DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

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

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

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

```
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [2]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
test_data = pd.read_csv('test_data.csv')
```

In [3]:

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

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

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
**************************************************
Number of data points in test data (72832, 15)
--------------------------------------------------
The attributes of data : ['id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects']
```

In [4]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
```

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


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


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


project_data.head(2)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	

In [5]:

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

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

Out[5]:

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

1.2 preprocessing of `project_subject_categories`

In [6]:

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

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

```
        temp = temp.replace('&', '_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [7]:

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

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

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

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

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

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

1.3 Text preprocessing

In [8]:

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

In [9]:

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

Out[9]:

Unnamed:	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_t

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Enginee STEAM the Prim Classr
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Sens Tools Fo

In [10]:

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

In [11]:

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

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classroom as well as the STEM j
ournals, which my students really enjoyed.  I would love to implement more of the Lakeshore STEM k
its in my classroom for the next school year as they provide excellent and engaging STEM
lessons.My students come from a variety of backgrounds, including language and socioeconomic statu
s.  Many of them don't have a lot of experience in science and engineering and these 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 would use the kits and robot to help guide my science i
nstruction in engaging and meaningful ways.  I can adapt the kits to my current language arts paci
ng 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 the 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 challenging to d
evelop high quality science activities.  These kits give me the materials I need to provide my
students with science activities that will go along with the curriculum in my classroom.  Although
I have some things (like magnets) in my classroom, I don't know how to use them effectively.  The
kits will 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 disabilities. My students all
vary in their ability level. However, the ultimate goal is to increase all students literacy level
s. This includes their reading, writing, and communication levels.I teach a really dynamic group o
f students. However, my students face a lot of challenges. My students all live in poverty and in
a dangerous neighborhood. Despite these challenges, I have students who have the the desire to def
eat these challenges. My students all have learning disabilities and currently all are performing
below grade level. My students 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 t
o school unprepared for class due to economic challenges.  I want my students to be able to focus
on learning and not how they will be able to get school supplies.  The supplies will last all year
.  Students will be able to complete written assignments and maintain a classroom journal.  The ch
art 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 pr
int student work that is completed on the classroom Chromebooks.I want to try and remove all barri
ers for the students learning and create opportunities for learning. One of the biggest barriers i
s the students not having the resources to get pens, 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 will gain math and literature skills as well as a life long enjoyment for healthy cooking.nannan
==
My classroom consists of twenty-two amazing sixth graders from different cultures and backgrounds. They are a social bunch who enjoy working in partners and working with groups. They are hard-working and eager to head to middle school next year. My job is to get them ready to make this transition and make it as smooth as possible. In order to do this, my students need to come to school every day and feel safe and ready to learn. Because they are getting ready to head to middle school, I give them lots of choice- choice on where to sit 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 welcoming, 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 together. 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 twenty-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 for seating. I allow my students to choose their own spots while they are working 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 more 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 students look forward to their work time so they can move around the room. I would like to get rid of the constricting desks and move toward more “fun” seating options. I am requesting various seating so my students have more options 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 would 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 together, 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 seating options, they will be that much more excited about coming to school! Thank you for your support in making my classroom one students will remember forever!nannan
==

In [12]:

```
# https://stackoverflow.com/a/47091490/4084039
import re
```

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

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

In [13]:

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

```
\"A person is a person, no matter how small.\" (Dr.Seuss) I teach the smallest students with the b
iggest enthusiasm for learning. My students learn in many different ways using all of our senses a
nd multiple intelligences. I use a wide range of techniques to help all my students succeed. \r\nS
tudents 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 su
ccessful learners which can be seen through collaborative student project based learning in and ou
t 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 wor
k 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 coo
king 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 knowled
ge 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 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 classro
om garden in the spring. We will also create our own cookbooks to be printed and shared with famil
ies. \r\nStudents will gain math and literature skills as well as a life long enjoyment for health
y cooking.nannan
==================================================
```

In [14]:

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

```
 A person is a person, no matter how small.  (Dr.Seuss) I teach the smallest students with the big
gest 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.
Students in my class come from a variety of different backgrounds which makes for wonderful
sharing of experiences and cultures, including Native Americans.  Our school is a caring community
of successful learners which can be seen through collaborative student project based learning in a
nd 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 t
o 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 p
retend kitchen in the early childhood classroom. I have had several kids ask me,  Can we try cooki
ng 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 knowled
ge 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 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 classro
om garden in the spring. We will also create our own cookbooks to be printed and shared with famil
ies.   Students will gain math and literature skills as well as a life long enjoyment for healthy
cooking.nannan
```

In [15]:

```
#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 students with the biggest
enthusiasm for learning My students learn in many different ways using all of our senses and multi
ple intelligences I use a wide range of techniques to help all my students succeed Students in my
class come from a variety of different backgrounds which makes for wonderful sharing of
experiences and cultures including Native Americans Our 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 knowled
ge of where the ingredients came from as well as how it is healthy for their bodies This project w
ould expand our learning of nutrition and agricultural cooking recipes by having us peel our own a
pples 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
Students will gain math and literature skills as well as a life long enjoyment for healthy cooking
nannan
```

In [16]:

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

In [17]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
```

```
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████| 109248/109248
[01:10<00:00, 1541.03it/s]
```

In [18]:

```
# after preprocesing
project_data['processed_essay'] = preprocessed_essays;
project_data.drop(['essay'], axis=1, inplace=True)
preprocessed_essays[20000]
```

Out[18]:

```
'person person no matter small dr seuss teach smallest students biggest enthusiasm learning
students learn many different ways using senses multiple intelligences use wide range techniques h
elp students succeed students class come variety different backgrounds makes wonderful sharing exp
eriences cultures including native americans school caring community successful learners seen coll
aborative student project based learning classroom kindergarteners class love work hands materials
many different opportunities practice skill mastered social skills work cooperatively friends cruc
ial 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 id
ea create common core cooking lessons learn important math writing concepts cooking delicious heal
thy food snack time students grounded appreciation work went making food knowledge ingredients cam
e well healthy bodies project would expand learning nutrition agricultural cooking recipes us peel
apples make homemade applesauce make bread mix healthy plants classroom garden spring also create
cookbooks printed shared families students gain math literature skills well life long enjoyment he
althy cooking nannan'
```

In [19]:

```
project_data.columns
```

Out[19]:

```
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', 'processed_essay'],
      dtype='object')
```

1.4 Preprocessing of `project_title`

In [20]:

```
# similarly you can preprocess the titles also

processed_titles = [];
for title in tqdm(project_data['project_title'].values):
    sent = decontracted(title)
    sent = re.sub('\S*\d\S*', '', sent);
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    processed_titles.append(sent.strip())
```

```
100%|██████████████████████████████████████████████████████████████| 109248/109248
[00:02<00:00, 42372.39it/s]
```

In [21]:

```
project_data.drop(['project_title'], axis=1, inplace=True)
project_data['processed_titles'] = processed_titles

#testing after preprocessing project_title column
print(processed_titles[3])

print(processed_titles[40]);
```

```
print(processed_titles[500]);

print(processed_titles[4000]);

project_data.columns
```

```
Flexible Seating for Flexible Learning
Duct Tape Upcycle
Special Needs Students Need Additional Access to Technology
Calling All Techno Kids
```

Out[21]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', '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', 'processed_essay',
       'processed_titles'],
      dtype='object')
```

1.5 Preparing data for models

In [22]:

```
project_data.columns
```

Out[22]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', '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', 'processed_essay',
       'processed_titles'],
      dtype='object')
```

we are going to consider

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

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

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

Merging Price of each project.

In [23]:

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

Out[23]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', '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', 'processed_essay',
       'processed_titles', 'price', 'quantity'],
      dtype='object')
```

1.5.1 Vectorizing Categorical data

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

In [24]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
categories_one_hot = vectorizer.fit_transform(project_data['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (109248, 9)
```

In [25]:

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (109248, 30)
```

In [26]:

```
# Please do the similar feature encoding with state, teacher_prefix and project_grade_category als
o
#one hot encoding of state
state_dict = dict(project_data['school_state'].value_counts())

vectorizer = CountVectorizer(vocabulary=list(state_dict.keys()), lowercase=False, binary=True);
vectorizer.fit(project_data['school_state'].values);
print(vectorizer.get_feature_names());

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

```
['CA', 'TX', 'NY', 'FL', 'NC', 'IL', 'GA', 'SC', 'MI', 'PA', 'IN', 'MO', 'OH', 'LA', 'MA', 'WA', 'C
K', 'NJ', 'AZ', 'VA', 'WI', 'AL', 'UT', 'TN', 'CT', 'MD', 'NV', 'MS', 'KY', 'OR', 'MN', 'CO', 'AR',
'ID', 'IA', 'KS', 'NM', 'DC', 'HI', 'ME', 'WV', 'NH', 'AK', 'DE', 'NE', 'SD', 'RI', 'MT', 'ND', 'WY
', 'VT']
shape of matrix after one hot encoding of school_state  (109248, 51)
```

In [27]:

```
#Error: np.nan is an invalid document, expected byte or unicode string CountVectorizer,
#solution found at: https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-v
alueerror-np-nan-is-an-invalid-document/39308809#39308809

#one hot encoding of teacher_prefix
teacher_dict = dict(project_data['teacher_prefix'].value_counts());
```

```
vectorizer = CountVectorizer(vocabulary=list(teacher_dict.keys()), lowercase=False, binary=True);

vectorizer.fit(project_data['teacher_prefix'].values.astype('U'));
print(vectorizer.get_feature_names());

teacher_prefix_one_hot = vectorizer.transform(project_data['teacher_prefix'].values.astype('U'));
print('shape of materix after one hot encoding of teacher_prefix ', teacher_prefix_one_hot.shape)
```

```
['Mrs.', 'Ms.', 'Mr.', 'Teacher', 'Dr.']
shape of materix after one hot encoding of teacher_prefix  (109248, 5)
```

In [28]:

```
grades_dict = dict(project_data['project_grade_category'].value_counts());

vectorizer = CountVectorizer(vocabulary=list(grades_dict.keys()), lowercase=False, binary=True);

vectorizer.fit(project_data['project_grade_category'].values);
print(vectorizer.get_feature_names());

grades_one_hot = vectorizer.transform(project_data['project_grade_category'].values);
print('shape of matrix after one hot encoding of project_grade_category', grades_one_hot.shape)
```

```
['Grades PreK-2', 'Grades 3-5', 'Grades 6-8', 'Grades 9-12']
shape of matrix after one hot encoding of project_grade_category (109248, 4)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [29]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_bow.shape)
```

```
Shape of matrix after one hot encodig  (109248, 16512)
```

In [30]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
print(project_data['processed_titles'].values)
vectorizer = CountVectorizer(min_df=10);
title_bow = vectorizer.fit_transform(project_data['processed_titles'].values);
print('Shape of matrix after one hot encoding ', title_bow.shape)
```

```
['Engineering STEAM into the Primary Classroom' 'Sensory Tools for Focus'
 'Mobile Learning with a Mobile Listening Center' ...
 'Bringing Agriculture and Sustainability to the Classroom through Computing'
 'Cricket Cutting Machine Needed' 'News for Kids']
Shape of matrix after one hot encoding  (109248, 3300)
```

1.5.2.2 TFIDF vectorizer

In [31]:

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

```
Shape of matrix after one hot encodig  (109248, 16512)
```

In [32]:

```
# Similarly you can vectorize for title also

vectorizer = TfidfVectorizer(min_df=10);
title_tfidf = vectorizer.fit_transform(project_data['processed_titles'].values);
print('Shape of matrix after one hot encoding ', title_tfidf.shape)
```

```
Shape of matrix after one hot encoding  (109248, 3300)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [33]:

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

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

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

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

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

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

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

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


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

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


'''
```

Out[33]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
```

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

In [34]:

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

In [35]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors.append(vector)

print(len(avg_w2v_vectors))
print(len(avg_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 109248/109248
[00:32<00:00, 3401.20it/s]
```

```
109248
300
```

In [36]:

```
# Similarly you can vectorize for title also
title_avg_w2v_vectors = [];
for sentance in tqdm(project_data['processed_titles']):
    vector = np.zeros(300);
    cnt_words = 0;
    for word in sentance.split():
        if word in glove_words:
            vector += model[word];
            cnt_words += 1;

    if cnt_words != 0:
        vector /= cnt_words;
    title_avg_w2v_vectors.append(vector);

print(len(title_avg_w2v_vectors));
print(len(title_avg_w2v_vectors[0]));
```

```
100%|██████████████████████████████████████████████████████████████████████████| 109248/109248
[00:00<00:00, 112459.49it/s]
```

```
109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [37]:

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

In [38]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors.append(vector)

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

```
100%|██████████████████████████████████████████████████████████████████████████| 109248/109248
[03:41<00:00, 492.63it/s]
```

```
109248
300
```

In [39]:

```
# Similarly you can vectorize for title also
# Similarly you can vectorize for title also
tfidf_model = TfidfVectorizer();
tfidf_model.fit(project_data['processed_titles'].values);
dictionary_idf = dict(zip(tfidf_model.get_feature_names(), tfidf_model.idf_));
tfidf_words = tfidf_model.get_feature_names();
```

In [40]:

```
title_tf_idf_w2v = [];
for sentance in tqdm(project_data['processed_titles']):
    vector = np.zeros(300);
    tf_idf_weight = 0;
    for word in sentance.split():
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word];
            tfidf = dictionary_idf[word] * ( sentance.count(word)/len(sentance.split()) );
            vector = (tfidf * vec);
            tf_idf_weight += tfidf;
    if tf_idf_weight != 0:
        vector /= tf_idf_weight;
    title_tf_idf_w2v.append(vector);

print(len(title_tf_idf_w2v))
```

```
print(len(title_cl_idf_w2v))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 109248/109248 [-
1:58:27<00:00, -1173.94it/s]
```

```
109248
```

1.5.3 Vectorizing Numerical features

In [41]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()
price_scalar.fit(project_data['price'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

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

```
Mean : 298.1193425966608, Standard deviation : 367.49634838483496
```

In [42]:

```
price_standardized
```

Out[42]:

```
array([[ 1.16172762],
       [-0.23153793],
       [ 0.08402983],
       ...,
       [ 0.27450792],
       [-0.0282706 ],
       [-0.79625102]])
```

1.5.4 Merging all the above features

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

In [43]:

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

```
(109248, 9)
(109248, 30)
(109248, 16512)
(109248, 1)
```

In [44]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
```

```
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
X.shape
```

Out[44]:

```
(109248, 16552)
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)
2. **Hyper paramter tuning to find best K**
   - Find the best hyper parameter which results in the maximum AUC value
   - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
   - Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task
3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
   - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points
4. **[Task-2]**
   - Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features
     -
     ```
     from sklearn.datasets import load_digits
     from sklearn.feature_selection import SelectKBest, chi2
     X, y = load_digits(return_X_y=True)
     X.shape
     X_new = SelectKBest(chi2, k=20).fit_transform(X, y)
     X_new.shape
     ========
     output:
     (1797, 64)
     (1797, 20)
     ```
   - Repeat the steps 2 and 3 on the data matrix after feature selection
5. **Conclusion**
   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

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

2. K Nearest Neighbor

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

In [45]:

```
project_data.columns
```

Out[45]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', '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', 'processed_essay',
       'processed_titles', 'price', 'quantity'],
      dtype='object')
```

In [46]:

```
#splitting project_data into x and y, y=project_is_approved.

#fetching all the columns except project_is_approved.
cols_to_select = [col for col in project_data.columns if col != 'project_is_approved'];
X = project_data[cols_to_select]
print(X.columns)
y = project_data['project_is_approved'];
print(y.shape)
```

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'processed_essay', 'processed_titles', 'price',
       'quantity'],
      dtype='object')
(109248,)
```

In [47]:

```
#splitting project_data into train and test and CV data.
X_1, X_test, y_1, y_test = model_selection.train_test_split(X, y, test_size=0.3, random_state=1)
X_train, X_cv, y_train, y_cv = model_selection.train_test_split(X_1, y_1, test_size=0.3, random_sta
te=1);

print('shape of train data ', X_train.shape);
print('shape of test data ', X_test.shape);
print('shape of cross validation data ', X_cv.shape)
```

```
shape of train data  (53531, 19)
shape of test data  (32775, 19)
shape of cross validation data  (22942, 19)
```

Vectorizing categorical values

We have following columns with categorical values:

- school_state
- clean_categories
- clean_subcategories
- project_grade_category
- teacher_prefix

In [48]:

```
#vectorizing school_state
from sklearn.feature_extraction.text import CountVectorizer
```

```
#creating dictionary for school_state as state as keys along with no. of projects from that state
as values.
school_state_dict = dict(X_train['school_state'].value_counts());

#configuring CountVectorizer for school state, in which vocabulary will be name of states.
vectorizer = CountVectorizer(vocabulary=list(school_state_dict.keys()), lowercase=False, binary=Tr
ue);

#applying vectorizer on school_state column to obtain numerical value for each state.
vectorizer.fit(X_train['school_state'].values);

school_state_vector = vectorizer.transform(X_train['school_state'].values);
test_school_state_vector = vectorizer.transform(X_test['school_state'].values);
cv_school_state_vector = vectorizer.transform(X_cv['school_state'].values);

print('shape of matrix after one hot encoding of school_state for train data ',
school_state_vector.shape);
print('shape of matrix after one hot encoding of school_state for test data ',
test_school_state_vector.shape);
print('shape of matrix after one hot encoding of school_state for cv data ',
cv_school_state_vector.shape);
```

```
shape of matrix after one hot encoding of school_state for train data  (53531, 51)
shape of matrix after one hot encoding of school_state for test data  (32775, 51)
shape of matrix after one hot encoding of school_state for cv data  (22942, 51)
```

In [49]:

```
#vectorizing categories

#creating dictionary for clean_categories.
categories_dict = dict(X_train['clean_categories'].value_counts());

vectorizer = CountVectorizer(vocabulary=list(categories_dict.keys()), lowercase=False, binary=True
);

vectorizer.fit(X_train['clean_categories'].values);

categories_vector = vectorizer.transform(X_train['clean_categories'].values);
test_categories_vector = vectorizer.transform(X_test['clean_categories'].values);
cv_categories_vector = vectorizer.transform(X_cv['clean_categories'].values);

print('shape of matrix after one hot encoding of clean_categories for train data',
categories_vector.shape)
print('shape of matrix after one hot encoding of clean_categories for test data',
test_categories_vector.shape)
print('shape of matrix after one hot encoding of clean_categories for cv data',
cv_categories_vector.shape)
```

```
shape of matrix after one hot encoding of clean_categories for train data (53531, 50)
shape of matrix after one hot encoding of clean_categories for test data (32775, 50)
shape of matrix after one hot encoding of clean_categories for cv data (22942, 50)
```

In [50]:

```
#vectorizing subcategories

subcategories_dict = dict(X_train['clean_subcategories'].value_counts());

vectorizer = CountVectorizer(vocabulary=list(categories_dict.keys()), lowercase=False, binary=True
);

vectorizer.fit(X_train['clean_subcategories'].values);

subcategories_vector = vectorizer.transform(X_train['clean_subcategories'].values);
test_subcategories_vector = vectorizer.transform(X_test['clean_subcategories'].values);
cv_subcategories_vector = vectorizer.transform(X_cv['clean_subcategories'].values);

print('shape of matrix after one hot encoding of clean_subcategories for train data',
subcategories_vector.shape)
print('shape of matrix after one hot encoding of clean_subcategories for test data',
test_subcategories_vector.shape)
print('shape of matrix after one hot encoding of clean_subcategories for cv data',
cv_subcategories_vector.shape)
```

```
cv_subcategories_vector.shape)
```

```
shape of matrix after one hot encoding of clean_subcategories for train data (53531, 50)
shape of matrix after one hot encoding of clean_subcategories for test data (32775, 50)
shape of matrix after one hot encoding of clean_subcategories for cv data (22942, 50)
```

In [51]:

```
#vectorizing project_grade_category

grade_dict = dict(X_train['project_grade_category'].value_counts());

vectorizer = CountVectorizer(vocabulary=list(grade_dict.keys()), lowercase=False, binary=True);

vectorizer.fit(X_train['project_grade_category'].values);

grade_vector = vectorizer.transform(X_train['project_grade_category'].values);
test_grade_vector = vectorizer.transform(X_test['project_grade_category'].values);
cv_grade_vector = vectorizer.transform(X_cv['project_grade_category'].values);

print('shape of matrix after one hot encoding of grade_category for train data', grade_vector.shap
e)
print('shape of matrix after one hot encoding of grade_category for test data', test_grade_vector.
shape)
print('shape of matrix after one hot encoding of grade_category for cv data', cv_grade_vector.shap
e)
```

```
shape of matrix after one hot encoding of grade_category for train data (53531, 4)
shape of matrix after one hot encoding of grade_category for test data (32775, 4)
shape of matrix after one hot encoding of grade_category for cv data (22942, 4)
```

In [52]:

```
#vectorizing teacher_prefix

teacher_prefix_dict = dict(X_train['teacher_prefix'].value_counts());

vectorizer = CountVectorizer(vocabulary=list(teacher_prefix_dict.keys()), lowercase=False, binary=
True);

vectorizer.fit(X_train['teacher_prefix'].values.astype('U'));

teacher_prefix_vector = vectorizer.transform(X_train['teacher_prefix'].values.astype('U'));
test_teacher_prefix_vector = vectorizer.transform(X_test['teacher_prefix'].values.astype('U'));
cv_teacher_prefix_vector = vectorizer.transform(X_cv['teacher_prefix'].values.astype('U'));

print('shape of matrix after one hot encoding of teacher_prefix for train data',
teacher_prefix_vector.shape)
print('shape of matrix after one hot encoding of teacher_prefix for test data',
test_teacher_prefix_vector.shape)
print('shape of matrix after one hot encoding of teacher_prefix for cv data',
cv_teacher_prefix_vector.shape)
```

```
shape of matrix after one hot encoding of teacher_prefix for train data (53531, 5)
shape of matrix after one hot encoding of teacher_prefix for test data (32775, 5)
shape of matrix after one hot encoding of teacher_prefix for cv data (22942, 5)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

Encoding Numerical data

we have following columns with numerical data:

- teacher_number_of_previously_posted_projects
- price
- quantity

In [91]:

```
#vectorizing price

from sklearn.preprocessing import Normalizer
price_normalizer = Normalizer()
#configuring StandarScaler to obtain the mean and variance.
price_normalizer.fit(X_train['price'].values.reshape(-1, 1));

# Now standardize the data with maen and variance obtained above.
price_standardized = price_normalizer.transform(X_train['price'].values.reshape(-1, 1))
test_price_standardized = price_normalizer.transform(X_test['price'].values.reshape(-1, 1))
cv_price_standardized = price_normalizer.transform(X_cv['price'].values.reshape(-1, 1))
```

In [92]:

```
#vectorizing teacher_number_of_previously_posted_projects

teacher_normalizer = Normalizer();

teacher_normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1
));

teacher_number_standardized =
teacher_normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape
(-1,1));

test_teacher_number_standardized =
teacher_normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape
(-1,1));

cv_teacher_number_standardized =
teacher_normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(-
1,1));
```

In [104]:

```
#vectorizing quantity:

quantity_normalizer = Normalizer();

quantity_normalizer.fit(X_train['quantity'].values.reshape(-1, 1));

quantity_standardized = quantity_normalizer.transform(X_train['quantity'].values.reshape(-1, 1))

test_quantity_standardized = quantity_normalizer.transform(X_test['quantity'].values.reshape(-1, 1)
)

cv_quantity_standardized = quantity_normalizer.transform(X_cv['quantity'].values.reshape(-1, 1))
```

2.3 Make Data Model Ready: encoding eassay, and project_title

Vectorizing using BOW on train data.

In [56]:

```
#vectorizing essay

#configure CountVectorizer with word to occur in at least 10 documents.
vectorizer = CountVectorizer(min_df=10);

vectorizer.fit(X_train['processed_essay']);

#transforming essay into vector
essay_bow = vectorizer.transform(X_train['processed_essay']);
cv_essay_bow = vectorizer.transform(X_cv['processed_essay']);
test_essay_bow = vectorizer.transform(X_test['processed_essay']);

print('Shape of matrix after one hot encoding for train data: ', essay_bow.shape);
print('Shape of matrix after one hot encoding for test data: ', test_essay_bow.shape);
print('Shape of matrix after one hot encoding for cv data: ', cv_essay_bow.shape);
```

```
Shape of matrix after one hot encoding for train data:  (53531, 12442)
Shape of matrix after one hot encoding for test data:  (32775, 12442)
Shape of matrix after one hot encoding for cv data:  (22942, 12442)
```

In [57]:

```
#vectorizing project_title

#configure CountVectorizer with word to occur in at least 10 documents.
vectorizer = CountVectorizer(min_df=10);

vectorizer.fit(X_train['processed_titles']);

#transforming title into vector
title_bow = vectorizer.transform(X_train['processed_titles']);
cv_title_bow = vectorizer.transform(X_cv['processed_titles']);
test_title_bow = vectorizer.transform(X_test['processed_titles']);

print('Shape of matrix after one hot encoding for train data: ', title_bow.shape);
print('Shape of matrix after one hot encoding for test data: ', test_title_bow.shape);
print('Shape of matrix after one hot encoding for cv data: ', cv_title_bow.shape);
```

```
Shape of matrix after one hot encoding for train data:  (53531, 2186)
Shape of matrix after one hot encoding for test data:  (32775, 2186)
Shape of matrix after one hot encoding for cv data:  (22942, 2186)
```

Vectorizing using tf-idf

In [58]:

```
#vectorizing essay

#importing TfidfVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

#configuring TfidfVectorizer with a word to occur atleast in 10 documnets.
vectorizer = TfidfVectorizer(min_df=10)

vectorizer.fit(X_train['processed_essay']);

#vectorizing essay using tfidf
essay_tfidf = vectorizer.transform(X_train['processed_essay']);
test_essay_tfidf = vectorizer.transform(X_test['processed_essay']);
cv_essay_tfidf = vectorizer.transform(X_cv['processed_essay']);

print("Shape of matrix after one hot encoding for train data: ",essay_tfidf.shape)
print("Shape of matrix after one hot encoding for test data: ",test_essay_tfidf.shape)
print("Shape of matrix after one hot encoding for cv data: ",cv_essay_tfidf.shape)
```

```
Shape of matrix after one hot encoding for train data:  (53531, 12442)
Shape of matrix after one hot encoding for test data:  (32775, 12442)
Shape of matrix after one hot encoding for cv data:  (22942, 12442)
```

In [59]:

```
#vectorizing project_title

vectorizer = TfidfVectorizer(min_df = 10);

vectorizer.fit(X_train['processed_titles']);

title_tfidf = vectorizer.transform(X_train['processed_titles']);
test_title_tfidf = vectorizer.transform(X_test['processed_titles']);
cv_title_tfidf = vectorizer.transform(X_cv['processed_titles']);

print('Shape of title_tfidf after one hot encoding for train data ', title_tfidf.shape)
print('Shape of title_tfidf after one hot encoding for test data ', test_title_tfidf.shape)
print('Shape of title_tfidf after one hot encoding for cv data ', cv_title_tfidf.shape)
```

```
Shape of title_tfidf after one hot encoding for train data  (53531, 2186)
```

```
Shape of title_tfidf after one hot encoding for test data  (32775, 2186)
Shape of title_tfidf after one hot encoding for cv data  (22942, 2186)
```

Vectorizing using avg w2v on train

In [60]:

```
#vectorizing essay

essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['processed_essay']): # 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
    essay_avg_w2v.append(vector)

#printing number of documents
print(len(essay_avg_w2v))

#printing dimension of each essay avg w2v
print(len(essay_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 53531/53531
[00:15<00:00, 3491.65it/s]
```

```
53531
300
```

In [61]:

```
#vectorizing project_title

title_avg_w2v = [];
for sentance in tqdm(X_train['processed_titles']):
    vector = np.zeros(300);
    cnt_words = 0;
    for word in sentance.split():
        if word in glove_words:
            vector += model[word];
            cnt_words += 1;
    if cnt_words != 0:
        vector /= cnt_words;
    title_avg_w2v.append(vector);

print(len(title_avg_w2v));
print(len(title_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 53531/53531
[00:00<00:00, 102412.13it/s]
```

```
53531
300
```

Vectorizing using avg w2v on CV

In [62]:

```
#vectorizing essay

cv_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['processed_essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
```

```
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    cv_essay_avg_w2v.append(vector)

#printing number of documents
print(len(cv_essay_avg_w2v))

#printing dimension of each essay avg w2v
print(len(cv_essay_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 22942/22942
[00:06<00:00, 3434.14it/s]
```

```
22942
300
```

In [63]:

```
#vectorizing project_title

cv_title_avg_w2v = [];
for sentance in tqdm(X_cv['processed_titles']):
    vector = np.zeros(300);
    cnt_words = 0;
    for word in sentance.split():
        if word in glove_words:
            vector += model[word];
            cnt_words += 1;
    if cnt_words != 0:
        vector /= cnt_words;
    cv_title_avg_w2v.append(vector);

print(len(cv_title_avg_w2v));
print(len(cv_title_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 22942/22942
[00:00<00:00, 110361.36it/s]
```

```
22942
300
```

Vectorizing using avg w2v on test data

In [64]:

```
#vectorizing essay

test_essay_avg_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['processed_essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    test_essay_avg_w2v.append(vector)

#printing number of documents
print(len(test_essay_avg_w2v))

#printing dimension of each essay avg w2v
print(len(test_essay_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 32775/32775
```

```
[00:09<00:00, 3423.73it/s]
```

```
32775
300
```

In [65]:

```
#vectorizing project_title

test_title_avg_w2v = [];
for sentance in tqdm(X_test['processed_titles']):
    vector = np.zeros(300);
    cnt_words = 0;
    for word in sentance.split():
        if word in glove_words:
            vector += model[word];
            cnt_words += 1;
    if cnt_words != 0:
        vector /= cnt_words;
    test_title_avg_w2v.append(vector);

print(len(test_title_avg_w2v));
print(len(test_title_avg_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 32775/32775
[00:00<00:00, 105785.87it/s]
```

```
32775
300
```

Vectorizing using tfidf weighted w2v

In [66]:

```
#finding out tfidf words and corresponding idf value for essay

tfidf_model = TfidfVectorizer()

tfidf_model.fit(X_train['processed_essay'])

# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))

tfidf_words = set(tfidf_model.get_feature_names())
```

In [67]:

```
#vectorizing essay

essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['processed_essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    essay_tfidf_w2v.append(vector)

print(len(essay_tfidf_w2v))
print(len(essay_tfidf_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 53531/53531 [01:
```

```
47<00:00, 495.89it/s]
```

```
53531
300
```

In [68]:

```
#vectorizing essay

cv_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['processed_essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    cv_essay_tfidf_w2v.append(vector)

print(len(cv_essay_tfidf_w2v))
print(len(cv_essay_tfidf_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 22942/22942 [00:
46<00:00, 493.89it/s]
```

```
22942
300
```

In [69]:

```
#vectorizing essay

test_essay_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['processed_essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    test_essay_tfidf_w2v.append(vector)

print(len(test_essay_tfidf_w2v))
print(len(test_essay_tfidf_w2v[0]))
```

```
100%|████████████████████████████████████████████████████████████████| 32775/32775 [01:
07<00:00, 499.44it/s]
```

```
32775
300
```

In [70]:

```
#finding out tfidf words and corresponding idf value for project_title

tfidf_model = TfidfVectorizer()
```

```
tfidf_model.fit(X_train['processed_titles'])

# 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 [71]:

```
#vectorizing project_tile

title_tfidf_w2v = [];

for sentance in tqdm(X_train['processed_titles']):
    vector = np.zeros(300);
    tfidf_weight = 0;
    for word in sentance.split():
        if (word in glove_words) and (word in tfidf_words):
            tfidf = dictionary[word] * (sentance.count(word) / len(sentance.split()));
            vector = tfidf * model[word];
            tfidf_weight += tfidf;
    if tfidf_weight != 0:
        vector /= tfidf_weight;
    title_tfidf_w2v.append(vector);

print(len(title_tfidf_w2v))
print(len(title_tfidf_w2v[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 53531/53531
[00:00<00:00, 86669.26it/s]
```

```
53531
300
```

In [72]:

```
#vectorizing project_tile

cv_title_tfidf_w2v = [];

for sentance in tqdm(X_cv['processed_titles']):
    vector = np.zeros(300);
    tfidf_weight = 0;
    for word in sentance.split():
        if (word in glove_words) and (word in tfidf_words):
            tfidf = dictionary[word] * (sentance.count(word) / len(sentance.split()));
            vector = tfidf * model[word];
            tfidf_weight += tfidf;
    if tfidf_weight != 0:
        vector /= tfidf_weight;
    cv_title_tfidf_w2v.append(vector);

print(len(cv_title_tfidf_w2v))
print(len(cv_title_tfidf_w2v[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 22942/22942
[00:00<00:00, 77548.73it/s]
```

```
22942
300
```

In [73]:

```
#vectorizing project_tile

test_title_tfidf_w2v = [];

for sentance in tqdm(X_test['processed_titles']):
    vector = np.zeros(300);
    tfidf_weight = 0;
    for word in sentance.split():
```

```
        if (word in glove_words) and (word in tfidf_words):
            tfidf = dictionary[word] * (sentance.count(word) / len(sentance.split()));
            vector = tfidf * model[word];
            tfidf_weight += tfidf;
    if tfidf_weight != 0:
        vector /= tfidf_weight;
    test_title_tfidf_w2v.append(vector);

print(len(test_title_tfidf_w2v))
print(len(test_title_tfidf_w2v[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████| 32775/32775
[00:00<00:00, 81374.31it/s]
```

```
32775
300
```

Merging data

In [105]:

```
from scipy.sparse import hstack

#concatinating train data
#with bow
train_set_1 = hstack((school_state_vector, categories_vector, subcategories_vector, grade_vector, t
eacher_prefix_vector, price_standardized, teacher_number_standardized, quantity_standardized,
essay_bow, title_bow)).tocsr()

#with tfidf
train_set_2 = hstack((school_state_vector, categories_vector, subcategories_vector, grade_vector, t
eacher_prefix_vector, price_standardized, teacher_number_standardized, quantity_standardized,
essay_tfidf, title_tfidf)).tocsr()

#with avg w2v
train_set_3 = hstack((school_state_vector, categories_vector, subcategories_vector, grade_vector, t
eacher_prefix_vector, price_standardized, teacher_number_standardized, quantity_standardized,
essay_avg_w2v, title_avg_w2v)).tocsr()

#with tfidf wt w2v
train_set_4 = hstack((school_state_vector, categories_vector, subcategories_vector, grade_vector, t
eacher_prefix_vector, price_standardized, teacher_number_standardized, quantity_standardized,
essay_tfidf_w2v, title_tfidf_w2v)).tocsr()

#concatinating cv data
#with bow
cv_set_1 = hstack((cv_school_state_vector, cv_categories_vector, cv_subcategories_vector,
cv_grade_vector, cv_teacher_prefix_vector, cv_price_standardized, cv_teacher_number_standardized,
cv_quantity_standardized, cv_essay_bow, cv_title_bow)).tocsr()

#with tfidf
cv_set_2 = hstack((cv_school_state_vector, cv_categories_vector, cv_subcategories_vector,
cv_grade_vector, cv_teacher_prefix_vector, cv_price_standardized, cv_teacher_number_standardized,
cv_quantity_standardized, cv_essay_tfidf, cv_title_tfidf)).tocsr()

#with avg w2v
cv_set_3 = hstack((cv_school_state_vector, cv_categories_vector, cv_subcategories_vector,
cv_grade_vector, cv_teacher_prefix_vector, cv_price_standardized, cv_teacher_number_standardized,
cv_quantity_standardized, cv_essay_avg_w2v, cv_title_avg_w2v)).tocsr()

#with tfidf wt w2v
cv_set_4 = hstack((cv_school_state_vector, cv_categories_vector, cv_subcategories_vector,
cv_grade_vector, cv_teacher_prefix_vector, cv_price_standardized, cv_teacher_number_standardized,
cv_quantity_standardized, cv_essay_tfidf_w2v, cv_title_tfidf_w2v)).tocsr()

#concatinating test data
#with bow
test_set_1 = hstack((test_school_state_vector, test_categories_vector, test_subcategories_vector,
test_grade_vector, test_teacher_prefix_vector, test_price_standardized,
test_teacher_number_standardized, test_quantity_standardized, test_essay_bow,
test_title_bow)).tocsr()

#with tfidf
```

```
test_set_2 = hstack((test_school_state_vector, test_categories_vector, test_subcategories_vector,
test_grade_vector, test_teacher_prefix_vector, test_price_standardized,
test_teacher_number_standardized, test_quantity_standardized, test_essay_tfidf, test_title_tfidf))
.tocsr()

#with avg w2v
test_set_3 = hstack((test_school_state_vector, test_categories_vector, test_subcategories_vector,
test_grade_vector, test_teacher_prefix_vector, test_price_standardized,
test_teacher_number_standardized, test_quantity_standardized, test_essay_avg_w2v,
test_title_avg_w2v)).tocsr()

#with tfidf wt w2v
test_set_4 = hstack((test_school_state_vector, test_categories_vector, test_subcategories_vector,
test_grade_vector, test_teacher_prefix_vector, test_price_standardized,
test_teacher_number_standardized, test_quantity_standardized, test_essay_tfidf_w2v,
test_title_tfidf_w2v)).tocsr()
```

In [106]:

```
print(train_set_1.shape, cv_set_1.shape, test_set_1.shape)
print(train_set_2.shape, cv_set_2.shape, test_set_2.shape)
print(train_set_3.shape, cv_set_3.shape, test_set_3.shape)
print(train_set_4.shape, cv_set_4.shape, test_set_4.sh
      ape)
```

```
(53531, 14791) (22942, 14791) (32775, 14791)
(53531, 14791) (22942, 14791) (32775, 14791)
(53531, 763) (22942, 763) (32775, 763)
(53531, 763) (22942, 763) (32775, 763)
```

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 [95]:

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        pred = clf.predict_proba(data[i:i+1000])
        y_data_pred.extend(pred[:,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 [96]:

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

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

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

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

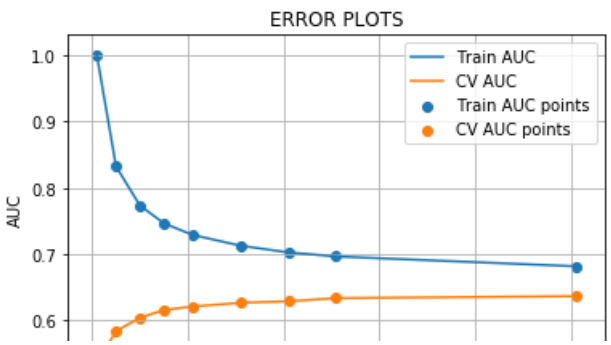
```
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

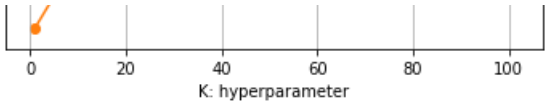
2.4.1 Applying KNN brute force on BOW, SET 1

In [107]:

```
# Please write all the code with proper documentation
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

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

"""

train_auc = []
cv_auc = []

#defining list of K we will use.
K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
print(train_set_1.shape, y_train.shape, cv_set_1.shape, y_cv.shape)

#Performing KNN classification with all the k, to find the best value of K
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(train_set_1, y_train)

    y_train_pred = batch_predict(neigh, train_set_1)
    y_cv_pred = batch_predict(neigh, cv_set_1)

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

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

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

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

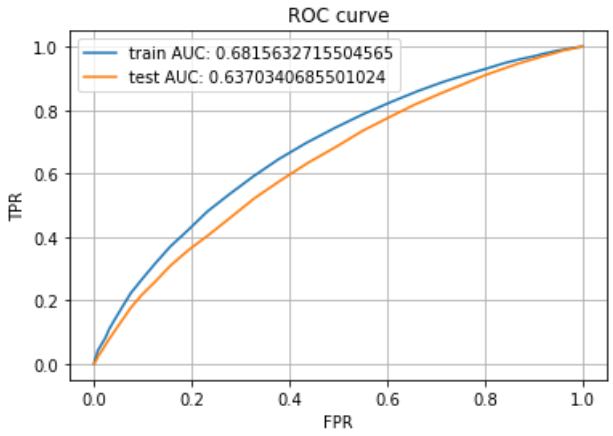
```
(53531, 14791) (53531,) (22942, 14791) (22942,)
```





In [108]:

```
#selecting the best K value.
optimal_k = K[cv_auc.index(max(cv_auc))]

set1_k = optimal_k;

#configuring knn with best k value
knn = KNeighborsClassifier(n_neighbors = optimal_k)
knn.fit(train_set_1, y_train);

y_train_pred = batch_predict(knn, train_set_1)
y_test_pred = batch_predict(knn, test_set_1)

train_fpr, train_tpr, train_threshold = metrics.roc_curve(y_train,  y_train_pred)
train_auc = metrics.roc_auc_score(y_train, y_train_pred)

test_fpr, test_tpr, test_threshold = metrics.roc_curve(y_test, y_test_pred)
test_auc = metrics.roc_auc_score(y_test, y_test_pred);

set1_auc = test_auc;

plt.plot(train_fpr, train_tpr, label="train AUC: "+str(train_auc))
plt.plot(test_fpr, test_tpr, label="test AUC: "+str(test_auc))


plt.grid();
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend();
plt.show()
```



In [109]:

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

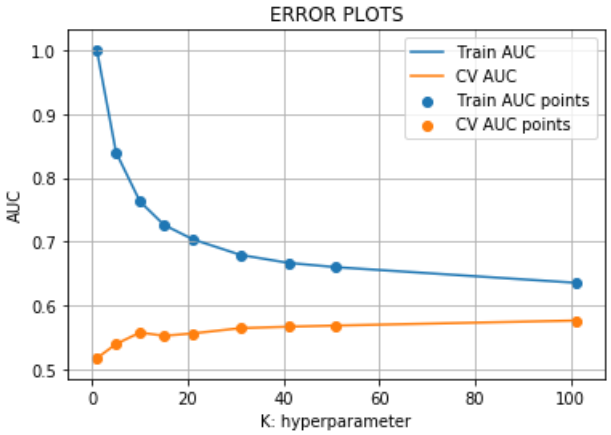
```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.40142088620590527 for threshold 0.782
[[ 5046  3071]
 [16089 29325]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.35745745655623506 for threshold 0.792
[[ 3017  1865]
 [11759 16134]]
```

2.4.2 Applying KNN brute force on TFIDF, SET 2

In [110]:

```
train_auc = []
cv_auc = []

K = [1, 5, 10, 15, 21, 31, 41, 51, 101]

print(train_set_2.shape, y_train.shape, cv_set_2.shape, y_cv.shape)

for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(train_set_2, y_train)

    y_train_pred = batch_predict(neigh, train_set_2)
    y_cv_pred = batch_predict(neigh, cv_set_2)

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

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

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

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

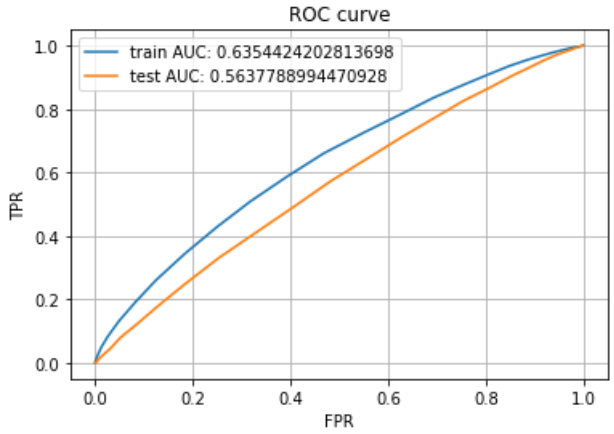
```
(53531, 14791) (53531,) (22942, 14791) (22942,)
```



In [111]:

```
#selecting the best K value.
optimal_k = K[cv_auc.index(max(cv_auc))]
set2_k = optimal_k;


#configuring knn with best k value
knn = KNeighborsClassifier(n_neighbors = optimal_k)
knn.fit(train_set_2, y_train);

y_train_pred = batch_predict(knn, train_set_2)
y_test_pred = batch_predict(knn, test_set_2)

train_fpr, train_tpr, train_threshold = metrics.roc_curve(y_train,  y_train_pred)
```

```
train_auc = metrics.roc_auc_score(y_train, y_train_pred)

test_fpr, test_tpr, test_threshold = metrics.roc_curve(y_test, y_test_pred)
test_auc = metrics.roc_auc_score(y_test, y_test_pred);

set2_auc = test_auc;

plt.plot(train_fpr, train_tpr, label="train AUC: "+str(train_auc))
plt.plot(test_fpr, test_tpr, label="test AUC: "+str(test_auc))


plt.grid();
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend();
plt.show()
```



In [112]:

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

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.3564147626729982 for threshold 0.851
[[ 4932  3185]
 [18775 26639]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.2965914412824698 for threshold 0.861
[[ 2888  1994]
 [14123 13770]]
```

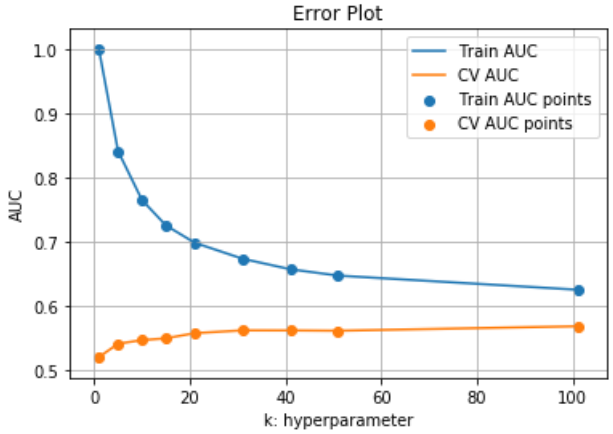
2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [113]:

```
train_auc = []
cv_auc = []


K = [1, 5, 10, 15, 21, 31, 41, 51, 101]

print(train_set_3.shape, y_train.shape, cv_set_3.shape, y_cv.shape)

for i in K:
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(train_set_3, y_train)

    y_train_pred = batch_predict(knn, train_set_3);
```

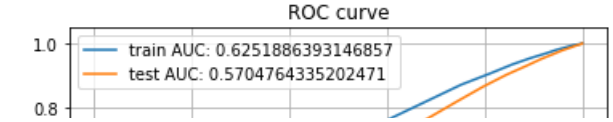
```
    y_cv_pred = batch_predict(knn, cv_set_3);

    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 Plot');
plt.grid();
plt.show();
```

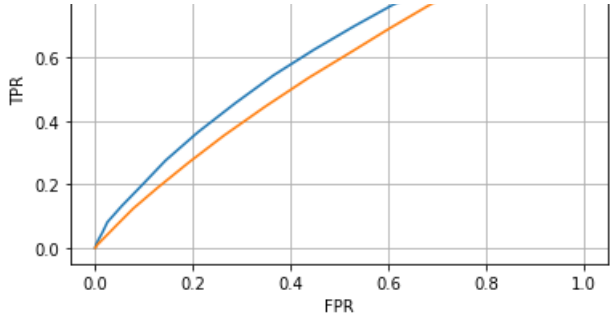
```
(53531, 763) (53531,) (22942, 763) (22942,)
```



In [114]:

```
#selecting the best K value.
optimal_k = K[cv_auc.index(max(cv_auc))]
set3_k = optimal_k;

#configuring knn with best k value
knn = KNeighborsClassifier(n_neighbors = optimal_k)
knn.fit(train_set_3, y_train);

y_train_pred = batch_predict(knn, train_set_3)
y_test_pred = batch_predict(knn, test_set_3)

train_fpr, train_tpr, train_threshold = metrics.roc_curve(y_train,  y_train_pred)
train_auc = metrics.roc_auc_score(y_train, y_train_pred)

test_fpr, test_tpr, test_threshold = metrics.roc_curve(y_test, y_test_pred)
test_auc = metrics.roc_auc_score(y_test, y_test_pred);

set3_auc = test_auc

plt.plot(train_fpr, train_tpr, label="train AUC: "+str(train_auc))
plt.plot(test_fpr, test_tpr, label="test AUC: "+str(test_auc))


plt.grid();
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend();
plt.show()
```





In [115]:

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

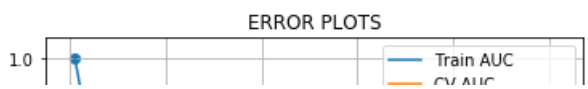
```
====================================================================================================

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.34533353881020007 for threshold 0.861
[[ 5149  2968]
 [20691 24723]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.30167680193813745 for threshold 0.871
[[ 3182  1700]
 [15472 12421]]
```

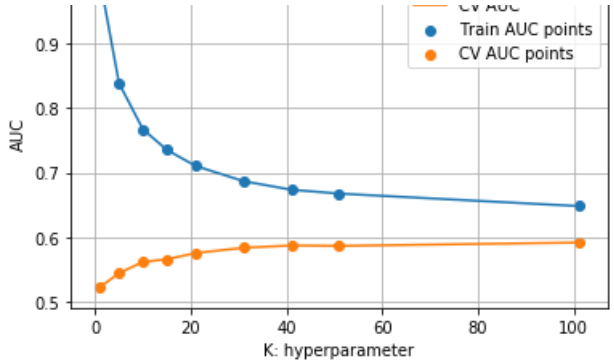
2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [116]:

```
train_auc = []
cv_auc = []
K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
print(train_set_4.shape, y_train.shape, cv_set_4.shape, y_cv.shape)
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(train_set_4, y_train)

    y_train_pred = batch_predict(neigh, train_set_4)
    y_cv_pred = batch_predict(neigh, cv_set_4)

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

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

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

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

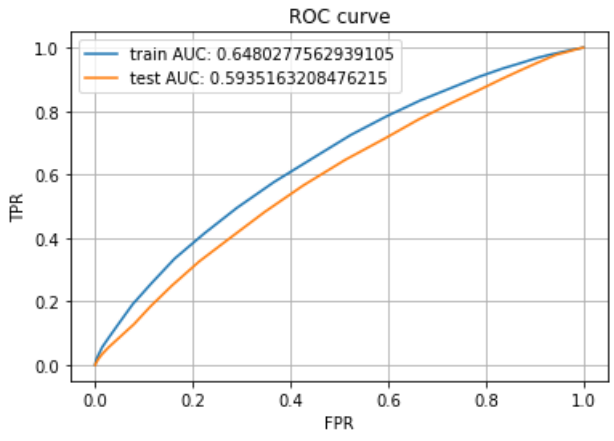
```
(53531, 763) (53531,) (22942, 763) (22942,)
```





In [117]:

```
#selecting the best K value.
optimal_k = K[cv_auc.index(max(cv_auc))]
set4_k = optimal_k

#configuring knn with best k value
knn = KNeighborsClassifier(n_neighbors = optimal_k)
knn.fit(train_set_4, y_train);

y_train_pred = batch_predict(knn, train_set_4)
y_test_pred = batch_predict(knn, test_set_4)

train_fpr, train_tpr, train_threshold = metrics.roc_curve(y_train,  y_train_pred)
train_auc = metrics.roc_auc_score(y_train, y_train_pred)

test_fpr, test_tpr, test_threshold = metrics.roc_curve(y_test, y_test_pred)
test_auc = metrics.roc_auc_score(y_test, y_test_pred);

set4_auc = test_auc

plt.plot(train_fpr, train_tpr, label="train AUC: "+str(train_auc))
plt.plot(test_fpr, test_tpr, label="test AUC: "+str(test_auc))


plt.grid();
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend();
plt.show()
```



In [118]:

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

==========

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.3652297755967671 for threshold 0.851
[[ 5142  2975]
 [19231 26183]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3245404069654428 for threshold 0.851
[[ 2804  2078]
 [12132 15761]]
```

2.5 Feature selection with `SelectKBest`

In [119]:

```
#selecting best 2000 features from set2 containing tidf
from sklearn.feature_selection import SelectKBest, chi2

selector = SelectKBest(chi2, k=2000)

selector.fit(train_set_2, y_train)

new_train_set_2 = selector.transform(train_set_2)
new_test_set_2 = selector.transform(test_set_2)
new_cv_set_2 = selector.transform(cv_set_2)
print('Train data: Shape of older data: {}, shape of new data: {}'.format(train_set_2.shape, new_tr
ain_set_2.shape))
print('Test data: Shape of older data: {}, shape of new data: {}'.format(test_set_2.shape, new_test
_set_2.shape))
print('CV data: Shape of older data: {}, shape of new data: {}'.format(cv_set_2.shape, new_cv_set_2
.shape))
```

```
Train data: Shape of older data: (53531, 14791), shape of new data: (53531, 2000)
Test data: Shape of older data: (32775, 14791), shape of new data: (32775, 2000)
CV data: Shape of older data: (22942, 14791), shape of new data: (22942, 2000)
```

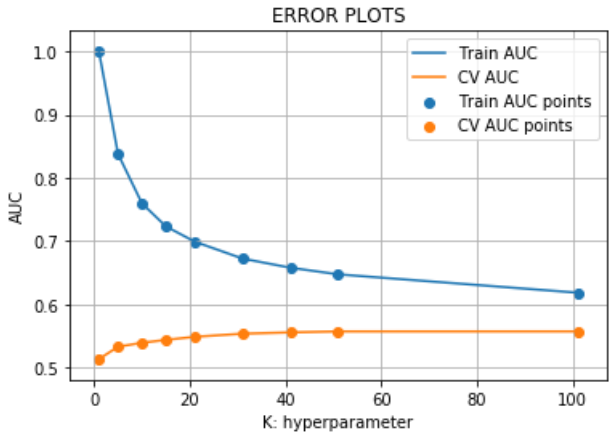
Applying knn on set2 containing new features

In [120]:

```
train_auc = []
cv_auc = []

#defining list of K we will use.
K = [1, 5, 10, 15, 21, 31, 41, 51, 101]
print(new_train_set_2.shape, y_train.shape, new_cv_set_2.shape, y_cv.shape)

#Performing KNN classification with all the k, to find the best value of K
for i in K:
    neigh = KNeighborsClassifier(n_neighbors=i)
    neigh.fit(new_train_set_2, y_train)

    y_train_pred = batch_predict(neigh, new_train_set_2)
    y_cv_pred = batch_predict(neigh, new_cv_set_2)

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

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

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

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

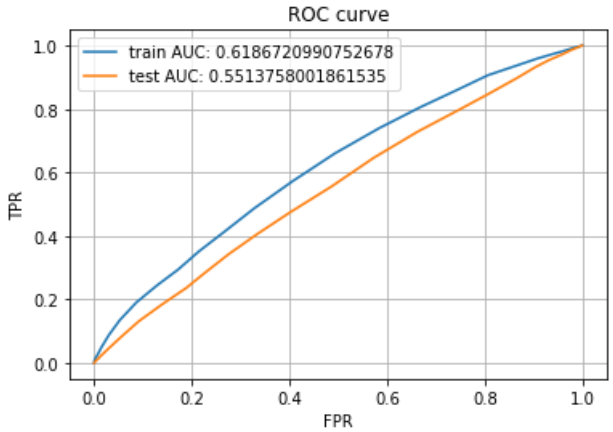
```
plt.show()
```

```
(53531, 2000) (53531,) (22942, 2000) (22942,)
```



In [121]:

```
#selecting the best K value.
optimal_k = K[cv_auc.index(max(cv_auc))]
new_set2_k = optimal_k;


#configuring knn with best k value
knn = KNeighborsClassifier(n_neighbors = optimal_k)
knn.fit(new_train_set_2, y_train);

y_train_pred = batch_predict(knn, new_train_set_2)
y_test_pred = batch_predict(knn, new_test_set_2)

train_fpr, train_tpr, train_threshold = metrics.roc_curve(y_train,  y_train_pred)
train_auc = metrics.roc_auc_score(y_train, y_train_pred)

test_fpr, test_tpr, test_threshold = metrics.roc_curve(y_test, y_test_pred)
test_auc = metrics.roc_auc_score(y_test, y_test_pred);

new_set2_auc = test_auc;

plt.plot(train_fpr, train_tpr, label="train AUC: "+str(train_auc))
plt.plot(test_fpr, test_tpr, label="test AUC: "+str(test_auc))


plt.grid();
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend();
plt.show()
```



In [122]:

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

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

Train confusion matrix
the maximum value of tpr*(1-fpr) 0.3399264594430947 for threshold 0.842
[[ 4820  3297]
 [19417 25997]]
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.28571658215225904 for threshold 0.842
[[ 2509  2373]
 [12386 15507]]
```

3. Conclusions

In [128]:

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

table = PrettyTable();
table.field_names = ['Vectorizer', 'Model', 'Hyper parameter', 'AUC'];

table.add_row(['BOW', 'Brute', set1_k, set1_auc]);
table.add_row(['TFIDF', 'Brute', set2_k, set2_auc]);
table.add_row(['W2V', 'Brute', set3_k, set3_auc]);
table.add_row(['TFIDFW2V', 'Brute', set4_k, set4_auc]);
table.add_row(['TFID with top 2000 features', 'Brute', new_set2_k, new_set2_auc]);

print(table)
```

```
+------------------------------+-------+-----------------+--------------------+
|          Vectorizer          | Model | Hyper parameter |        AUC         |
+------------------------------+-------+-----------------+--------------------+
|             BOW              | Brute |       101       | 0.6370340685501024 |
|            TFIDF             | Brute |       101       | 0.5637788994470928 |
|             W2V              | Brute |       101       | 0.5704764335202471 |
|           TFIDFW2V           | Brute |       101       | 0.5935163208476215 |
| TFID with top 2000 features | Brute |       101       | 0.5513758001861535 |
+------------------------------+-------+-----------------+--------------------+
```

In [ ]:

```
```