shape, y_cv.shape)
```

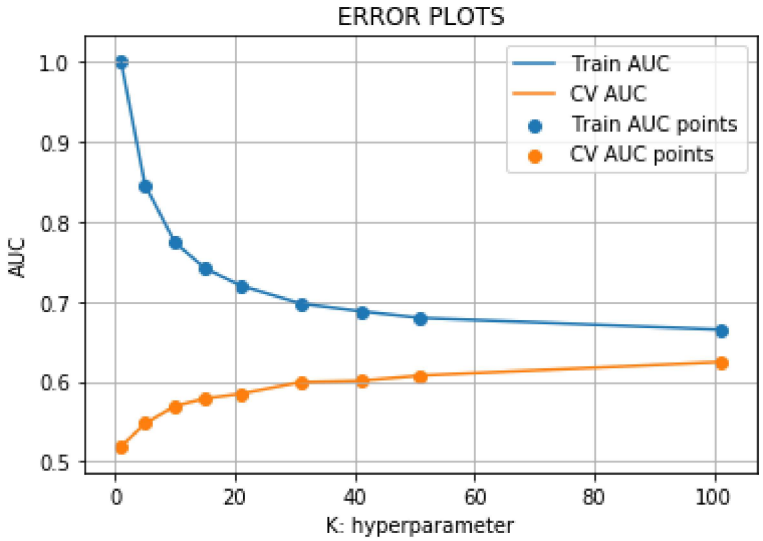
```
(22445, 10026) (22445,)
(16500, 10026) (16500,)
(11055, 10026) (11055,)
```

In [121]:

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

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, co
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with gre

"""

train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train2, y_train)

    y_train_pred = batch_predict(neigh, X_train2)
    y_cv_pred = batch_predict(neigh, X_cv2)

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

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

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

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

```
  0%|
| 0/9 [00:00<?, ?it/s]
 11%|
| 1/9 [01:31<12:12, 91.60s/it]
 22%|
| 2/9 [03:17<11:11, 96.00s/it]
 33%|
| 3/9 [05:05<09:57, 99.61s/it]
 44%|
| 4/9 [06:57<08:35, 103.08s/it]
 56%|
| 5/9 [08:45<06:58, 104.57s/it]
 67%|
| 6/9 [10:30<05:14, 104.79s/it]
```

```
 78%|
| 7/9 [12:04<03:23, 101.53s/it]
 89%|
|          | 8/9 [13:46<01:41, 101.79s/it]
100%|
| 9/9 [15:26<00:00, 101.29s/it]
```

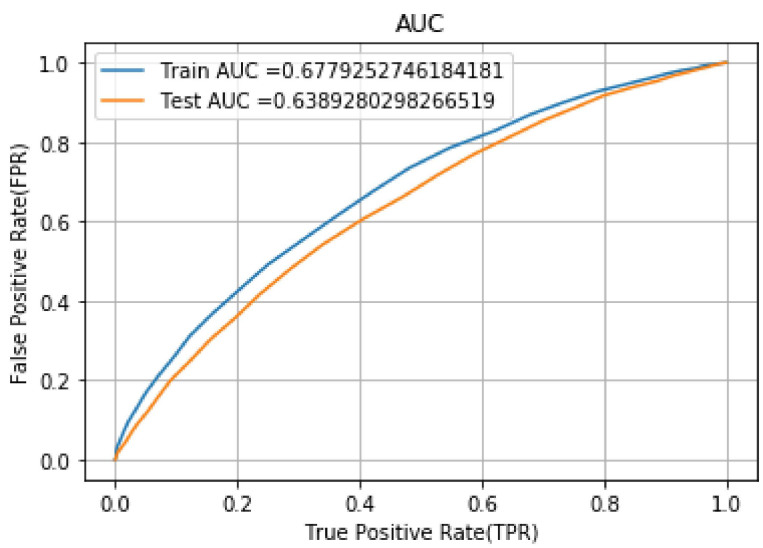


Fitting Model to Hyperparameter Curve

In [126]:

```
best_k_2 = 107
# The Highest value of K can neglect the noise and outliers
```

In [127]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curv
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k_2)
neigh.fit(X_train1, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability est
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train1)
y_test_pred = batch_predict(neigh, X_test1)

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


```



Confusion Matix

In [128]:
```
print("="*100)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_f

```

```
====================================================================================
=====================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.2496422516685402 for threshold 0.776
[[ 1797  1666]
 [ 5043 13939]]
```

Test Data

In [129]:
```
#This is Future Unseen Data
print("Test confusion matrix for Test Data")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24915521413069439 for threshold 0.785
[[1347 1199]
 [4716 9238]]
```

Type *Markdown* and LaTeX: α^2

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

2.4.3 Applying KNN brute force on AVG W2V_SET-3

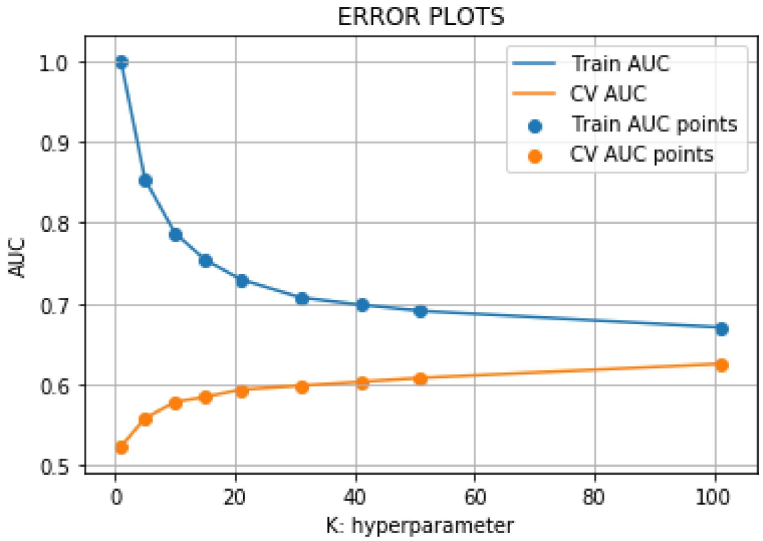
In [148]:
```
#train_avg_w2v_vectors
X_train3 = hstack((train_categories_one_hot,train_sub_categories_one_hot,tra
                  train_teacher_one_hot,train_avg_w2v_vectors, train_title_a
                  train_prev_proj_standar, train_price_standar)).tocsr()
X_test3 = hstack((test_categories_one_hot,test_sub_categories_one_hot,test_s
                  test_teacher_one_hot, test_avg_w2v_vectors, test_title_avg
                  test_prev_proj_standar, test_price_standar)).tocsr()
X_cv3 = hstack((cv_categories_one_hot,cv_sub_categories_one_hot,cv_state_one
                  cv_teacher_one_hot, cv_avg_w2v_vectors, cv_title_avg_w2v_v
                  cv_prev_proj_standar, cv_price_standar)).tocsr()
print(X_train3.shape, y_train.shape)
print(X_test3.shape, y_test.shape)
print(X_cv3.shape, y_cv.shape)
```

```
(22445, 703) (22445,)
(16500, 703) (16500,)
(11055, 703) (11055,)
```

In [132]:

```
#train_avg_w2v_vectors
#test_avg_w2v_vectors
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train3, y_train)

    y_train_pred = batch_predict(neigh, X_train3)
    y_cv_pred = batch_predict(neigh, X_cv3)

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

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

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

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

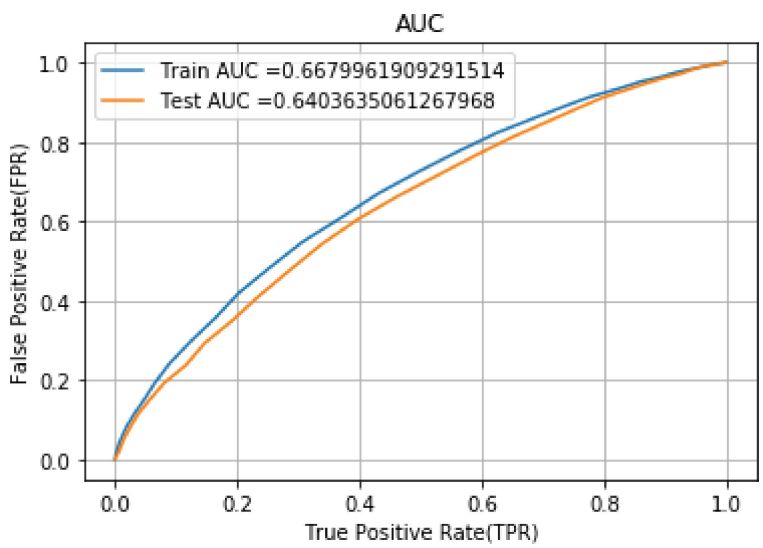
```
  0%|
| 0/9 [00:00<?, ?it/s]
 11%|
| 1/9 [19:11<2:33:33, 1151.75s/it]
 22%|
| 2/9 [37:28<2:12:26, 1135.21s/it]
 33%|
| 3/9 [55:45<1:52:22, 1123.82s/it]
 44%|
| 4/9 [1:14:02<1:32:59, 1115.83s/it]
 56%|
| 5/9 [1:32:20<1:14:01, 1110.26s/it]
 67%|
| 6/9 [1:50:39<55:20, 1106.97s/it]
 78%|
| 7/9 [2:08:57<36:48, 1104.30s/it]
 89%|
| 8/9 [2:27:15<18:22, 1102.43s/it]
100%|
| 9/9 [2:45:36<00:00, 1101.93s/it]
```



Fitting Model to HyperParameter Curve

In [137]:
```
best_k_3 = 121
# The Highest value of K can neglect the noise and outliers
```

In [138]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curv
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k_3)
neigh.fit(X_train3, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability est
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train3)
y_test_pred = batch_predict(neigh, X_test3)

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



Confusion Matrix

In [139]:
```
print("="*100)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_f

```

```
==========================================================================
=====================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999080665379794 for threshold 0.835
[[ 1742  1721]
 [ 5208 13774]]
```

Confusion Matrix for Test Data

In [140]:
```
#This is Future Unseen Data
print("Test confusion matrix for Test Data")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,
```

```
Test confusion matrix for Test Data
the maximum value of tpr*(1-fpr) 0.24913222780956096 for threshold 0.835
[[ 1198  1348]
 [ 3944 10010]]
```

In [ ]:
```

```

2.4.4 Applying KNN brute force on TFIDF W2V_SET-4

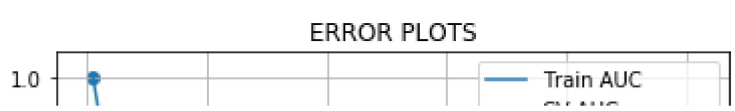
In [141]:
```
X_train4 = hstack((train_categories_one_hot,train_sub_categories_one_hot,tra
                  train_teacher_one_hot, train_essay_tfidf_w2v_vectors, trai
                  train_prev_proj_standar, train_price_standar)).tocsr()
X_test4 = hstack((test_categories_one_hot,test_sub_categories_one_hot,test_s
                  test_teacher_one_hot, test_essay_tfidf_w2v_vectors, test_t
                  test_prev_proj_standar, test_price_standar)).tocsr()
X_cv4 = hstack((cv_categories_one_hot,cv_sub_categories_one_hot,cv_state_one
                  cv_teacher_one_hot, cv_essay_tfidf_w2v_vectors, cv_title_t
                  cv_prev_proj_standar, cv_price_standar)).tocsr()
print(X_train4.shape, y_train.shape)
print(X_test4.shape, y_test.shape)
print(X_cv4.shape, y_cv.shape)
```

```
(22445, 703) (22445,)
(16500, 703) (16500,)
(11055, 703) (11055,)
```

In [150]:

```
#train_essay_tfidf_w2v_vectors
#test_essay_tfidf_w2v_vectors
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 81, 91, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train4, y_train)

    y_train_pred = batch_predict(neigh, X_train4)
    y_cv_pred = batch_predict(neigh, X_cv1)

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

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

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

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

```
 73%|
| 8/11 [12:12<04:41, 93.86s/it]

 82%|
| 9/11 [13:43<03:06, 93.24s/it]

 91%|
        | 10/11 [15:19<01:33, 93.86s/it]

100%|
| 11/11 [16:53<00:00, 93.89s/it]
```

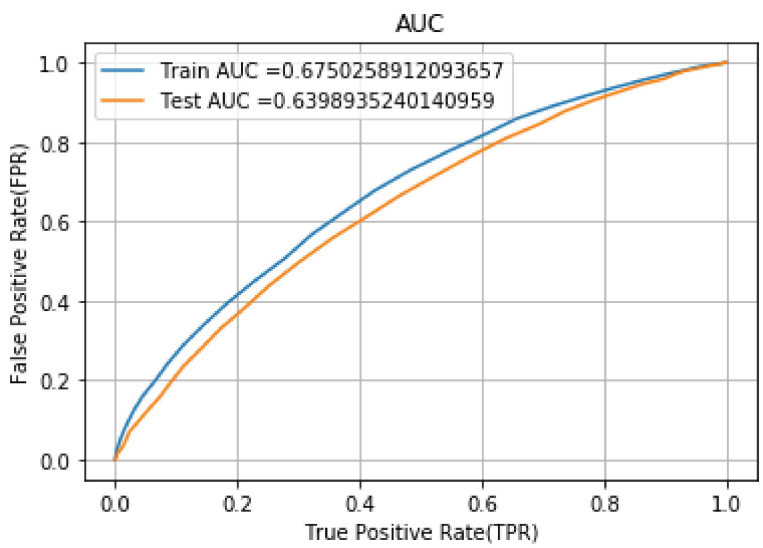


Fitting Model to HyperParameter Curve

In [152]:

```
best_k_4 = 121
# The Highest value of K can neglect the noise and outliers
```

In [153]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curv
from sklearn.metrics import roc_curve, auc


neigh = KNeighborsClassifier(n_neighbors=best_k_4)
neigh.fit(X_train4, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability est
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train4)
y_test_pred = batch_predict(neigh, X_test4)

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


```



In [ ]:
```

```

Confusion Matrix

In [154]:
```
print("="*100)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_f

```

```
====================================================================================
=====================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24972430384688798 for threshold 0.777
[[ 1789  1674]
 [ 5137 13845]]
```

Confusion Matix for Test Data

In [155]:
```
#This is Future Unseen Data
print("Test confusion matrix for Test Data")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,
```

```
Test confusion matrix for Test Data
the maximum value of tpr*(1-fpr) 0.24946298400090341 for threshold 0.777
[[1214 1332]
 [3982 9972]]
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

2.5 Feature selection with `SelectKBest`

In [197]:
```
X_train2 = hstack((train_categories_one_hot,train_sub_categories_one_hot,tra
                  train_teacher_one_hot,train_text_tfidf, train_title_tfidf,
                  train_prev_proj_standar, train_price_standar)).tocsr()
X_test2 = hstack((test_categories_one_hot,test_sub_categories_one_hot,test_s
                  test_teacher_one_hot, test_text_tfidf, test_title_tfidf, t
                  test_prev_proj_standar, test_price_standar)).tocsr()
X_cv2 = hstack((cv_categories_one_hot,cv_sub_categories_one_hot,cv_state_one
                  cv_teacher_one_hot, cv_text_tfidf, cv_title_tfidf, cv_quan
                  cv_prev_proj_standar, cv_price_standar)).tocsr()
print(X_train2.shape, y_train.shape)
print(X_test2.shape, y_test.shape)
print(X_cv2.shape, y_cv.shape)
```

```
(22445, 10026) (22445,)
(16500, 10026) (1797,)
(11055, 10026) (11055,)
```

In [199]:
```
##ValueError: Input X must be non-negative.
##https://stackoverflow.com/questions/25792012/feature-selection-using-sciki

#from sklearn.datasets import load_digits
#X_train2, y_test = load_digits(return_X_y=True)
#y_test1 = y_test.iloc[0]
#print(type(y_train1))
#print(y_train1)
print(X_train2.shape, y_train.shape)
```

```
(22445, 10026) (22445,)
```

Type *Markdown* and LaTeX: α^2

In [231]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

##ValueError: Input X must be non-negative.
##https://stackoverflow.com/questions/25792012/feature-selection-using-sciki
X_train2_new = SelectKBest(f_classif, k=2000).fit_transform(X_train2, y_trai
X_test2_new = SelectKBest(f_classif, k=2000).fit_transform(X_test2, y_test)
X_cv2_new = SelectKBest(f_classif, k=2000).fit_transform(X_cv2, y_cv)
```

In [232]:

```
#train_essay_tfidf_w2v_vectors
#test_essay_tfidf_w2v_vectors
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 81, 91, 101]
for i in tqdm(K):
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(X_train2_new, y_train)

    y_train_pred = batch_predict(neigh, X_train2_new)
    y_cv_pred = batch_predict(neigh, X_cv2_new)

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

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

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

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

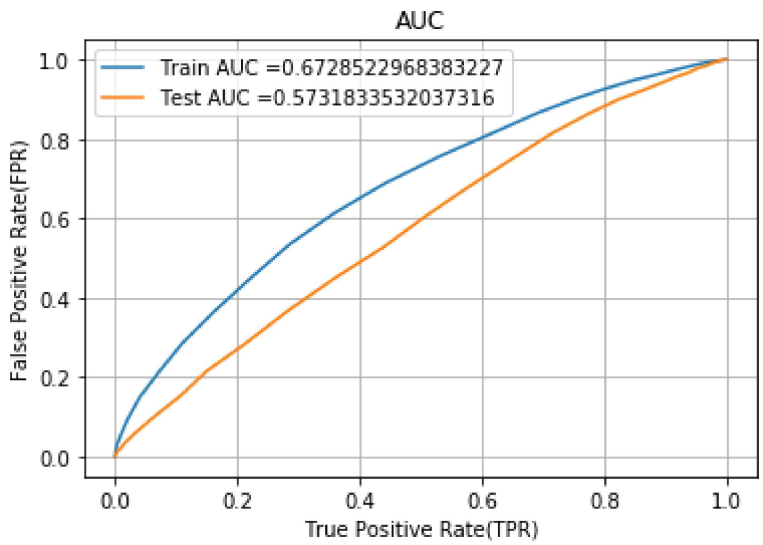
```
  0%|
| 0/11 [00:00<?, ?it/s]

  9%|
| 1/11 [00:52<08:44, 52.48s/it]

 18%|
| 2/11 [01:52<08:12, 54.76s/it]

 27%|
| 3/11 [02:53<07:32, 56.54s/it]
```

Fitting to HyperParameter

In [237]:

```
best_K_5 = 131
```

In [242]:
```
neigh = KNeighborsClassifier(n_neighbors=91)
neigh.fit(X_train2_new, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability est
# not the predicted outputs

y_train_pred = batch_predict(neigh, X_train2_new)
y_test_pred = batch_predict(neigh, X_test2_new)

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

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



Confusion Matrix

In [243]:
```
print("="*100)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_f
```

```
====================================================================================
=====================
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24901835492220095 for threshold 0.824
[[ 1623  1840]
 [ 4618 14364]]
```

Confusion Matirx on Test Data

In [244]:
```
#This is Future Unseen Data
print("Test confusion matrix for Test Data")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr,
```

```
Test confusion matrix for Test Data
the maximum value of tpr*(1-fpr) 0.24993196666026957 for threshold 0.835
[[1252 1294]
 [5501 8453]]
```

3. Conclusions

In [263]:
```
# Please compare all your models using Prettytable library
#how to use pretty table http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

tb = PrettyTable()
tb.field_names= ("Vectorizer", "Model", "HyperParameter", "AUC")
tb.add_row(["BOW", "Auto", 97, 63])
tb.add_row(["Tf-Idf", "Auto", 107, 65])
tb.add_row(["AVG-W2v", "Auto", 121, 64])
tb.add_row(["Tf-Idf W2v", "Auto", 121, 64])
tb.add_row(["Tf-Idf KBest", "Auto", 91, 57])
print(tb.get_string(titles = "KNN - Observations"))
#print(tb)
```

```
+--------------+-------+----------------+-----+
|  Vectorizer  | Model | HyperParameter | AUC |
+--------------+-------+----------------+-----+
|     BOW      |  Auto |       97       |  63 |
|    Tf-Idf    |  Auto |      107       |  65 |
|   AVG-W2v    |  Auto |      121       |  64 |
|  Tf-Idf W2v  |  Auto |      121       |  64 |
| Tf-Idf KBest |  Auto |       91       |  57 |
+--------------+-------+----------------+-----+
```

In [ ]:
```

```