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 the • `Art Will` •
`project_grade_category`	Grade level of students for which the project is targeted. • • • •

Feature	
project_subject_categories	One or more (comma-separated) subject categories fo following enum • Lit • Literacy & Language
school_state	State where school is located (Two-le (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviati
project_subject_subcategories	One or more (comma-separated) subject subcateg • Literature & Writing,
project_resource_summary	An explanation of the resources needed for th • My students need hands on literacy mate sens
project_essay_1	Fi
project_essay_2	Seco
project_essay_3	Thi
project_essay_4	Four
project_submitted_datetime	Datetime when project application was submitted. **Exa**
teacher_id	A unique identifier for the teacher of the propose bdf8baa8fedef6bf
teacher_prefix	Teacher's title. One of the following
teacher_number_of_previously_posted_projects	Number of project applications previously submitted

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

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

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

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

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

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

Notes on the Essay Data

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

- **project_essay_1:** "Introduce us to your classroom"
- **project_essay_2:** "Tell us more about your students"
- **project_essay_3:** "Describe how your students will use the materials you're requesting"
- **project_essay_3:** "Close by sharing why your project will make a difference"

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

- **project_essay_1:** "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- **project_essay_2:** "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

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

In [1]:
```
# Note - several code snippets have been used from the following link: https://co
# This link was provided by the Appliedai team to answer a question about data le
```

In [2]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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

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

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

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

from tqdm import tqdm
import os

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 [3]:
```
#Use all records and test running time
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [4]:
```
# search: how to randomly select rows from a pandas dataset  --> https://www.geeks


#Use only 50K
#project_data = pd.read_csv('train_data.csv')
#project_data = project_data.sample(n = 50000, replace = False)

#resource_data = pd.read_csv('resources.csv')
#filter only the records that match the project_id from the 50K records
#resource_data = resource_data[resource_data['id'].isin(project_data['id'])]
```

In [5]:
```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' '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']
```

In [6]:
```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_da


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


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084
project_data = project_data[cols]


project_data.head(2)
```

Out[6]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Dat
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	201 04-2 00:27:3
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	201 04-2 00:31:2

In [7]:
```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[7]:

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

1.2 preprocessing of project_subject_categories

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

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

1.3 preprocessing of project_subject_subcategories

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

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

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

1.3 Text preprocessing

In [10]:
```
# 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 [11]:
```
project_data.head(2)
```

Out[11]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Dat
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	201(04-2 00:27:3
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	201(04-2 00:31:2

1.4.2.3 Using Pretrained Models: TFIDF weighted W2V

In [12]:
```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classroom
as well as the STEM journals, which my students really enjoyed.  I would love
to implement more of the Lakeshore STEM kits in my classroom for the next sch
ool year as they provide excellent and engaging STEM lessons.My students come
from a variety of backgrounds, including language and socioeconomic status.
Many of them don't have a lot of experience in science and engineering and th
ese kits give me the materials to provide these exciting opportunities for my
students.Each month I try to do several science or STEM/STEAM projects.  I wo
uld use the kits and robot to help guide my science instruction in engaging a
nd meaningful ways.  I can adapt the kits to my current language arts pacing
guide where we already teach some of the material in the kits like tall tales
(Paul Bunyan) or Johnny Appleseed.  The following units will be taught in the
next school year where I will implement these kits: magnets, motion, sink vs.
float, robots.  I often get to these units and don't know If I am teaching th
e right way or using the right materials.    The kits will give me additional
ideas, strategies, and lessons to prepare my students in science.It is challe
nging to develop high quality science activities.  These kits give me the mat
erials I need to provide my students with science activities that will go alo
ng with the curriculum in my classroom.  Although I have some things (like ma
gnets) in my classroom, I don't know how to use them effectively.  The kits w
ill provide me with the right amount of materials and show me how to use them
in an appropriate way.
==================================================
I teach high school English to students with learning and behavioral disabili
ties. My students all vary in their ability level. However, the ultimate goal
is to increase all students literacy levels. This includes their reading, wri
ting, and communication levels.I teach a really dynamic group of students. Ho
wever, my students face a lot of challenges. My students all live in poverty
and in a dangerous neighborhood. Despite these challenges, I have students wh
o have the the desire to defeat these challenges. My students all have learni
ng disabilities and currently all are performing below grade level. My studen
ts are visual learners and will benefit from a classroom that fulfills their
preferred learning style.The materials I am requesting will allow my students
to be prepared for the classroom with the necessary supplies.  Too often I am
challenged with students who come to school unprepared for class due to econo
mic challenges.  I want my students to be able to focus on learning and not h
ow they will be able to get school supplies.  The supplies will last all yea
r.  Students will be able to complete written assignments and maintain a clas
sroom journal.  The chart paper will be used to make learning more visual in
class and to create posters to aid students in their learning.  The students
have access to a classroom printer.  The toner will be used to print student
work that is completed on the classroom Chromebooks.I want to try and remove
all barriers for the students learning and create opportunities for learning.
One of the biggest barriers is the students not having the resources to get p
```

ens, paper, and folders. My students will be able to increase their literacy skills because of this project.
==========
\"Life moves pretty fast. If you don't stop and look around once in awhile, you could miss it.\" from the movie, Ferris Bueller's Day Off. Think back... what do you remember about your grandparents? How amazing would it be to be able to flip through a book to see a day in their lives?My second graders are voracious readers! They love to read both fiction and nonfiction books. Their favorite characters include Pete the Cat, Fly Guy, Piggie and Elephant, and Mercy Watson. They also love to read about insects, space and plants. My students are hungry bookworms! My students are eager to learn and read about the world around them. My kids love to be at school and are like little sponges absorbing everything around them. Their parents work long hours and usually do not see their children. My students are usually cared for by their grandparents or a family friend. Most of my students do not have someone who speaks English at home. Thus it is difficult for my students to acquire language.Now think forward... wouldn't it mean a lot to your kids, nieces or nephews or grandchildren, to be able to see a day in your life today 30 years from now? Memories are so precious to us and being able to share these memories with future generations will be a rewarding experience. As part of our social studies curriculum, students will be learning about changes over time. Students will be studying photos to learn about how their community has changed over time.
In particular, we will look at photos to study how the land, buildings, clothing, and schools have changed over time. As a culminating activity, my students will capture a slice of their history and preserve it through scrap booking. Key important events in their young lives will be documented with the date, location, and names. Students will be using photos from home and from school to create their second grade memories. Their scrap books will preserve their unique stories for future generations to enjoy.Your donation to this project will provide my second graders with an opportunity to learn about social studies in a fun and creative manner. Through their scrapbooks, children will share their story with others and have a historical document for the rest of their lives.
==========
\"A person's a person, no matter how small.\" (Dr.Seuss) I teach the smallest students with the biggest enthusiasm for learning. My students learn in many different ways using all of our senses and multiple intelligences. I use a wide range of techniques to help all my students succeed. \r\nStudents in my class come from a variety of different backgrounds which makes for wonderful sharing of experiences and cultures, including Native Americans.\r\nOur school is a caring community of successful learners which can be seen through collaborative student project based learning in and out of the classroom. Kindergarteners in my class love to work with hands-on materials and have many different opportunities to practice a skill before it is mastered. Having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is the perfect place to learn about agriculture and nutrition. My students love to role play in our pretend kitchen in the early childhood classroom. I have had several kids ask me, \"Can we try cooking with REAL food?\" I will take their idea and create \"Common Core Cooking Lessons\" where we learn important math and writing concepts while cooking delicious healthy food for snack time. My students will have a grounded appreciation for the work that went into making the food and knowledge of where the ingredients came from as well as how it's healthy for their bodies. This project would expand our learning of nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classroom garden in the spring. We will also create our own cookbooks to be printed and shared with families. \r\nStudents w

ill gain math and literature skills as well as a life long enjoyment for heal thy cooking.nannan
==========
My classroom consists of twenty-two amazing sixth graders from different cult ures and backgrounds. They are a social bunch who enjoy working in partners a nd working with groups. They are hard-working and eager to head to middle sch ool next year. My job is to get them ready to make this transition and make i t as smooth as possible. In order to do this, my students need to come to sch ool every day and feel safe and ready to learn. Because they are getting read y to head to middle school, I give them lots of choice- choice on where to si t and work, the order to complete assignments, choice of projects, etc. Part of the students feeling safe is the ability for them to come into a welcomin g, encouraging environment. My room is colorful and the atmosphere is casual. I want them to take ownership of the classroom because we ALL share it togeth er. Because my time with them is limited, I want to ensure they get the most of this time and enjoy it to the best of their abilities.Currently, we have t wenty-two desks of differing sizes, yet the desks are similar to the ones the students will use in middle school. We also have a kidney table with crates f or seating. I allow my students to choose their own spots while they are work ing independently or in groups. More often than not, most of them move out of their desks and onto the crates. Believe it or not, this has proven to be mor e successful than making them stay at their desks! It is because of this that I am looking toward the “Flexible Seating” option for my classroom.\r\n The s tudents look forward to their work time so they can move around the room. I w ould like to get rid of the constricting desks and move toward more “fun” sea ting options. I am requesting various seating so my students have more option s to sit. Currently, I have a stool and a papasan chair I inherited from the previous sixth-grade teacher as well as five milk crate seats I made, but I w ould like to give them more options and reduce the competition for the “good seats”. I am also requesting two rugs as not only more seating options but to make the classroom more welcoming and appealing. In order for my students to be able to write and complete work without desks, I am requesting a class set of clipboards. Finally, due to curriculum that requires groups to work togeth er, I am requesting tables that we can fold up when we are not using them to leave more room for our flexible seating options.\r\nI know that with more se ating options, they will be that much more excited about coming to school! Th ank you for your support in making my classroom one students will remember fo rever!nannan
==========

In [13]:
```
# 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 [14]:
```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

```
\"A person is a person, no matter how small.\" (Dr.Seuss) I teach the smallest
students with the biggest enthusiasm for learning. My students learn in many di
fferent ways using all of our senses and multiple intelligences. I use a wide r
ange of techniques to help all my students succeed. \r\nStudents in my class co
me from a variety of different backgrounds which makes for wonderful sharing of
experiences and cultures, including Native Americans.\r\nOur school is a caring
community of successful learners which can be seen through collaborative studen
t project based learning in and out of the classroom. Kindergarteners in my cla
ss love to work with hands-on materials and have many different opportunities t
o practice a skill before it is mastered. Having the social skills to work coop
eratively with friends is a crucial aspect of the kindergarten curriculum.Monta
na is the perfect place to learn about agriculture and nutrition. My students l
ove to role play in our pretend kitchen in the early childhood classroom. I hav
e had several kids ask me, \"Can we try cooking with REAL food?\" I will take t
heir idea and create \"Common Core Cooking Lessons\" where we learn important m
ath and writing concepts while cooking delicious healthy food for snack time. M
y students will have a grounded appreciation for the work that went into making
the food and knowledge of where the ingredients came from as well as how it is
healthy for their bodies. This project would expand our learning of nutrition a
nd agricultural cooking recipes by having us peel our own apples to make homema
de applesauce, make our own bread, and mix up healthy plants from our classroom
garden in the spring. We will also create our own cookbooks to be printed and s
hared with families. \r\nStudents will gain math and literature skills as well
as a life long enjoyment for healthy cooking.nannan
==================================================
```

In [15]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-bre
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
 A person is a person, no matter how small.  (Dr.Seuss) I teach the smallest st
udents with the biggest enthusiasm for learning. My students learn in many diff
erent ways using all of our senses and multiple intelligences. I use a wide ran
ge of techniques to help all my students succeed.   Students in my class come f
rom a variety of different backgrounds which makes for wonderful sharing of exp
eriences and cultures, including Native Americans.  Our school is a caring comm
unity of successful learners which can be seen through collaborative student pr
oject based learning in and out of the classroom. Kindergarteners in my class l
ove to work with hands-on materials and have many different opportunities to pr
actice a skill before it is mastered. Having the social skills to work cooperat
ively with friends is a crucial aspect of the kindergarten curriculum.Montana i
s the perfect place to learn about agriculture and nutrition. My students love
to role play in our pretend kitchen in the early childhood classroom. I have ha
d several kids ask me,  Can we try cooking with REAL food?  I will take their i
dea and create  Common Core Cooking Lessons  where we learn important math and
writing concepts while cooking delicious healthy food for snack time. My studen
ts will have a grounded appreciation for the work that went into making the foo
d and knowledge of where the ingredients came from as well as how it is healthy
for their bodies. This project would expand our learning of nutrition and agric
ultural cooking recipes by having us peel our own apples to make homemade apple
sauce, make our own bread, and mix up healthy plants from our classroom garden
in the spring. We will also create our own cookbooks to be printed and shared w
ith families.   Students will gain math and literature skills as well as a life
long enjoyment for healthy cooking.nannan
```

In [16]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
 A person is a person no matter how small Dr Seuss I teach the smallest student
s with the biggest enthusiasm for learning My students learn in many different
ways using all of our senses and multiple intelligences I use a wide range of t
echniques to help all my students succeed Students in my class come from a vari
ety of different backgrounds which makes for wonderful sharing of experiences a
nd cultures including Native Americans Our school is a caring community of succ
essful learners which can be seen through collaborative student project based l
earning in and out of the classroom Kindergarteners in my class love to work wi
th hands on materials and have many different opportunities to practice a skill
before it is mastered Having the social skills to work cooperatively with frien
ds is a crucial aspect of the kindergarten curriculum Montana is the perfect pl
ace to learn about agriculture and nutrition My students love to role play in o
ur pretend kitchen in the early childhood classroom I have had several kids ask
me Can we try cooking with REAL food I will take their idea and create Common C
ore Cooking Lessons where we learn important math and writing concepts while co
oking delicious healthy food for snack time My students will have a grounded ap
preciation for the work that went into making the food and knowledge of where t
he ingredients came from as well as how it is healthy for their bodies This pro
ject would expand our learning of nutrition and agricultural cooking recipes by
having us peel our own apples to make homemade applesauce make our own bread an
d mix up healthy plants from our classroom garden in the spring We will also cr
eate our own cookbooks to be printed and shared with families Students will gai
n math and literature skills as well as a life long enjoyment for healthy cooki
ng nannan
```

In [17]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= {'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', '
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itsel
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because'
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'th
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all'
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "di
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn'
            '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())
```

```
100%|██████████████████████████████████████████████████████████████████████████████|
109248/109248 [00:18<00:00, 6042.36it/s]
```

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

Out[19]:
```
'person person no matter small dr seuss teach smallest students biggest enthusi
asm learning students learn many different ways using senses multiple intellige
nces use wide range techniques help students succeed students class come variet
y different backgrounds makes wonderful sharing experiences cultures including
native americans school caring community successful learners seen collaborative
student project based learning classroom kindergarteners class love work hands
materials many different opportunities practice skill mastered social skills wo
rk cooperatively friends crucial aspect kindergarten curriculum montana perfect
place learn agriculture nutrition students love role play pretend kitchen early
childhood classroom several kids ask try cooking real food take idea create com
mon core cooking lessons learn important math writing concepts cooking deliciou
s healthy food snack time students grounded appreciation work went making food
knowledge ingredients came well healthy bodies project would expand learning nu
trition agricultural cooking recipes us peel apples make homemade applesauce ma
ke bread mix healthy plants classroom garden spring also create cookbooks print
ed shared families students gain math literature skills well life long enjoymen
t healthy cooking nannan'
```

1.4 Preprocessing of `project_title`

In [20]:
```
# similarly you can preprocess the titles also
preprocessed_titles = []
for sentance in tqdm(project_data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████| 1
09248/109248 [00:01<00:00, 79646.24it/s]
```

In [21]:
```
# after preprocesing
preprocessed_titles[1000]
```

Out[21]:
```
'empowering students through art learning about then now'
```

1.5 Preprocessing of `project_grade_category`

In [22]:
```
#uunique values:
#array(['Grades PreK-2', 'Grades 9-12', 'Grades 6-8', 'Grades 3-5'],
#      dtype=object)

#preprocess project_grade_category for CountVectorizer
project_data['project_grade_category'] = project_data['project_grade_category'].s
project_data['project_grade_category'] = project_data['project_grade_category'].s
```

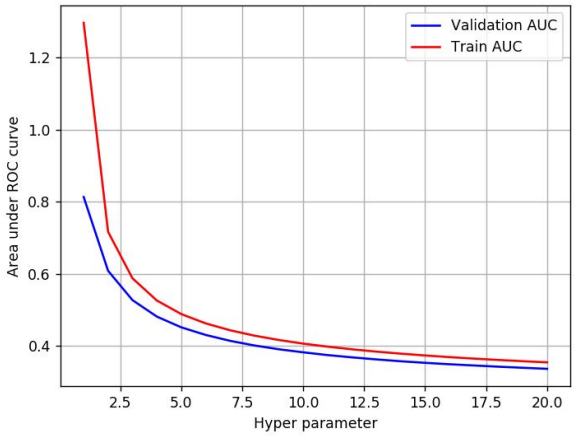
Assignment 3: Apply KNN

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)

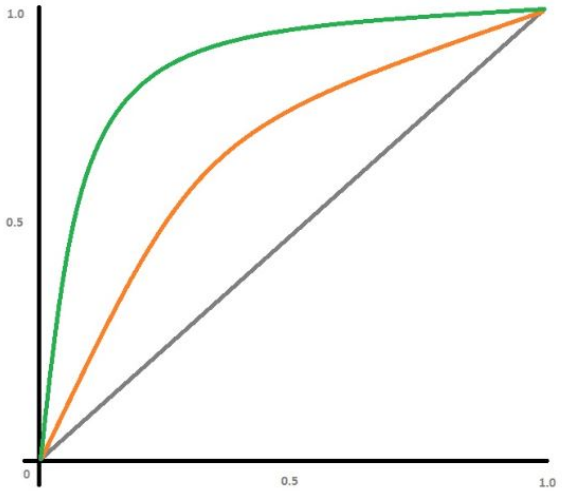
2. **Hyper paramter tuning to find best K**
   - Find the best hyper parameter which results in the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - Find the best hyper paramter using k-fold cross 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

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

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 SelectKBes
t, 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/)

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

Preparing data for models

In [23]:
```
project_data.columns
```

Out[23]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_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
```

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. K Nearest Neighbor

https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/2954/code-samplecross-validation/3/module-3-foundations-of-natural-language-processing-and-machine-learning (https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/2954/code-samplecross-validation/3/module-3-foundations-of-natural-language-processing-and-machine-learning)

Here's the procedure
1) Import the libraries and read the data into a dataframe.
2) Perform Exploratory Data Analysis and summarize the characteristics of the dataset.
3) Perform Text preprocessing and remove the punctuations, stopwords andconvert all the words to lowercase.
4)
a) If you want to go for Simple Cross Validation, Split the dataset into 3 parts. D_train,D_cv,D_test and convert the text into features on D_train and using the same feature, convert the text into D_cv and D_test into columns. Perform hyper-parameter tuning through Simple Cross validation and find out the optimal value of 'K' and build an optimal model now with the optimal value of 'K'. Make predictions on the test data and then compute the performance metrics.
b) If you want to go for K'-fold Cross Validation, Split the dataset into 3 parts. D_train and D_test and convert the text into features on D_train and using the same feature, convert the text into

D_test into columns. Perform hyper-parameter tuning through K'-fold Cross validation and find out the optimal value of 'K' and build an optimal model now with the optimal value of 'K'. Make predictions on the test data and then compute the performance metrics.

After you split the data into train and test datasets, you need to use any of the vectorizers like BOW/TFIDF and vectorize the data according to the training data and then transform both train and test data using the vectorizer. This way you'll get same features in both train and test datasets and then you can perform feature scaling and start performing cross validation.

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

In [24]:
```
# please write all the code with proper documentation, and proper titles for each
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debuggin
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label



from sklearn.model_selection import train_test_split

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratif
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33,
```

2.2 Make Data Model Ready: encoding numerical, categorical features

2.2.1 Vectorizing Numerical features

In [25]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).r
X_train = pd.merge(X_train, price_data, on='id', how='left')
X_cv = pd.merge(X_cv, price_data, on='id', how='left')
X_test = pd.merge(X_test, price_data, on='id', how='left')
```

In [26]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(-1, 1))

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

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

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

In [27]:
```
#X_train_price_standardized
#X_test_price_standardized
```

In [28]:
```
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.res

X_train_previously_posted_projects_norm = normalizer.transform(X_train['teacher_n
X_cv_previously_posted_projects_norm = normalizer.transform(X_cv['teacher_number_
X_test_previously_posted_projects_norm = normalizer.transform(X_test['teacher_num

print("After vectorizations")
print(X_train_previously_posted_projects_norm.shape, y_train.shape)
print(X_cv_previously_posted_projects_norm.shape, y_cv.shape)
print(X_test_previously_posted_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================
====================
```

In [29]:
```
#X_train_previously_posted_projects_standardized
#X_test_previously_posted_projects_standardized
```

2.2.2 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 [30]:
```
from collections import Counter
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on tr

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

print("After vectorizations")
print(X_train_clean_cat_ohe.shape, y_train.shape)
print(X_cv_clean_cat_ohe.shape, y_cv.shape)
print(X_test_clean_cat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 9) (49041,)
(24155, 9) (24155,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy
_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
============================================================================
====================
```

In [31]:
```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_sub_ohe = vectorizer.transform(X_train['clean_subcategories'].value
X_cv_clean_sub_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_clean_sub_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_clean_sub_ohe.shape, y_train.shape)
print(X_cv_clean_sub_ohe.shape, y_cv.shape)
print(X_test_clean_sub_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 30) (49041,)
(24155, 30) (24155,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'env
ironmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlang
uages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geogra
phy', 'literacy', 'literature_writing', 'mathematics', 'music', 'nutritioneduca
tion', 'other', 'parentinvolvement', 'performingarts', 'socialsciences', 'speci
alneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================
====================
```

In [32]:
```
# you can do the similar thing with state, teacher_prefix and project_grade_categ
```

In [33]:
```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train

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

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 51) (49041,)
(24155, 51) (24155,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia',
'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms',
'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa',
'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
====================================================================================
======================
```

In [34]:
```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].fillna(' ').values) # fit has to happen

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

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
#print("="*100)
```

```
After vectorizations
(49041, 5) (49041,)
(24155, 5) (24155,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

In [35]:
```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only

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

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 4) (49041,)
(24155, 4) (24155,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================
====================
```

2.3 Make Data Model Ready: encoding eassay, and project_title (Vectorizing Text data)

2.3.1 Bag of words

In [36]:
```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

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

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

```
After vectorizations
(49041, 44189) (49041,)
(24155, 44189) (24155,)
(36052, 44189) (36052,)
['00', '000', '003', '005nannan', '00am', '00pm', '01', '010', '01g', '01ip',
'02', '021', '03', '030', '0315', '034', '04', '041', '04112016', '047', '0
5', '050', '059', '05a', '06', '060', '07', '072', '076', '08', '084', '09',
'0the', '0ver', '10', '100', '1000', '1000blackgirlbooks', '100m', '100s', '1
00th', '101', '102', '1020', '103', '104', '1043', '105', '1057', '106', '10
7', '1070', '1077', '108', '109', '1099', '10_things_children_learn_block_pla
y', '10k', '10pk', '10pm', '10s', '10th', '10u', '10x', '10x10', '11', '110',
'1100', '1104', '110mph', '111', '112', '1120', '11242', '112th', '113', '11
4', '115', '116', '117', '1170l', '11am', '11pm', '11th', '11x14', '11x17',
'11x25', '12', '120', '1200', '1202', '120th', '121', '122', '122514', '123',
'1238', '123d', '123s', '124', '1248', '125', '1250', '125th', '126', '127',
'128', '128oz', '129', '12pm', '12th', '12u', '12x12', '13', '130', '1300',
'1307', '130ish', '131', '131210', '132', '133', '134', '135', '1350', '135
4', '1358', '136', '137', '138', '139', '13th', '14', '140', '1400', '1402',
'1404', '141', '142', '143', '144', '1440', '145', '1450', '145boys', '145m',
```

In [37]:
```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on train

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['project_title'].values)
X_cv_title_bow = vectorizer.transform(X_cv['project_title'].values)
X_test_title_bow = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 11943) (49041,)
(24155, 11943) (24155,)
(36052, 11943) (36052,)
['000', '03', '04', '05', '06', '09', '0n', '0s', '10', '100', '1000', '1000b
lackgirlbooks', '100th', '101', '103', '104', '105', '106', '107', '109', '10
th', '11', '112', '118', '119', '11th', '12', '123', '123s', '124', '12th',
'13', '14', '1402', '15', '153rd', '16', '17', '175', '18', '180', '185', '1
9', '1920s', '1984', '19th', '1a', '1b', '1e', '1s', '1st', '20', '200', '20
1', '2016', '2017', '2018', '2028', '203', '2030', '2032', '204', '205', '20
6', '207', '209', '20th', '21', '212', '213', '21st', '21th', '22', '220', '2
23', '24', '25', '250', '250million', '26', '27', '270lbs', '273', '28', '280
lbs', '288b', '29', '2b', '2d', '2e', '2nd', '30', '3000', '301', '302', '30
3', '304', '306', '310', '311', '312', '32', '321', '34', '35', '36', '360',
'365', '37', '39', '3c', '3d', '3do', '3doodle', '3doodler', '3doodlers', '3d
oodling', '3dprinter', '3f', '3p', '3r', '3rd', '3rdgradescholars', '3rs', '3
s', '40', '404', '409stingstore', '42', '469', '4creating', '4cs', '4k', '4
p', '4th', '4thgraders', '50', '501', '51', '55', '58', '5k', '5th', '60', '6
1', '616', '63', '6504', '6th', '702', '721q', '7th', '80', '800', '833', '83
```

2.3.2 TFIDF vectorizer

In [38]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_Tfidf = vectorizer.transform(X_train['essay'].values)
X_cv_essay_Tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_Tfidf = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_Tfidf.shape, y_train.shape)
print(X_cv_essay_Tfidf.shape, y_cv.shape)
print(X_test_essay_Tfidf.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 44189) (49041,)
(24155, 44189) (24155,)
(36052, 44189) (36052,)
['00', '000', '003', '005nannan', '00am', '00pm', '01', '010', '01g', '01ip',
'02', '021', '03', '030', '0315', '034', '04', '041', '04112016', '047', '0
5', '050', '059', '05a', '06', '060', '07', '072', '076', '08', '084', '09',
'0the', '0ver', '10', '100', '1000', '1000blackgirlbooks', '100m', '100s', '1
00th', '101', '102', '1020', '103', '104', '1043', '105', '1057', '106', '10
7', '1070', '1077', '108', '109', '1099', '10_things_children_learn_block_pla
y', '10k', '10pk', '10pm', '10s', '10th', '10u', '10x', '10x10', '11', '110',
'1100', '1104', '110mph', '111', '112', '1120', '11242', '112th', '113', '11
4', '115', '116', '117', '1170l', '11am', '11pm', '11th', '11x14', '11x17',
'11x25', '12', '120', '1200', '1202', '120th', '121', '122', '122514', '123',
'1238', '123d', '123s', '124', '1248', '125', '1250', '125th', '126', '127',
'128', '128oz', '129', '12pm', '12th', '12u', '12x12', '13', '130', '1300',
'1307', '130ish', '131', '131210', '132', '133', '134', '135', '1350', '135
4', '1358', '136', '137', '138', '139', '13th', '14', '140', '1400', '1402',
'1404', '141', '142', '143', '144', '1440', '145', '1450', '145boys', '145m',
```

In [39]:
```
# Similarly you can vectorize for title also
vectorizer = TfidfVectorizer()
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on train

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_Tfidf = vectorizer.transform(X_train['project_title'].values)
X_cv_title_Tfidf = vectorizer.transform(X_cv['project_title'].values)
X_test_title_Tfidf = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_title_Tfidf.shape, y_train.shape)
print(X_cv_title_Tfidf.shape, y_cv.shape)
print(X_test_title_Tfidf.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 11943) (49041,)
(24155, 11943) (24155,)
(36052, 11943) (36052,)
['000', '03', '04', '05', '06', '09', '0n', '0s', '10', '100', '1000', '1000b
lackgirlbooks', '100th', '101', '103', '104', '105', '106', '107', '109', '10
th', '11', '112', '118', '119', '11th', '12', '123', '123s', '124', '12th',
'13', '14', '1402', '15', '153rd', '16', '17', '175', '18', '180', '185', '1
9', '1920s', '1984', '19th', '1a', '1b', '1e', '1s', '1st', '20', '200', '20
1', '2016', '2017', '2018', '2028', '203', '2030', '2032', '204', '205', '20
6', '207', '209', '20th', '21', '212', '213', '21st', '21th', '22', '220', '2
23', '24', '25', '250', '250million', '26', '27', '270lbs', '273', '28', '280
lbs', '288b', '29', '2b', '2d', '2e', '2nd', '30', '3000', '301', '302', '30
3', '304', '306', '310', '311', '312', '32', '321', '34', '35', '36', '360',
'365', '37', '39', '3c', '3d', '3do', '3doodle', '3doodler', '3doodlers', '3d
oodling', '3dprinter', '3f', '3p', '3r', '3rd', '3rdgradescholars', '3rs', '3
s', '40', '404', '409stingstore', '42', '469', '4creating', '4cs', '4k', '4
p', '4th', '4thgraders', '50', '501', '51', '55', '58', '5k', '5th', '60', '6
1', '616', '63', '6504', '6th', '702', '721q', '7th', '80', '800', '833', '83
```

2.3.3 Using Pretrained Models: Avg W2V

In [40]:
```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

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

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

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

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

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

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

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


# stronging variables into pickle files python: http://www.jessicayung.com/how-to

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


'''
```

Out[40]:
```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef (https://stackoverflow.com/a/38230349/4084039\ndef) loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n        word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        model[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.txt\')\n\n# ==========================\nOutput:\n\nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n# ==========================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus", len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words that are present in both glove vectors and our coupus",       len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove = set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\nprint("word 2 vec length", len(words_courpus))\n\n\n\n# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport (http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport) pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n\n'
```

In [41]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to
# 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 [42]:
```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in t
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

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

```
100%|████████████████████████████████████████████████████████████████
██| 49041/49041 [00:17<00:00, 2793.95it/s]
```

In [43]:
```
avg_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_cv.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████
| 24155/24155 [00:08<00:00, 2876.89it/s]
```

In [44]:
```
avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in th
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████
| 36052/36052 [00:11<00:00, 3017.73it/s]
```

In [45]:
```
# Similarly you can vectorize for title also
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_titles_train = []; # the avg-w2v for each sentence/review is stor
# for each review/sentence
for sentence in tqdm(X_train['project_title'].values):
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_train.append(vector)

print(len(avg_w2v_vectors_titles_train))
print(len(avg_w2v_vectors_titles_train[0]))
print(avg_w2v_vectors_titles_train[0])
```

```
100%|██████████████████████████████████████████████████████████████████████████|
49041/49041 [00:00<00:00, 135888.73it/s]

49041
300
[-2.4837e-01 -4.5461e-01  3.9227e-02 -2.8422e-01 -3.1852e-02  2.6355e-01
 -4.6323e+00  1.3890e-02 -5.3928e-01 -8.4454e-02  6.1556e-02 -4.1552e-01
 -1.4599e-01 -5.9321e-01 -2.8738e-02 -3.4991e-02 -2.9698e-01 -7.9850e-02
  2.7312e-01  2.2040e-01 -8.9859e-02  8.8265e-04 -4.1991e-01 -1.2536e-01
 -5.4629e-02  3.0550e-02  1.9340e-01 -6.3945e-02  2.7405e-02  5.1193e-02
 -3.8656e-01 -1.1085e-01  1.7259e-01  2.9804e-01 -3.5183e-01  1.3150e-01
 -5.4006e-01 -7.6677e-01 -5.5168e-04  1.3076e-01  2.5101e-02  6.2106e-01
 -2.4797e-01 -3.9790e-01 -3.6116e-01 -5.1967e-01  3.0138e-02 -5.2436e-02
  6.9281e-02  3.5252e-02 -2.1402e-01  2.4836e-01 -1.5693e-01  1.2829e-01
  3.5425e-01 -1.6080e-01 -5.0720e-03 -3.0656e-01 -2.9514e-01 -1.3554e-01
 -1.4385e-01 -4.0552e-01  5.7233e-01 -2.7670e-01  3.0519e-01  1.5586e-01
  1.6086e-02 -2.2009e-01  4.8589e-01 -4.1384e-01  2.0546e-01  4.0491e-01
  4.1558e-02 -1.3542e-01  2.2544e-01 -2.3629e-01  1.5193e-01 -1.0859e-02
 -8.2662e-02 -5.5484e-01 -6.1584e-02 -1.1112e-01 -1.1982e-01 -3.7064e-01
  1.6501e-01  4.4063e-01 -3.3883e-01 -5.7676e-01  5.0847e-01 -3.5707e-02
 -5.9233e-02  3.0748e-02 -2.7689e-01 -7.0433e-02  2.7786e-02 -5.9336e-01
 -2.8220e+00 -1.0052e-01  6.7168e-01 -1.7046e-01 -2.5902e-01  2.7938e-01
  3.9992e-01  3.7480e-02 -2.6409e-01 -2.6378e-01  2.0645e-01  1.7564e-01
 -8.0807e-02 -3.8376e-01  2.6602e-01  3.6214e-01 -9.5112e-02  3.5199e-01
 -8.6994e-01 -1.5747e-01 -2.2550e-01 -6.4948e-02 -2.4845e-01  1.5038e-01
 -3.2951e-01 -2.2285e-01 -2.5509e-02 -2.9725e-01 -3.7715e-01  8.9296e-02
 -3.4399e-02  3.3640e-01  3.5534e-01  3.8253e-01  1.7646e-01  1.3305e-01
 -3.2743e-01 -4.7115e-01  2.4673e-01 -1.5964e-01  1.8212e-01 -4.1241e-01
  9.8565e-02  3.8118e-01  3.3043e-01  5.1987e-02 -2.1824e-01  2.2214e-01
 -5.9450e-02 -6.3743e-02  4.3723e-01  1.1068e-01  4.7444e-01  5.6891e-01
  3.1123e-01 -2.0272e-01  8.0078e-02 -4.3905e-01 -1.2246e-01 -2.5057e-02
 -5.7162e-02  1.4250e-01  9.4468e-02  1.2991e-01  1.0444e-01 -3.9447e-01
 -2.9337e-01 -2.0466e-01  2.0815e-01 -1.6010e-01 -1.4665e-01  5.4511e-01
  2.9740e-01 -2.2959e-01 -1.7050e-01 -6.2371e-02 -5.0399e-01 -3.8000e-01
 -3.9528e-01  5.7552e-01 -4.6892e-01 -4.3308e-01  1.5018e-01 -4.1179e-02
  6.2157e-01  1.9874e-02 -1.1969e-01 -2.5611e-01  2.6602e-01 -3.7383e-01
  1.2936e-01 -5.0006e-02 -1.1554e-01 -1.7163e-01 -4.2430e-01  1.9844e-01
```

```
 5.0611e-01 -1.1093e-01 -1.3939e-01 -5.9377e-01  6.7338e-01  3.8497e-01
 6.2604e-01 -2.0128e-01  3.0058e-01 -1.3946e-01 -1.6186e-01  1.2168e-01
-1.8410e-02  6.1356e-01 -1.9887e-01  1.9250e-01  8.4372e-03 -5.0757e-01
 3.5858e-01 -4.9729e-01 -4.4725e-01  2.1423e-02 -2.0769e-01  8.3729e-02
 2.2032e-01  1.4404e-01  1.2590e-03 -4.4309e-01 -1.7242e-01 -3.5300e-01
-2.9477e-01  3.2898e-01 -3.1910e+00  3.8910e-01  3.5654e-01  5.2134e-02
 2.0576e-01 -8.8649e-02  1.6398e-01  1.1203e-01  2.8590e-01  2.8940e-01
-4.4349e-01  9.1036e-01 -3.0902e-01 -1.3985e-01 -3.9499e-01 -2.7299e-02
-1.5201e-01  8.4418e-02 -3.7196e-01  4.9827e-02  1.4128e-01 -1.5126e-01
-1.6107e-01  4.0226e-03  1.6799e-01 -2.5429e-01 -1.5074e-01 -5.7409e-01
-1.5611e-01  6.8407e-02  2.4832e-01  1.6828e-01  7.2764e-02 -8.6728e-02
 2.1982e-03  1.3593e-01  7.0224e-01 -4.5976e-01 -2.4506e-01 -3.3874e-01
-1.0952e-01  2.4698e-01 -5.5919e-01 -3.8866e-01 -1.3372e-01  9.1943e-02
-1.0543e-01 -3.1319e-01 -2.9952e-01 -2.0611e-01  1.7976e-01  4.5800e-01
-7.2402e-02  1.6118e-01 -4.1649e-01 -3.0103e-01  2.3234e-01 -5.0139e-02
 1.0026e-01  3.8974e-01 -6.1342e-02  2.6626e-01 -1.5671e-01  7.5136e-02
-4.2926e-01 -1.2025e-01  8.2736e-02 -6.2469e-01  4.4267e-02  6.0673e-01
-1.2458e-01 -1.5443e-01 -1.6339e-01  5.3097e-02  1.5458e-01 -3.8053e-01]
```

In [46]:
```
avg_w2v_vectors_titles_cv = []; # the avg-w2v for each sentence/review is stored
for sentence in tqdm(X_cv['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_cv.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████████|
24155/24155 [00:00<00:00, 154286.89it/s]
```

In [47]:
```
avg_w2v_vectors_titles_test = []; # the avg-w2v for] each sentence/review is stor
for sentence in tqdm(X_test['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_titles_test.append(vector)
```

```
100%|██████████████████████████████████████████████████████████████████████████████|
36052/36052 [00:00<00:00, 148394.74it/s]
```

2.3.4 Using Pretrained Models: TFIDF weighted W2V

In [48]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['essay'].values)
# 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 [49]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this
for sentence in tqdm(X_train['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

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

```
100%|████████████████████████████████████████████████████████████████
| 49041/49041 [02:55<00:00, 279.01it/s]

49041
300
```

In [50]:
```
tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in th
for sentence in tqdm(X_cv['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_cv.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████
| 24155/24155 [01:27<00:00, 277.28it/s]
```

In [51]:
```
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in
for sentence in tqdm(X_test['essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)
```

```
100%|████████████████████████████████████████████████████████████████
| 36052/36052 [02:10<00:00, 276.41it/s]
```

In [52]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model_titles = TfidfVectorizer()
tfidf_model_titles.fit(X_train['project_title'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_titles.get_feature_names(), list(tfidf_model_ti
tfidf_words_titles = set(tfidf_model_titles.get_feature_names())
```

In [53]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_titles = []; # the avg-w2v for each sentence/review is stored i
for sentence in tqdm(X_train['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_titles):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles.append(vector)

print(len(tfidf_w2v_vectors_titles))
print(len(tfidf_w2v_vectors_titles[0]))
```

```
100%|████████████████████████████████████████████████████████████████|
49041/49041 [00:00<00:00, 95078.30it/s]

49041
300
```

In [54]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_titles_cv = []; # the avg-w2v for each sentence/review is store
for sentence in tqdm(X_cv['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_titles):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_cv.append(vector)

print(len(tfidf_w2v_vectors_titles_cv))
print(len(tfidf_w2v_vectors_titles_cv[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████|
24155/24155 [00:00<00:00, 96694.23it/s]

24155
300
```

In [55]:
```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_titles_test = []; # the avg-w2v for each sentence/review is sto
for sentence in tqdm(X_test['project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_titles):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_titles_test.append(vector)

print(len(tfidf_w2v_vectors_titles_test))
print(len(tfidf_w2v_vectors_titles_test[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████|
36052/36052 [00:00<00:00, 100757.38it/s]

36052
300
```

Features to use

numerical

X_train_price_norm
X_cv_price_norm
X_test_price_norm

X_train_previously_posted_projects_norm
X_cv_previously_posted_projects_norm
X_test_previously_posted_projects_norm

categorical

X_train_clean_cat_ohe
X_cv_clean_cat_ohe
X_test_clean_cat_ohe

X_train_clean_sub_ohe
X_cv_clean_sub_ohe
X_test_clean_sub_ohe

X_train_state_ohe
X_cv_state_ohe
X_test_state_ohe

X_train_teacher_ohe
X_cv_teacher_ohe
X_test_teacher_ohe

X_train_grade_ohe
X_cv_grade_ohe
X_test_grade_ohe

Bag of Word

X_train_essay_bow
X_cv_essay_bow
X_test_essay_bow

X_train_title_bow
X_cv_title_bow
X_test_title_bow

TFIDF vectorizer

X_train_essay_Tfidf
X_cv_essay_Tfidf
X_test_essay_Tfidf

X_train_title_Tfidf
X_cv_title_Tfidf
X_test_title_Tfidf

Avg W2V

avg_w2v_vectors_train
avg_w2v_vectors_cv
avg_w2v_vectors_test

avg_w2v_vectors_titles_train
avg_w2v_vectors_titles_cv
avg_w2v_vectors_titles_test

TFIDF weighted W2V

tfidf_w2v_vectors
tfidf_w2v_vectors_cv
tfidf_w2v_vectors_test

tfidf_w2v_vectors_titles
tfidf_w2v_vectors_titles_cv
tfidf_w2v_vectors_titles_test

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

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

In [56]:
```
# please write all the code with proper documentation, and proper titles for each
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debuggin

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

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation

import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score

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

from scipy.sparse import hstack
import time
from sklearn.metrics import confusion_matrix
```

```
C:\Users\francisco.porrata\AppData\Local\Continuum\anaconda3\lib\site-packages
\sklearn\cross_validation.py:41: DeprecationWarning:

This module was deprecated in version 0.18 in favor of the model_selection modu
le into which all the refactored classes and functions are moved. Also note tha
t the interface of the new CV iterators are different from that of this module.
This module will be removed in 0.20.
```

In [57]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1
    # 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
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [58]:
```
def model_performance(X_tr, y_train,X_cr,y_cv):


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

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

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

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

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

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

In [59]:
```
def best_parameter_ROC(X_tr, y_train,  X_te, y_test, best_k):
    neigh = KNeighborsClassifier(n_neighbors=best_k, algorithm = "brute", n_jobs=
    neigh.fit(X_tr, y_train)
    # roc_auc_score(y_true, y_scor, e) the 2nd parameter should be probability es
    # not the predicted outputs

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

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

    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_t
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.xlabel("False Positive Rate (fpr)")
    plt.ylabel("True Positive Rate (tpr)")
    plt.title("ROC")
    plt.grid()
    plt.show()
    return (train_fpr, train_tpr, tr_thresholds, y_train_pred, y_test_pred)
```

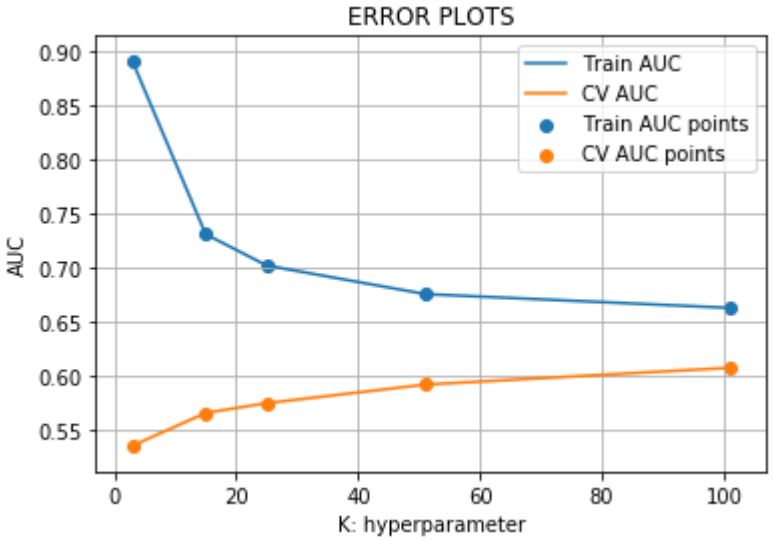
In [60]:
```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold",
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

2.4.1 Applying KNN brute force on BOW, SET 1

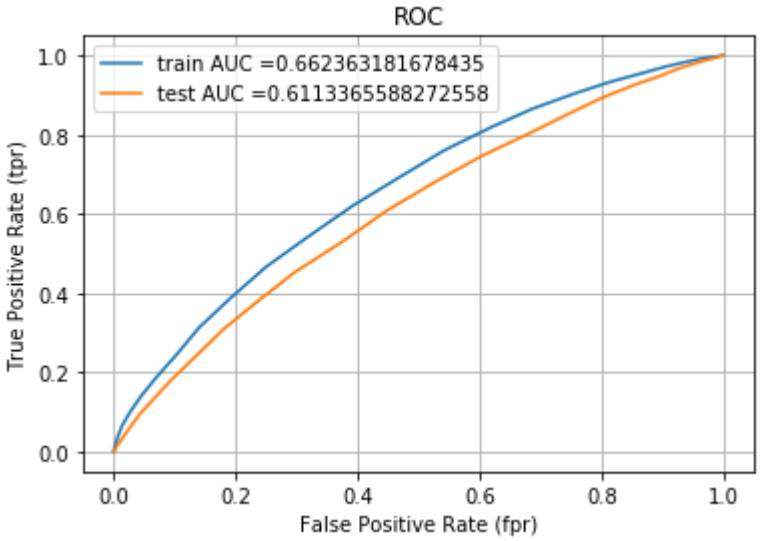
In [61]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

X_tr = hstack((X_train_essay_bow, X_train_title_bow, X_train_state_ohe, X_train_t
               X_train_clean_cat_ohe , X_train_clean_sub_ohe , X_train_price_norm
               X_train_previously_posted_projects_norm )).tocsr()

X_cr = hstack((X_cv_essay_bow, X_cv_title_bow, X_cv_state_ohe, X_cv_teacher_ohe, 
               X_cv_clean_sub_ohe , X_cv_price_norm,X_cv_previously_posted_projec

X_te = hstack((X_test_essay_bow, X_test_title_bow, X_test_state_ohe, X_test_teach
               X_test_clean_cat_ohe, X_test_clean_sub_ohe , X_test_price_norm,X_t


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

model_performance(X_tr, y_train,X_cr,y_cv)
```

```
Final Data matrix
(49041, 56233) (49041,)
(24155, 56233) (24155,)
(36052, 56233) (36052,)
====================================================================================
======================
```

100%|██████████| 5/5 [36:37<00:00, 438.75s/it]



In [62]:
```
#best k using loop
best_k_bow_loop = 101
train_fpr, train_tpr, tr_thresholds, y_train_pred, y_test_pred = best_parameter_R
```



In [63]:
```
print("="*100)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

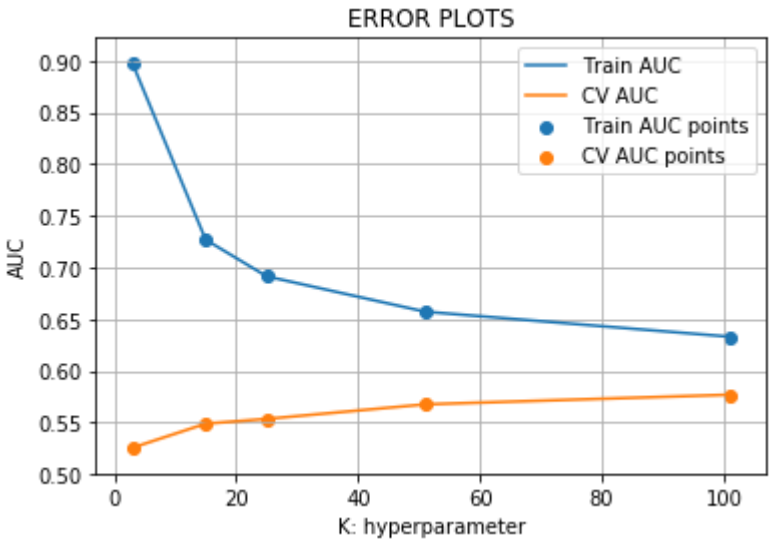
```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.37747830094126844 for threshold 0.822
Train confusion matrix
[[ 4512  2914]
 [15761 25854]]
Test confusion matrix
[[ 3015  2444]
 [11981 18612]]
```

2.4.2 Applying KNN brute force on TFIDF, SET 2

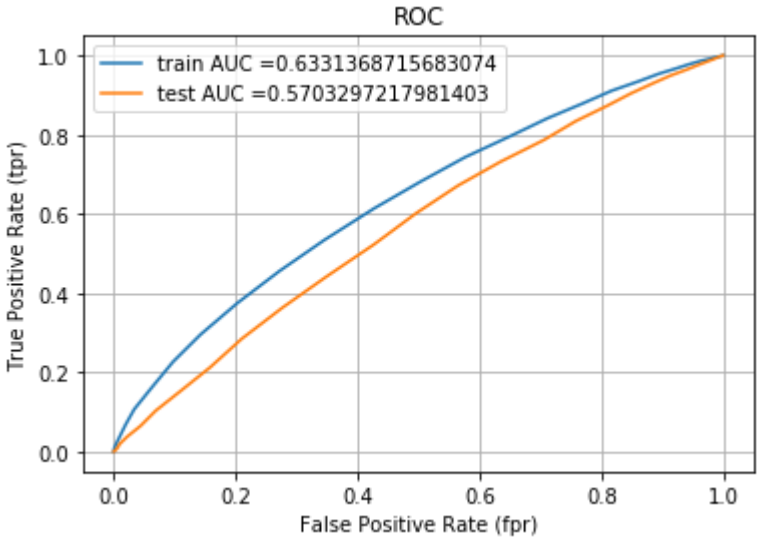
In [64]:
```
X_tr = hstack((X_train_essay_Tfidf, X_train_title_Tfidf, X_train_state_ohe, X_tra
X_cr = hstack((X_cv_essay_Tfidf, X_cv_title_Tfidf, X_cv_state_ohe, X_cv_teacher_o
X_te = hstack((X_test_essay_Tfidf, X_test_title_Tfidf, X_test_state_ohe, X_test_t

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

model_performance(X_tr, y_train,X_cr,y_cv)
```

```
Final Data matrix
(49041, 56233) (49041,)
(24155, 56233) (24155,)
(36052, 56233) (36052,)
====================================================================================
====================
```

```
100%|██████████| 5/5 [36:46<00:00, 443.46s/it]
```



In [65]:
```
best_k_tfidf_loop = 101
train_fpr, train_tpr, tr_thresholds, y_train_pred, y_test_pred = best_parameter_R
```



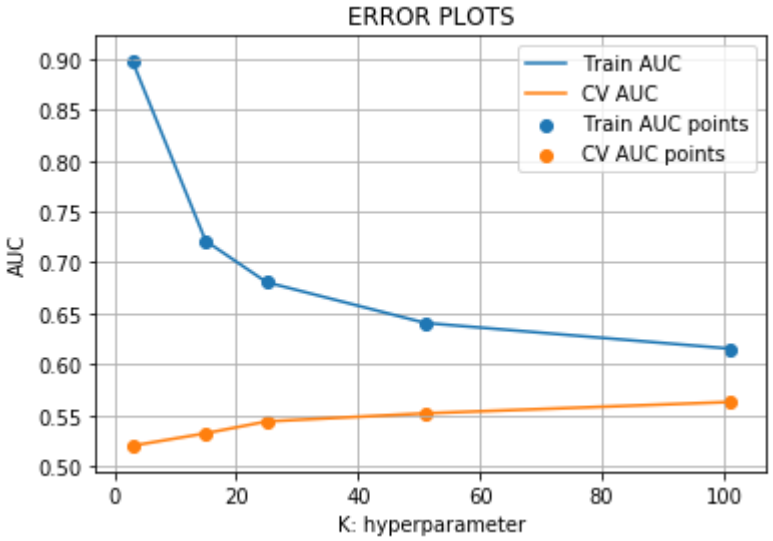
In [66]:
```
print("="*100)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.3518190468920486 for threshold 0.851
Train confusion matrix
[[ 4253  3173]
 [16051 25564]]
Test confusion matrix
[[ 2752  2707]
 [12183 18410]]
```

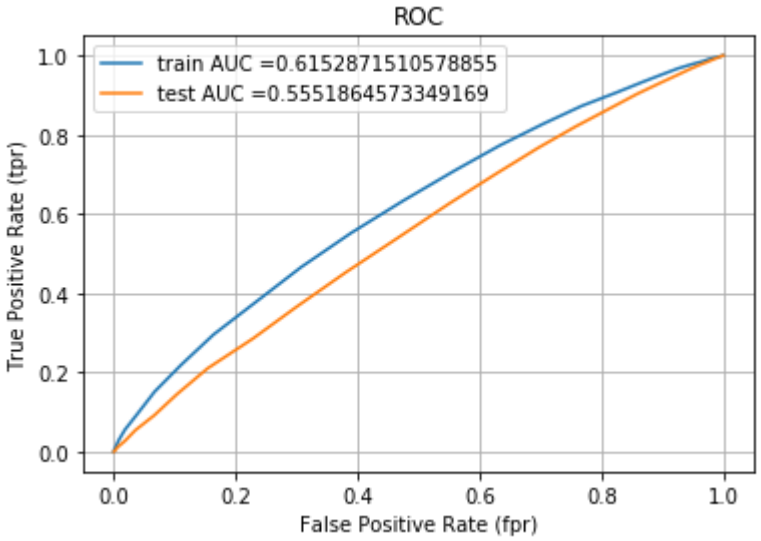
2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [67]:
```
X_tr = hstack((avg_w2v_vectors_train, avg_w2v_vectors_titles_train, X_train_state
X_cr = hstack((avg_w2v_vectors_cv, avg_w2v_vectors_titles_cv, X_cv_state_ohe, X_c
X_te = hstack((avg_w2v_vectors_test, avg_w2v_vectors_titles_test, X_test_state_oh

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

model_performance(X_tr, y_train,X_cr,y_cv)
```

```
Final Data matrix
(49041, 701) (49041,)
(24155, 701) (24155,)
(36052, 701) (36052,)
====================================================================================
======================

100%|████████████████████████████████████████████████████████████████████████████
██████| 5/5 [2:22:43<00:00, 1711.80s/it]
```



In [68]:
```
best_k_w2v_loop = 101
train_fpr, train_tpr, tr_thresholds, y_train_pred, y_test_pred = best_parameter_R
```



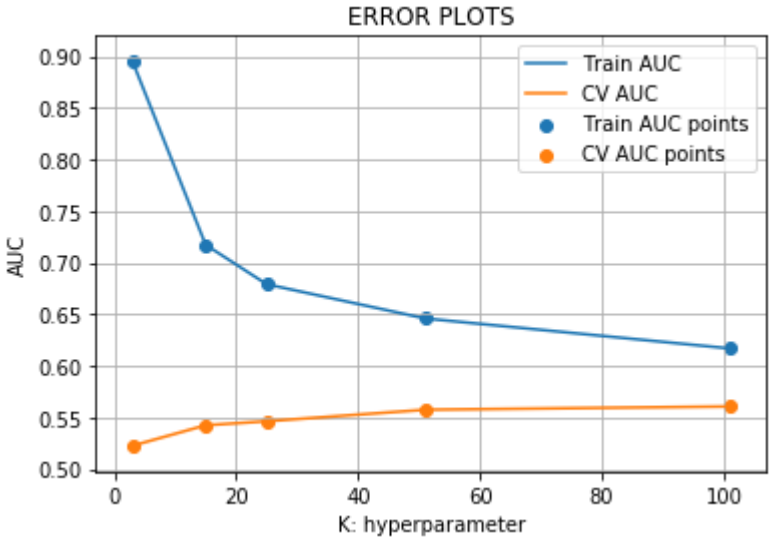
In [69]:
```
print("="*100)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.3371537291212825 for threshold 0.861
Train confusion matrix
[[ 4535  2891]
 [18640 22975]]
Test confusion matrix
[[ 2915  2544]
 [14079 16514]]
```

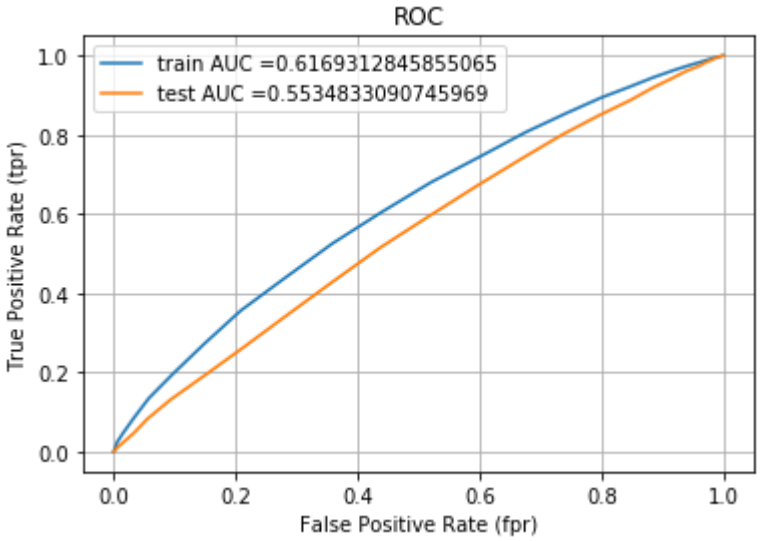
2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [70]:
```
X_tr = hstack((tfidf_w2v_vectors, tfidf_w2v_vectors_titles, X_train_state_ohe, X_
X_cr = hstack((tfidf_w2v_vectors_cv, tfidf_w2v_vectors_titles_cv, X_cv_state_ohe,
X_te = hstack((tfidf_w2v_vectors_test, tfidf_w2v_vectors_titles_test, X_test_stat

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

model_performance(X_tr, y_train,X_cr,y_cv)
```

```
Final Data matrix
(49041, 701) (49041,)
(24155, 701) (24155,)
(36052, 701) (36052,)
====================================================================================
====================

100%|████████████████████████████████████████████████████████████████████████████
██████| 5/5 [2:20:07<00:00, 1682.64s/it]
```



In [71]:
```
best_k_tfidfw2v_loop = 101
train_fpr, train_tpr, tr_thresholds, y_train_pred, y_test_pred = best_parameter_R
```



In [72]:
```
print("="*100)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

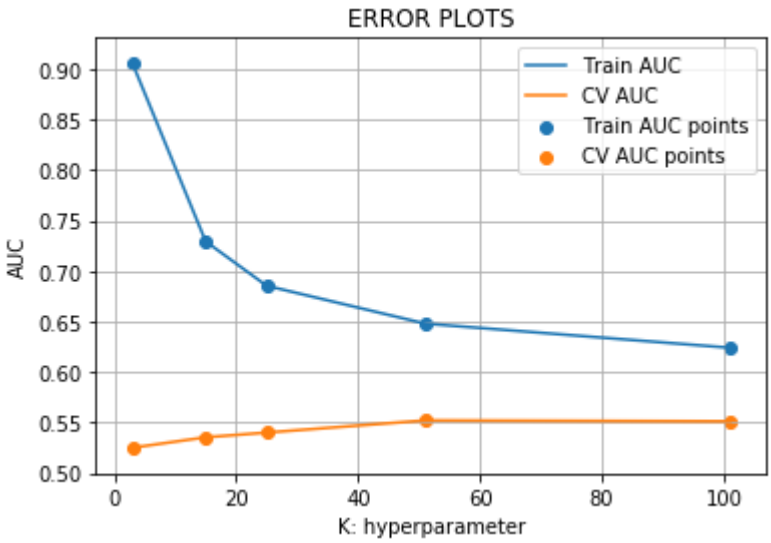
```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.3390388450113368 for threshold 0.851
Train confusion matrix
[[ 4162  3264]
 [16441 25174]]
Test confusion matrix
[[ 2631  2828]
 [12377 18216]]
```

2.5 Feature selection with `SelectKBest`

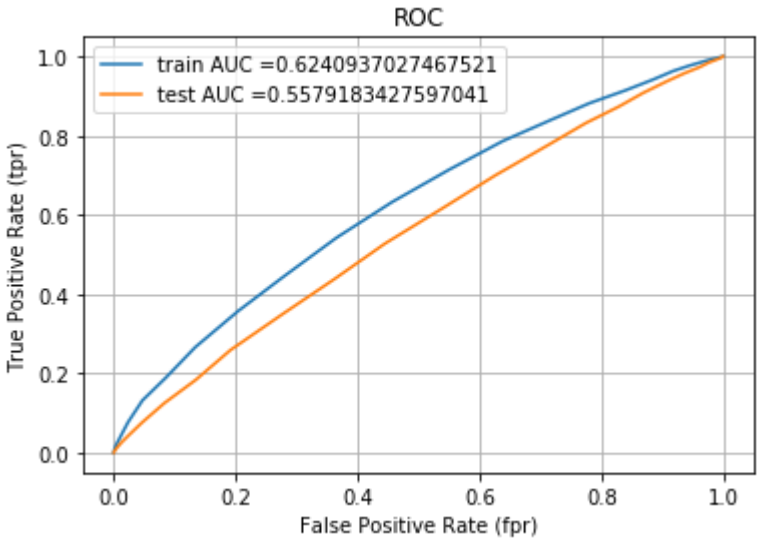
2.5.1 Applying KNN brute force on TFIDF, SET 2

In [73]:
```
#https://stackoverflow.com/questions/43479399/implementing-feature-selection


from sklearn.feature_selection import SelectKBest, chi2

#select best 2000 features
selector = SelectKBest(chi2, k=2000)


#fit_transform train
X_tr = hstack((X_train_essay_Tfidf, X_train_title_Tfidf, X_train_state_ohe, X_tra
X_new_tr = selector.fit_transform(X_tr, y_train)
#y_new_train = selector.transform(y_train.reshape(1,-1))

#transform cv
X_cr = hstack((X_cv_essay_Tfidf, X_cv_title_Tfidf, X_cv_state_ohe, X_cv_teacher_o
X_new_cr = selector.transform(X_cr)
#y_new_cv = selector.transform(y_cv).reshape(1,-1)

#transform test
X_te = hstack((X_test_essay_Tfidf, X_test_title_Tfidf, X_test_state_ohe, X_test_t
X_new_te = selector.transform(X_te)
#y_new_test = selector.transform(y_test).reshape(1,-1)

print("Final Data matrix")
print(X_new_tr.shape, y_train.shape)
print(X_new_cr.shape, y_cv.shape)
print(X_new_te.shape, y_test.shape)
print("="*100)


model_performance(X_new_tr, y_train,X_new_cr,y_cv)
```

```
Final Data matrix
(49041, 2000) (49041,)
(24155, 2000) (24155,)
(36052, 2000) (36052,)
====================================================================================
====================

100%|██████████| 5/5 [25:49<00:00, 309.56s/it]
```



In [74]:
```
best_k_bow_K_loop = 101
train_fpr, train_tpr, tr_thresholds, y_train_pred, y_test_pred = best_parameter_R
```



In [75]:
```
print("="*100)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.344350963953719 for threshold 0.871
Train confusion matrix
[[ 4712  2714]
 [19031 22584]]
Test confusion matrix
[[ 3030  2429]
 [14443 16150]]
```

3. Conclusions

In [2]:
```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

# Using Loop to determine best Hyperparameters

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameter", "AUC"]
x.add_row(["BOW", "Brute",best_k_bow_loop,0.6113])
x.add_row(["TFIDF", "Brute",best_k_tfidf_loop,0.5703])
x.add_row(["W2V", "Brute",best_k_w2v_loop,0.5551])
x.add_row(["TFIDFW2V", "Brute",best_k_tfidfw2v_loop,0.5534])

print(x)
```

```
+------------+-------+----------------+--------+
| Vectorizer | Model | Hyperparameter |  AUC   |
+------------+-------+----------------+--------+
|    BOW     | Brute |      101       | 0.6113 |
|   TFIDF    | Brute |      101       | 0.5703 |
|    W2V     | Brute |      101       | 0.5551 |
|  TFIDFW2V  | Brute |      101       | 0.5534 |
+------------+-------+----------------+--------+
```

In [3]:
```
# Using SelectKBest with TFIDF
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameter", "AUC"]
x.add_row(["TFIDF (Loop)", "Brute",best_k_bow_K_loop, 0.5579])

print(x)
```

```
+--------------+-------+----------------+--------+
|  Vectorizer  | Model | Hyperparameter |  AUC   |
+--------------+-------+----------------+--------+
| TFIDF (Loop) | Brute |      101       | 0.5579 |
+--------------+-------+----------------+--------+
```

Observations

1) BOW gave the best Test AUC

2) The performance of SelectKBest = 2000 was worse than using all the features for SET 2 -> TFIDF