# DonorsChoose

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

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

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

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

## About the DonorsChoose Data Set

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

| Feature | |
|---|---|
| **`project_id`** | A unique identifier for the proposed proj |
| **`project_title`** | Title<br>• `Art Wi`<br>• |
| **`project_grade_category`** | Grade level of students for which the project is targe<br>•<br>•<br>•<br>• |

| Feature | |
|---|---|
| **`project_subject_categories`** | One or more (comma-separated) subject categorie following e<br>•<br>•<br>•<br>•<br>• I<br>•<br>•<br>•<br>•<br>•<br>• `Literacy & Langua` |
| **`school_state`** | State where school is located [(Tw (https://en.wikipedia.org/wiki/List_of_U.S._state_abbr](https://en.wikipedia.org/wiki/List_of_U.S._state_abbr) |
| **`project_subject_subcategories`** | One or more (comma-separated) subject subc<br>•<br>• `Literature & Writin` |
| **`project_resource_summary`** | An explanation of the resources needed f<br>• `My students need hands on literacy m se` |
| **`project_essay_1`** | |
| **`project_essay_2`** | S |
| **`project_essay_3`** | |
| **`project_essay_4`** | |
| **`project_submitted_datetime`** | Datetime when project application was submitted. **E** |
| **`teacher_id`** | A unique identifier for the teacher of the pro `bdf8baa8fedef6` |
| **`teacher_prefix`** | Teacher's title. One of the follo<br>•<br>•<br>•<br>•<br>•<br>• |
| **`teacher_number_of_previously_posted_projects`** | Number of project applications previously submi |

* 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
```

## 1.1 Reading Data

In [2]:

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

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

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

In [4]:
```
# how to replace elements in list python: https://stackoverflow.com/a/25821
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(proj


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datet
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/1314861
project_data = project_data[cols]


# Taking 15k points from project data
project_data = project_data.head(50000)
```

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://stackoverf

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

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunge
    for j in i.split(','): # it will split it in three parts ["Math & Scien
        if 'The' in j.split(): # this will split each of the catogory based
            j=j.replace('The','') # if we have the words "The" we are going
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the
        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/2289
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]:

| ategory | project_title | project_essay_1 | project_essay_2 | project_essay_3 | project_essay_4 | project_reso |
|---|---|---|---|---|---|---|
| s PreK-2 | Engineering STEAM into the Primary Classroom | I have been fortunate enough to use the Fairy ... | My students come from a variety of backgrounds... | Each month I try to do several science or STEM... | It is challenging to develop high quality scie... | My students to |
| ades 3-5 | Sensory Tools for Focus | Imagine being 8-9 years old. You're in your th... | Most of my students have autism, anxiety, anot... | It is tough to do more than one thing at a tim... | When my students are able to calm themselves d... | My studer Boards fı |

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[2000])
print("="*50)
print(project_data['essay'].values[9999])
print("="*50)
```

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classr
oom as well as the STEM journals, which my students really enjoyed.  I wo
uld love to implement more of the Lakeshore STEM kits 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 so
cioeconomic status.  Many of them don't have a lot of experience in scien
ce and engineering and these kits give me the materials to provide these
exciting opportunities for my students.Each month I try to do several sci
ence or STEM/STEAM projects.  I would use the kits and robot to help guid
e my science instruction in engaging and meaningful ways.  I can adapt th
e kits to my current language arts pacing guide where we already teach so
me of the material in the kits like tall tales (Paul Bunyan) or Johnny Ap
pleseed.  The following units will be taught in the next school year wher
e 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 challengi
ng to develop high quality science activities.  These kits give me the ma
terials I need to provide my students with science activities that will g
o 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 h
ow to use them in an appropriate way.
==================================================
I teach high school English to students with learning and behavioral disa
bilities. My students all vary in their ability level. However, the ultim
ate goal is to increase all students literacy levels. This includes their
reading, writing, and communication levels.I teach a really dynamic group
of students. However, my students face a lot of challenges. My students a
ll live in poverty and in a dangerous neighborhood. Despite these challen
ges, I have students who have the the desire to defeat these challenges.
My students all have learning disabilities and currently all are performi
ng below grade level. My students are visual learners and will benefit fr
om a classroom that fulfills their preferred learning style.The materials
I am requesting will allow my students to be prepared for the classroom w
ith the necessary supplies.  Too often I am challenged with students who
come to school unprepared for class due to economic challenges.  I want m
y 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 chart paper will be used to make learning more visual in class and to
create posters to aid students in their learning.  The students have acce
ss to a classroom printer.  The toner will be used to print student work
that is completed on the classroom Chromebooks.I want to try and remove a
ll barriers for the students learning and create opportunities for learni
```

```
ng. One of the biggest barriers is the students not having the resources
to get pens, paper, and folders. My students will be able to increase the
ir literacy skills because of this project.
==================================================
\"Life moves pretty fast. If you don't stop and look around once in awhil
e, you could miss it.\"  from the movie, Ferris Bueller's Day Off.  Think
back...what do you remember about your grandparents?  How amazing would i
t be to be able to flip through a book to see a day in their lives?My sec
ond graders are voracious readers! They love to read both fiction and non
fiction books.  Their favorite characters include Pete the Cat, Fly Guy,
Piggie and Elephant, and Mercy Watson. They also love to read about insec
ts, space and plants. My students are hungry bookworms! My students are e
ager 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. Thei
r parents work long hours and usually do not see their children. My stude
nts 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... would
n'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 p
recious to us and being able to share these memories with future generati
ons will be a rewarding experience.  As part of our social studies curric
ulum, students will be learning about changes over time.  Students will b
e studying photos to learn about how their community has changed over tim
e.  In particular, we will look at photos to study how the land, building
s, clothing, and schools have changed over time.  As a culminating activi
ty, my students will capture a slice of their history and preserve it thr
ough scrap booking. Key important events in their young lives will be doc
umented with the date, location, and names.   Students will be using phot
os from home and from school to create their second grade memories.   The
ir scrap books will preserve their unique stories for future generations
to enjoy.Your donation to this project will provide my second graders wit
h an opportunity to learn about social studies in a fun and creative mann
er.  Through their scrapbooks, children will share their story with other
s and have a historical document for the rest of their lives.
==================================================
\"Creativity is intelligence having fun.\" --Albert Einstein. Our element
ary library at Greenville Elementary is anything but a quiet, hushed spac
e. It's a place for collaboration and research. It's a place for incorpor
ating technology. It's a place for innovation. And it's a place for creat
ing.Our school serves 350 third and fourth graders who primarily live in
rural and poverty-stricken areas in our community. Being a Title I schoo
l, approximately 85% of them receive free or reduced lunch. But they are
inquisitive, creative, and eager to learn. They love visiting the library
to check out books, hear \r\nstories, create digital stories, and use the
computer lab for learning and fun. We want to build our library's Makersp
ace with activities revolving around art and literacy to provide more eng
aging, hands-on activities.We want to begin \"Makerspace Fridays!\" Our s
chool recently received a $1000 grant for books for our arts-integrated M
akerspace. We have received titles such as \"Origami for Everyone,\" \"Ho
w to Make Stuff with Ducktape,\" and \"Cool Engineering Activities for Gi
rls.\"  We now need supplies to correlate with these new informational te
xts. By adding these art and craft supplies, students will be able to des
ign and create masterpieces related to their coursework. \r\n\r\nFor exam
ple, while studying Native Americans, students can use the looms and yarn
to recreate Navajo and/or Pueblo weaving. Weaving can also be integrated
with literacy through Greek mythology and the story of Arachne.\r\n\r\nCr
```

```
eating art with perler beads has many possibilities! Students can design
their own animals after studying their characteristics. They can use symm
etry and patterning to create one-of-a-kind originals. \r\n\r\nOrigami re
inforces geometry, thinking skills, fractions, problem-solving, and just
fun science!Our students need to be able to apply what they read and lear
n. If they read a how-to book, they will apply that reading through a han
ds-on art activity and actually create a product. This is a crucial skill
in the real world. By creating and designing their own masterpieces, they
are using many critical thinking skills. Students will become more analyt
ical thinkers.
==================================================
\"Can you be our teacher next year, you made Science FUN?\" asked a stude
nt.  \"I would love to, however, I have been reassigned to 3rd grade EL
A,\" I brokenheartedly replied.\r\n\r\nLet's rewind to 10 weeks ago...\r
\n\r\n     After an 8 month medical leave, I was assigned to complete the
academic year as a Middle School Science teacher, quite a difference from
my last position as a 2nd grade teacher. Long story short, I was terrife
d.  Middle School? Science? Well, I dove in!    \r\n    Since I am not a
certified Science teacher, I had to follow a modified curriculum.  The cu
rriculum consisted of videos, textbooks, and worksheets.  Just reading th
e lesson plans, caused me, the TEACHER, to loose interest in the subject
matter.  Moreover, I contacted the principal and asked if I can modify th
e lessons.  And, so I did!\r\n    Students created metamorphic rocks by a
pplying heat and pressure to pop rocks and gum.  Students went on scaveng
er hunts, and even created menus through the perspective of a predator.\r
\n     My future students will be 8/9 year olds of Hispanic, African, Hai
tian, and Jamaican descent that live in an urban setting.  Involving stud
ents in many multi-sensory experiences inspires them to WANT to LEARN!I w
ould love for my students to have the opportunity to work with a document
camera.  This document camera would not only infuse technology into our l
essons, it would assist in featuring students' works and create comradery
between peers during their peer revisions. In addition, it would help dis
play text excerpts and serve as a means of differentiating reading strate
gies to my students.\r\n     Learning should be FUN!  SO, I have purchase
d 2 bar stools for our \"Computer Island\", 2 yoga balls for our \"Stress
-Free Writing Zone\", 2 stability wiggle cushions for our \"Moving Forwar
d Zone\", 2 long body pillows \"Relaxation Spot\", and a document camera
would make our class even more complete!  Thank you =)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[2000])
print(sent)
print("="*50)
```

```
\"Creativity is intelligence having fun.\" --Albert Einstein. Our element
ary library at Greenville Elementary is anything but a quiet, hushed spac
e. It is a place for collaboration and research. It is a place for incorp
orating technology. It is a place for innovation. And it is a place for c
reating.Our school serves 350 third and fourth graders who primarily live
in rural and poverty-stricken areas in our community. Being a Title I sch
ool, approximately 85% of them receive free or reduced lunch. But they ar
e inquisitive, creative, and eager to learn. They love visiting the libra
ry to check out books, hear \r\nstories, create digital stories, and use
the computer lab for learning and fun. We want to build our library is Ma
kerspace with activities revolving around art and literacy to provide mor
e engaging, hands-on activities.We want to begin \"Makerspace Fridays!\"
Our school recently received a $1000 grant for books for our arts-integra
ted Makerspace. We have received titles such as \"Origami for Everyone,\"
\"How to Make Stuff with Ducktape,\" and \"Cool Engineering Activities fo
r Girls.\"  We now need supplies to correlate with these new informationa
l texts. By adding these art and craft supplies, students will be able to
design and create masterpieces related to their coursework. \r\n\r\nFor e
xample, while studying Native Americans, students can use the looms and y
arn to recreate Navajo and/or Pueblo weaving. Weaving can also be integra
ted with literacy through Greek mythology and the story of Arachne.\r\n\r
\nCreating art with perler beads has many possibilities! Students can des
ign their own animals after studying their characteristics. They can use
symmetry and patterning to create one-of-a-kind originals. \r\n\r\nOrigam
i reinforces geometry, thinking skills, fractions, problem-solving, and j
ust fun science!Our students need to be able to apply what they read and
learn. If they read a how-to book, they will apply that reading through a
hands-on art activity and actually create a product. This is a crucial sk
ill in the real world. By creating and designing their own masterpieces,
they are using many critical thinking skills. Students will become more a
nalytical thinkers.
==================================================
```

In [14]:
```
# \r \n \t remove from string python: http://texthandler.com/info/remove-li
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

```
 Creativity is intelligence having fun.  --Albert Einstein. Our elementar
y library at Greenville Elementary is anything but a quiet, hushed space.
It is a place for collaboration and research. It is a place for incorpora
ting technology. It is a place for innovation. And it is a place for crea
ting.Our school serves 350 third and fourth graders who primarily live in
rural and poverty-stricken areas in our community. Being a Title I schoo
l, approximately 85% of them receive free or reduced lunch. But they are
inquisitive, creative, and eager to learn. They love visiting the library
to check out books, hear   stories, create digital stories, and use the c
omputer lab for learning and fun. We want to build our library is Makersp
ace with activities revolving around art and literacy to provide more eng
aging, hands-on activities.We want to begin  Makerspace Fridays!  Our sch
ool recently received a $1000 grant for books for our arts-integrated Mak
erspace. We have received titles such as  Origami for Everyone,   How to
Make Stuff with Ducktape,  and  Cool Engineering Activities for Girls.
We now need supplies to correlate with these new informational texts. By
adding these art and craft supplies, students will be able to design and
create masterpieces related to their coursework.     For example, while s
tudying Native Americans, students can use the looms and yarn to recreate
Navajo and/or Pueblo weaving. Weaving can also be integrated with literac
y through Greek mythology and the story of Arachne.    Creating art with
perler beads has many possibilities! Students can design their own animal
s after studying their characteristics. They can use symmetry and pattern
ing to create one-of-a-kind originals.     Origami reinforces geometry, t
hinking skills, fractions, problem-solving, and just fun science!Our stud
ents need to be able to apply what they read and learn. If they read a ho
w-to book, they will apply that reading through a hands-on art activity a
nd actually create a product. This is a crucial skill in the real world.
By creating and designing their own masterpieces, they are using many cri
tical thinking skills. Students will become more analytical thinkers.
```

In [15]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

```
 Creativity is intelligence having fun Albert Einstein Our elementary lib
rary at Greenville Elementary is anything but a quiet hushed space It is
a place for collaboration and research It is a place for incorporating te
chnology It is a place for innovation And it is a place for creating Our
school serves 350 third and fourth graders who primarily live in rural an
d poverty stricken areas in our community Being a Title I school approxim
ately 85 of them receive free or reduced lunch But they are inquisitive c
reative and eager to learn They love visiting the library to check out bo
oks hear stories create digital stories and use the computer lab for lear
ning and fun We want to build our library is Makerspace with activities r
evolving around art and literacy to provide more engaging hands on activi
ties We want to begin Makerspace Fridays Our school recently received a 1
000 grant for books for our arts integrated Makerspace We have received t
itles such as Origami for Everyone How to Make Stuff with Ducktape and Co
ol Engineering Activities for Girls We now need supplies to correlate wit
h these new informational texts By adding these art and craft supplies st
udents will be able to design and create masterpieces related to their co
ursework For example while studying Native Americans students can use the
looms and yarn to recreate Navajo and or Pueblo weaving Weaving can also
be integrated with literacy through Greek mythology and the story of Arac
hne Creating art with perler beads has many possibilities Students can de
sign their own animals after studying their characteristics They can use
symmetry and patterning to create one of a kind originals Origami reinfor
ces geometry thinking skills fractions problem solving and just fun scien
ce Our students need to be able to apply what they read and learn If they
read a how to book they will apply that reading through a hands on art ac
tivity and actually create a product This is a crucial skill in the real
world By creating and designing their own masterpieces they are using man
y critical thinking skills Students will become more analytical thinkers
```

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', 'y
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', '
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its',
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this',
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have'
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'be
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on'
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', '
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "sho
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't",
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "sh
            '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%|██████████| 50000/50000 [00:25<00:00, 1993.65it/s]
```

In [18]:

```
# after preprocesing
preprocessed_essays[2000]
```

Out[18]:

```
'creativity intelligence fun albert einstein elementary library greenvill
e elementary anything quiet hushed space place collaboration research pla
ce incorporating technology place innovation place creating school serves
350 third fourth graders primarily live rural poverty stricken areas comm
unity title school approximately 85 receive free reduced lunch inquisitiv
e creative eager learn love visiting library check books hear stories cre
ate digital stories use computer lab learning fun want build library make
rspace activities revolving around art literacy provide engaging hands ac
tivities want begin makerspace fridays school recently received 1000 gran
t books arts integrated makerspace received titles origami everyone make
stuff ducktape cool engineering activities girls need supplies correlate
new informational texts adding art craft supplies students able design cr
eate masterpieces related coursework example studying native americans st
udents use looms yarn recreate navajo pueblo weaving weaving also integra
ted literacy greek mythology story arachne creating art perler beads many
possibilities students design animals studying characteristics use symmet
ry patterning create one kind originals origami reinforces geometry think
ing skills fractions problem solving fun science students need able apply
read learn read book apply reading hands art activity actually create pro
duct crucial skill real world creating designing masterpieces using many
critical thinking skills students become analytical thinkers'
```

## 1.4 Preprocessing of `project_title`

In [19]:
```
# similarly you can preprocess the titles also
# Combining all the above stundents
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████| 50000/50000 [00:01<00:00, 45751.77it/s]
```

In [20]:
```
# after preprocesing
preprocessed_titles [2000]
```

Out[20]:
```
'empowering students art makerspace'
```

## 1.5 Preparing data for models

In [21]:
```
project_data.columns
```

Out[21]:
```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_stat
e',
       '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_approv
ed',
       'clean_categories', 'clean_subcategories', 'essay'],
      dtype='object')
```

we are going to consider

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

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

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

### 1.5.1 Vectorizing Categorical data

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

In [22]:

```
# 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()), lower
categories_one_hot = vectorizer.fit_transform(project_data['clean_categorie
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearnin
g', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (50000, 9)
```

In [23]:

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

```
['Warmth', 'Care_Hunger', 'FinancialLiteracy', 'Economics', 'ParentInvolv
ement', 'CommunityService', 'Extracurricular', 'ForeignLanguages', 'Civic
s_Government', 'PerformingArts', 'TeamSports', 'SocialSciences', 'Charact
erEducation', 'NutritionEducation', 'College_CareerPrep', 'Other', 'Musi
c', 'History_Geography', 'ESL', 'Health_LifeScience', 'EarlyDevelopment',
'EnvironmentalScience', 'VisualArts', 'Gym_Fitness', 'AppliedSciences',
'SpecialNeeds', 'Health_Wellness', 'Literature_Writing', 'Mathematics',
'Literacy']
Shape of matrix after one hot encodig  (50000, 30)
```

In [24]:
```
# you can do the similar thing with state, teacher_prefix and project_grade

# Feature encoding for state

vectorizer = CountVectorizer(lowercase=False, binary=True)
print (project_data['school_state'].head(5))
vectorizer.fit(project_data['school_state'].values)
print (vectorizer.get_feature_names())

states_one_hot = vectorizer.transform(project_data['school_state'].values)
print("Shape of matrix after one hot encodig ",states_one_hot.shape)
```

```
55660    CA
76127    UT
51140    CA
473      GA
41558    WA
Name: school_state, dtype: object
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI',
'IA', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN',
'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY', 'OH',
'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA',
'WI', 'WV', 'WY']
Shape of matrix after one hot encodig  (50000, 51)
```

In [25]:
```
print ('Nan Values:',project_data['teacher_prefix'].isnull().sum())
# Replacing the NaN values with most frequently used value of teacher prefi

project_data.loc[project_data['teacher_prefix'].isnull(),'teacher_prefix']=
print ('After Imputing:',project_data['teacher_prefix'].isnull().sum())


vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(project_data['teacher_prefix'])
print (vectorizer.get_feature_names())

teacher_prfx_one_hot = vectorizer.transform(project_data['teacher_prefix'])
print("Shape of matrix after one hot encodig ",teacher_prfx_one_hot.shape)
```

```
Nan Values: 2
After Imputing: 0
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher']
Shape of matrix after one hot encodig  (50000, 5)
```

In [26]:
```
my_counter = Counter()
for word in project_data['project_grade_category'].values:
    my_counter.update(word.split(','))

prjctgrd_dict = dict(my_counter)
sorted_prjctgrd_dict = dict(sorted(prjctgrd_dict.items(), key=lambda kv: kv

vectorizer = CountVectorizer(vocabulary=list(sorted_prjctgrd_dict.keys()),l
vectorizer.fit(project_data['project_grade_category'].values)
print (vectorizer.get_feature_names())

project_grade_category_one_hot = vectorizer.transform(project_data['project
print("Shape of matrix after one hot encodig ",project_grade_category_one_h
```

```
['Grades 9-12', 'Grades 6-8', 'Grades 3-5', 'Grades PreK-2']
Shape of matrix after one hot encodig  (50000, 4)
```

## 1.5.2 Vectorizing Text data

### 1.5.2.1 Bag of words

In [27]:
```
# We are considering only the words which appeared in at least 10 documents
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  (50000, 12016)
```

In [28]:
```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it

vectorizer = CountVectorizer(min_df=10)
titles_bow = vectorizer.fit_transform(preprocessed_titles)
print("Shape of matrix after one hot encodig ",titles_bow.shape)
```

```
Shape of matrix after one hot encodig  (50000, 1960)
```

### 1.5.2.2 TFIDF vectorizer

In [29]:
```
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  (50000, 12016)
```

In [30]:

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

```
Shape of matrix after one hot encodig  (50000, 1960)
```

#### 1.5.2.3 Using Pretrained Models: Avg W2V

In [31]:

```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/408
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 c
      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/

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


'''
```

Out[31]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/3823034
```

```
9/4084039\ndef (https://stackoverflow.com/a/38230349/4084039\ndef) loadGl
oveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(glo
veFile,\'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        mode
l[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n
return model\nmodel = loadGloveModel(\'glove.42B.300d.txt\')\n\n# =======
=====================\nOutput:\n    \nLoading Glove Model\n1917495it [06:
32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n# =====================
=======\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.spli
t(\' \'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))
\nprint("all the words in the coupus", len(words))\nwords = set(words)\np
rint("the unique words in the coupus", len(words))\n\ninter_words = set(m
odel.keys()).intersection(words)\nprint("The number of words that are pre
sent in both glove vectors and our coupus",      len(inter_words),"(",n
p.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nw
ords_glove = set(model.keys())\nfor i in words:\n    if i in words_glov
e:\n        words_courpus[i] = model[i]\nprint("word 2 vec length", len(w
ords_courpus))\n\n\n# stronging variables into pickle files python: htt
p://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-p
ython/\n\nimport (http://www.jessicayung.com/how-to-use-pickle-to-save-an
d-load-variables-in-python/\n\nimport) pickle\nwith open(\'glove_vectors
\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [32]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/
# 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 [33]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in t
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%|██████████| 50000/50000 [00:12<00:00, 3951.38it/s]

50000
300
```

### 1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [34]:

```
# 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 [35]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in
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/re
    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 t
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.sp
            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%|██████████| 50000/50000 [01:29<00:00, 556.49it/s]

50000
300
```

In [36]:

```
# Similarly you can vectorize for title also

# average Word2Vec
# compute average word2vec for each title.
avg_title_w2v_vectors = []; # the avg-w2v for each sentence/review is store
for sentence in tqdm(preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_title_w2v_vectors.append(vector)

print(len(avg_title_w2v_vectors))
print(len(avg_title_w2v_vectors[0]))
```

```
100%|██████████| 50000/50000 [00:00<00:00, 57150.30it/s]

50000
300
```

In [37]:
```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_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())

# average Word2Vec
# compute tfidf word2vec for each review.
tfidf_title_w2v_vectors = []; # the avg-w2v for each sentence/review is stc
for sentence in tqdm(preprocessed_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/re
    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 t
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.sp
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_title_w2v_vectors.append(vector)

print(len(tfidf_title_w2v_vectors))
print(len(tfidf_title_w2v_vectors[0]))
```

```
100%|██████████| 50000/50000 [00:01<00:00, 30935.14it/s]

50000
300
```

## 1.5.3 Vectorizing Numerical features

In [38]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'su
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [39]:

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

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

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

# Now standardize the data with above maen and variance.
price_standardized = price_scalar.transform(project_data['price'].values.re
```

```
Mean : 313.3845596, Standard deviation : 372.91313012251806
```

In [40]:

```
price_standardized
```

Out[40]:

```
array([[ 1.1039178 ],
       [-0.26910975],
       [ 0.0418742 ],
       ...,
       [-0.6661191 ],
       [-0.82267031],
       [-0.70631613]])
```

### 1.5.4 Merging all the above features

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

In [41]:

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

```
(50000, 9)
(50000, 30)
(50000, 12016)
(50000, 1)
```

In [42]:

```
# 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
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_sta
X.shape
```

Out[42]:

```
(50000, 12056)
```

# Assignment 3: Apply KNN

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

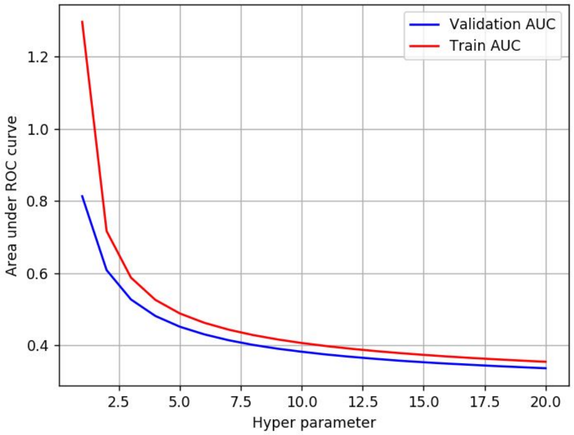
    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
    - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
    - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

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

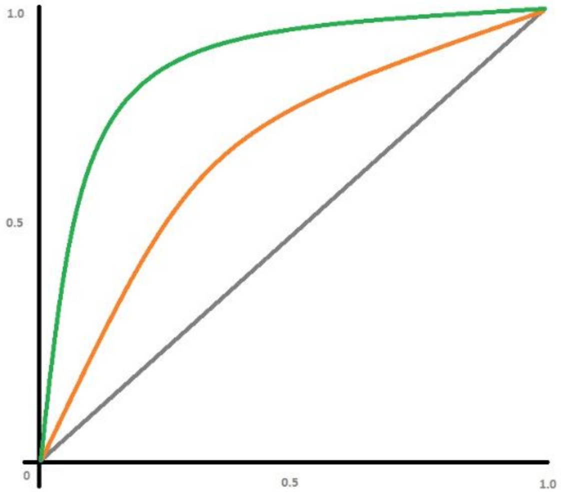
    - Find the best hyper parameter which results in the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
    - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data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

3. **Representation of results**

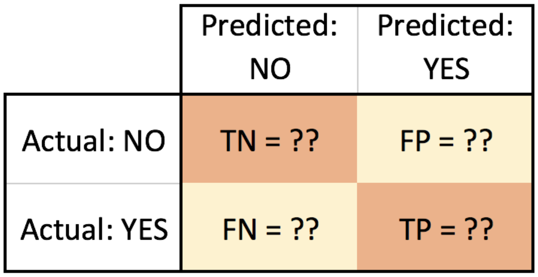
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure



    - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.



- Along with plotting ROC curve, you need to print the [confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points



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

4. **[Task-2]**

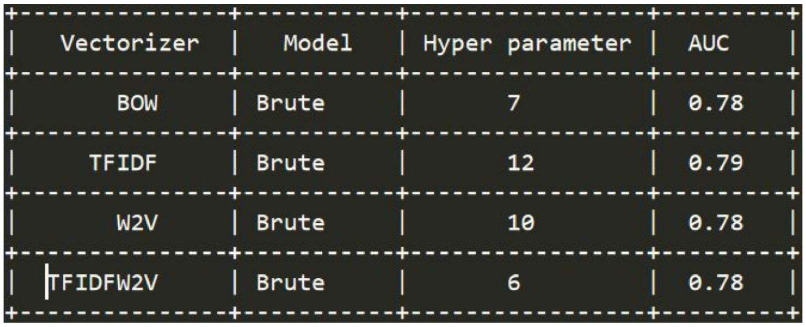
- Select top 2000 features from feature Set 2 using [`SelectKBest` (https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html)](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) and then apply KNN on top of these features

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

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

5. **Conclusion**

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



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

### Note: Data Leakage

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

# 2. K Nearest Neighbor

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

In [43]:

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

# Importing required libraries

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn import model_selection

# preparing the data matrix with all the required features


# split the data set into train and test with 70% train and 30% test
project_data_1, project_data_test = model_selection.train_test_split(projec

# split the train data set into cross validation train and cross validation
project_data_tr, project_data_cv = model_selection.train_test_split(project

print (project_data_tr.shape)
print (project_data_cv.shape)
print (project_data_test.shape)
```

```
(24500, 20)
(10500, 20)
(15000, 20)
```

## 2.2 Make Data Model Ready: encoding numerical, categorical features

In [44]:

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

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

# =============================Encoding Categorical features only on Train

print("="*75)
print("Encoding Categorical features on Train Data (project_data_tr)")
print("="*75)

# *************Encoding categories

from sklearn.feature_extraction.text import CountVectorizer
cat_vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), l
categories_one_hot_train = cat_vectorizer.fit_transform(project_data_tr['cl
# print(cat_vectorizer.get_feature_names())
print("Shape of Train categories (categories_one_hot_train) ",categories_on

# ************Encoding sub categories

sub_cat_vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.ke
sub_categories_one_hot_train = sub_cat_vectorizer.fit_transform(project_dat
# print(sub_cat_vectorizer.get_feature_names())
print("Shape of Train subcategories (sub_categories_one_hot_train) ",sub_ca

# ************Encoding school state

school_state_vectorizer = CountVectorizer(lowercase=False, binary=True)
states_one_hot_train = school_state_vectorizer.fit_transform(project_data_t
# print (school_state_vectorizer.get_feature_names())
print("Shape of school state (states_one_hot_train) ",states_one_hot_train.


# **************Encoding teacher prefix

# print ('Nan Values:',project_data_tr['teacher_prefix'].isnull().sum())
# Replacing the NaN values with most frequently used value of teacher prefi

# project_data.loc[project_data_tr['teacher_prefix'].isnull(),'teacher_pref
# print ('After Imputing:',project_data_tr['teacher_prefix'].isnull().sum()


tc_prefix_vectorizer = CountVectorizer(lowercase=False, binary=True)
tc_prefix_vectorizer.fit(project_data_tr['teacher_prefix'])
# print (tc_prefix_vectorizer.get_feature_names())

teacher_prfx_one_hot_train = tc_prefix_vectorizer.transform(project_data_tr
print("Shape of teacher prefix (teacher_prfx_one_hot_train) ",teacher_prfx_
# ***************Encoding project grade category

my_counter = Counter()
for word in project_data_tr['project_grade_category'].values:
    my_counter.update(word.split(','))

prjctgrd_dict = dict(my_counter)
sorted_prjctgrd_dict = dict(sorted(prjctgrd_dict.items(), key=lambda kv: kv

grade_vectorizer = CountVectorizer(vocabulary=list(sorted_prjctgrd_dict.key
grade_vectorizer.fit(project_data_tr['project_grade_category'].values)
# print (grade_vectorizer.get_feature_names())

project_grade_category_one_hot_train = grade_vectorizer.transform(project_d
print("Shape of grade (project_grade_category_one_hot_train) ",project_grad


# ****************Numerical features

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

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

price_scalar = StandardScaler()
price_scalar.fit(project_data_tr['price'].values.reshape(-1,1)) # finding t
# print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(pri

# Now standardize the data with above maen and variance.
price_standardized_train = price_scalar.transform(project_data_tr['price'].
print("Shape of price standardized (price_standardized_train) ",price_stand


# =============================Encoding Categorical features on cross valid

print("="*75)
print("Encoding Categorical features on cross validate Data (project_data_c
print("="*75)

# *************Encoding categories

categories_one_hot_cv = cat_vectorizer.transform(project_data_cv['clean_cat
# print(cat_vectorizer.get_feature_names())
print("Shape of cv categories (categories_one_hot_cv) ",categories_one_hot_

# ************Encoding sub categories

sub_categories_one_hot_cv = sub_cat_vectorizer.transform(project_data_cv['c
# print(sub_cat_vectorizer.get_feature_names())
print("Shape of cv subcategories (sub_categories_one_hot_cv) ",sub_categori

# ************Encoding school state
```

```
states_one_hot_cv = school_state_vectorizer.transform(project_data_cv['scho
# print (school_state_vectorizer.get_feature_names())
print("Shape of school state (states_one_hot_cv) ",states_one_hot_cv.shape)


# **************Encoding teacher prefix

# print ('Nan Values:',project_data_cv['teacher_prefix'].isnull().sum())
# Replacing the NaN values with most frequently used value of teacher prefi

# project_data.loc[project_data_cv['teacher_prefix'].isnull(),'teacher_pref
# print ('After Imputing:',project_data_cv['teacher_prefix'].isnull().sum()


teacher_prfx_one_hot_cv = tc_prefix_vectorizer.transform(project_data_cv['t
print("Shape of teacher prefix (teacher_prfx_one_hot_cv) ",teacher_prfx_one

# ***************Encoding project grade category

my_counter = Counter()
for word in project_data_cv['project_grade_category'].values:
    my_counter.update(word.split(','))

prjctgrd_dict = dict(my_counter)
sorted_prjctgrd_dict = dict(sorted(prjctgrd_dict.items(), key=lambda kv: kv

project_grade_category_one_hot_cv = grade_vectorizer.transform(project_data
print("Shape of grade (project_grade_category_one_hot_cv) ",project_grade_c


# *****************Numerical features

price_scalar = StandardScaler()
price_scalar.fit(project_data_cv['price'].values.reshape(-1,1)) # finding t
# print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(pri

# Now standardize the data with above maen and variance.
price_standardized_cv = price_scalar.transform(project_data_cv['price'].val
print("Shape of price standardized (price_standardized_cv)",price_standardi


# =============================Encoding Categorical features on Test Data (

print("="*75)
print("Encoding Categorical features on Test Data (project_data_test)")
print("="*75)

# *************Encoding categories

categories_one_hot_test = cat_vectorizer.transform(project_data_test['clean
# print(cat_vectorizer.get_feature_names())
print("Shape of test categories (categories_one_hot_test) ",categories_one_

# ************Encoding sub categories

sub_categories_one_hot_test = sub_cat_vectorizer.transform(project_data_tes
# print(sub_cat_vectorizer.get_feature_names())
print("Shape of test subcategories (sub_categories_one_hot_test) ",sub_cate

# ************Encoding school state

states_one_hot_test = school_state_vectorizer.transform(project_data_test['
# print (school_state_vectorizer.get_feature_names())
print("Shape of school state (states_one_hot_test) ",states_one_hot_test.sh


# **************Encoding teacher prefix

# print ('Nan Values:',project_data_test['teacher_prefix'].isnull().sum())
# Replacing the NaN values with most frequently used value of teacher prefi

# project_data.loc[project_data_test['teacher_prefix'].isnull(),'teacher_pr
# print ('After Imputing:',project_data_test['teacher_prefix'].isnull().sum


teacher_prfx_one_hot_test = tc_prefix_vectorizer.transform(project_data_tes
print("Shape of teacher prefix (teacher_prfx_one_hot_test) ",teacher_prfx_o

# ***************Encoding project grade category

my_counter = Counter()
for word in project_data_test['project_grade_category'].values:
    my_counter.update(word.split(','))

prjctgrd_dict = dict(my_counter)
sorted_prjctgrd_dict = dict(sorted(prjctgrd_dict.items(), key=lambda kv: kv

project_grade_category_one_hot_test = grade_vectorizer.transform(project_da
print("Shape of grade (project_grade_category_one_hot_test)  ",project_grad


# *****************Numerical features

price_scalar = StandardScaler()
price_scalar.fit(project_data_test['price'].values.reshape(-1,1)) # finding
# print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(pri

# Now standardize the data with above maen and variance.
price_standardized_test = price_scalar.transform(project_data_test['price']
print("Shape of price standardized (price_standardized_test) ",price_standa
```

```
==========================================================================
==
Encoding Categorical features on Train Data (project_data_tr)
==========================================================================
==
Shape of Train categories (categories_one_hot_train)  (24500, 9)
Shape of Train subcategories (sub_categories_one_hot_train)  (24500, 30)
Shape of school state (states_one_hot_train)  (24500, 51)
Shape of teacher prefix (teacher_prfx_one_hot_train)  (24500, 5)
Shape of grade (project_grade_category_one_hot_train)  (24500, 4)
Shape of price standardized (price_standardized_train)  (24500, 1)
=========================================================================
==
Encoding Categorical features on cross validate Data (project_data_cv)
=========================================================================
==
Shape of cv categories (categories_one_hot_cv)  (10500, 9)
Shape of cv subcategories (sub_categories_one_hot_cv)  (10500, 30)
Shape of school state (states_one_hot_cv)  (10500, 51)
Shape of teacher prefix (teacher_prfx_one_hot_cv)  (10500, 5)
Shape of grade (project_grade_category_one_hot_cv)  (10500, 4)
Shape of price standardized (price_standardized_cv) (10500, 1)
=========================================================================
==
Encoding Categorical features on Test Data (project_data_test)
=========================================================================
==
Shape of test categories (categories_one_hot_test)  (15000, 9)
Shape of test subcategories (sub_categories_one_hot_test)  (15000, 30)
Shape of school state (states_one_hot_test)  (15000, 51)
Shape of teacher prefix (teacher_prfx_one_hot_test)  (15000, 5)
Shape of grade (project_grade_category_one_hot_test)   (15000, 4)
Shape of price standardized (price_standardized_test)  (15000, 1)
```

## 2.3 Make Data Model Ready: encoding eassay, and project_title

In [45]:

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

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

# =============================Encoding eassay, and project_title only on T

print("="*75)
print("Encoding eassay, and project_title only on Train Data (project_data_
print("="*75)

# ********** Text prepocessing on essays and titles from Train Datapreproce
preprocessed_train_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data_tr['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_train_essays.append(sent.lower().strip())


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


# **************** BOW For Essays

# We are considering only the words which appeared in at least 10 documents
bow_essay_vectorizer = CountVectorizer(min_df=10)
train_text_bow = bow_essay_vectorizer.fit_transform(preprocessed_train_essa
print("BOW == Shape of Train Data Text encoding (train_text_bow) ",train_te

# **************** BOW For Titles
bow_title_vectorizer = CountVectorizer(min_df=10)
train_titles_bow = bow_title_vectorizer.fit_transform(preprocessed_train_ti
print("BOW == Shape of Train Data Title encoding (train_titles_bow) ",train
# **************** TFIDF For Essays
tfidf_essay_vectorizer = TfidfVectorizer(min_df=10)
train_text_tfidf = tfidf_essay_vectorizer.fit_transform(preprocessed_train_
print("TFID == Shape of Train Data Text encoding (train_text_tfidf) ",train

# **************** TFIDF For Titles
tfidf_title_vectorizer = TfidfVectorizer(min_df=10)
train_title_tfidf = tfidf_title_vectorizer.fit_transform(preprocessed_train
print("TFIDF == Shape of Train Data Title encoding (train_title_tfidf)  ",t



#  *************** Average Word2Vec for Essays

# compute average word2vec for each review.
avg_w2v_train_essay_vectors = []; # the avg-w2v for each sentence/review is
for sentence in tqdm(preprocessed_train_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_train_essay_vectors.append(vector)

print ("==== Avg W2V for text (avg_w2v_train_essay_vectors) ====")
print(len(avg_w2v_train_essay_vectors))
print(len(avg_w2v_train_essay_vectors[0]))


# ******************* TFIDF W2V For Essays

tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_train_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())

# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_train_essay_vectors = []; # the avg-w2v for each sentence/review
for sentence in tqdm(preprocessed_train_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/re
    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 t
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.sp
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_train_essay_vectors.append(vector)
```

```
print ("==== TFIDF W2V for text (tfidf_w2v_train_essay_vectors) ====")
print(len(tfidf_w2v_train_essay_vectors))
print(len(tfidf_w2v_train_essay_vectors[0]))


#  *************** Average Word2Vec for Titles

# compute average word2vec for each title.
avg_w2v_train_title_vectors = []; # the avg-w2v for each sentence/review is
for sentence in tqdm(preprocessed_train_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_train_title_vectors.append(vector)

print ("==== Avg W2V for title (avg_w2v_train_title_vectors) ====")
print(len(avg_w2v_train_title_vectors))
print(len(avg_w2v_train_title_vectors[0]))


# ******************* TFIDF W2V For Titles

tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_train_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())

# compute tfidf word2vec for each review.
tfidf_w2v_train_title_vectors = []; # the avg-w2v for each sentence/review
for sentence in tqdm(preprocessed_train_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/re
    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 t
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.sp
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_train_title_vectors.append(vector)

print ("==== TFIDF W2V for title (tfidf_w2v_train_title_vectors) ====")
print(len(tfidf_w2v_train_title_vectors))
print(len(tfidf_w2v_train_title_vectors[0]))


# =============================Encoding eassay, and project_title only on c
```

```
print("="*75)
print("Encoding eassay, and project_title only on cv Data (project_data_cv)
print("="*75)

# ********** Text prepocessing on essays and titles from cv Datapreprocesse

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


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


# **************** BOW For Essays

# We are considering only the words which appeared in at least 10 documents
cv_text_bow = bow_essay_vectorizer.transform(preprocessed_cv_essays)
print("BOW == Shape of cv Data Text encoding (cv_text_bow) ",cv_text_bow.sh

# **************** BOW For Titles
cv_titles_bow = bow_title_vectorizer.transform(preprocessed_cv_titles)
print("BOW == Shape of cv Data Title encoding (cv_titles_bow) ",cv_titles_b

# **************** TFIDF For Essays
cv_text_tfidf = tfidf_essay_vectorizer.transform(preprocessed_cv_essays)
print("TFID == Shape of cv Data Text encoding (cv_text_tfidf) ",cv_text_tfi

# **************** TFIDF For Titles
cv_title_tfidf = tfidf_title_vectorizer.transform(preprocessed_cv_titles)
print("TFIDF == Shape of cv Data Title encoding (cv_title_tfidf)  ",cv_titl



#  *************** Average Word2Vec for Essays

# compute average word2vec for each review.
avg_w2v_cv_essay_vectors = []; # the avg-w2v for each sentence/review is st
for sentence in tqdm(preprocessed_cv_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_cv_essay_vectors.append(vector)

print ("==== Avg W2V for text (avg_w2v_cv_essay_vectors) ====")
print(len(avg_w2v_cv_essay_vectors))
print(len(avg_w2v_cv_essay_vectors[0]))


# ******************* TFIDF W2V For Essays

tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_cv_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())

# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_cv_essay_vectors = []; # the avg-w2v for each sentence/review is
for sentence in tqdm(preprocessed_cv_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/re
    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 t
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.sp
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_cv_essay_vectors.append(vector)

print ("==== TFIDF W2V for text (tfidf_w2v_cv_essay_vectors) ====")
print(len(tfidf_w2v_cv_essay_vectors))
print(len(tfidf_w2v_cv_essay_vectors[0]))


#  *************** Average Word2Vec for Titles

# compute average word2vec for each title.
avg_w2v_cv_title_vectors = []; # the avg-w2v for each sentence/review is st
for sentence in tqdm(preprocessed_cv_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_cv_title_vectors.append(vector)

print ("==== Avg W2V for title (avg_w2v_cv_title_vectors) ====")
print(len(avg_w2v_cv_title_vectors))
print(len(avg_w2v_cv_title_vectors[0]))


# ******************* TFIDF W2V For Titles

tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_cv_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())

# compute tfidf word2vec for each review.
tfidf_w2v_cv_title_vectors = []; # the avg-w2v for each sentence/review is
for sentence in tqdm(preprocessed_cv_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/re
    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 t
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.sp
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_cv_title_vectors.append(vector)

print ("==== TFIDF W2V for title (tfidf_w2v_cv_title_vectors) ====")
print(len(tfidf_w2v_cv_title_vectors))
print(len(tfidf_w2v_cv_title_vectors[0]))


# =============================Encoding eassay, and project_title only on t

print("="*75)
print("Encoding eassay, and project_title only on test Data (project_data_t
print("="*75)

# ********** Text prepocessing on essays and titles from test Datapreproces
preprocessed_test_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data_test['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_test_essays.append(sent.lower().strip())
```

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


# **************** BOW For Essays

# We are considering only the words which appeared in at least 10 documents
test_text_bow = bow_essay_vectorizer.transform(preprocessed_test_essays)
print("BOW == Shape of test Data Text encoding (test_text_bow) ",test_text_

# **************** BOW For Titles
test_titles_bow = bow_title_vectorizer.transform(preprocessed_test_titles)
print("BOW == Shape of test Data Title encoding (test_titles_bow) ",test_ti

# **************** TFIDF For Essays
test_text_tfidf = tfidf_essay_vectorizer.transform(preprocessed_test_essays
print("TFID == Shape of test Data Text encoding (test_text_tfidf) ",test_te

# **************** TFIDF For Titles
test_title_tfidf = tfidf_title_vectorizer.transform(preprocessed_test_title
print("TFIDF == Shape of test Data Title encoding (test_title_tfidf)  ",tes



#  *************** Average Word2Vec for Essays

# compute average word2vec for each review.
avg_w2v_test_essay_vectors = []; # the avg-w2v for each sentence/review is
for sentence in tqdm(preprocessed_test_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_test_essay_vectors.append(vector)

print ("==== Avg W2V for text (avg_w2v_test_essay_vectors) ====")
print(len(avg_w2v_test_essay_vectors))
print(len(avg_w2v_test_essay_vectors[0]))


# ******************* TFIDF W2V For Essays

tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_test_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())

# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_test_essay_vectors = []; # the avg-w2v for each sentence/review i
for sentence in tqdm(preprocessed_test_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/re
    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 t
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.sp
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_test_essay_vectors.append(vector)

print ("==== TFIDF W2V for text (tfidf_w2v_test_essay_vectors) ====")
print(len(tfidf_w2v_test_essay_vectors))
print(len(tfidf_w2v_test_essay_vectors[0]))


#  *************** Average Word2Vec for Titles

# compute average word2vec for each title.
avg_w2v_test_title_vectors = []; # the avg-w2v for each sentence/review is
for sentence in tqdm(preprocessed_test_titles): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_test_title_vectors.append(vector)

print ("==== Avg W2V for title (avg_w2v_test_title_vectors) ====")
print(len(avg_w2v_test_title_vectors))
print(len(avg_w2v_test_title_vectors[0]))


# ******************* TFIDF W2V For Titles

tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_test_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())

# compute tfidf word2vec for each review.
tfidf_w2v_test_title_vectors = []; # the avg-w2v for each sentence/review i
for sentence in tqdm(preprocessed_test_titles): # for each review/sentence
```

```
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/re
    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 t
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.sp
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_test_title_vectors.append(vector)

print ("==== TFIDF W2V for title (tfidf_w2v_test_title_vectors) ====")
print(len(tfidf_w2v_test_title_vectors))
print(len(tfidf_w2v_test_title_vectors[0]))
```

```
  1%|          | 179/24500 [00:00<00:13, 1787.27it/s]

=======================================================================
====
Encoding eassay, and project_title only on Train Data (project_data_tr)
=======================================================================
====

100%|██████████| 24500/24500 [00:12<00:00, 1900.44it/s]
100%|██████████| 24500/24500 [00:00<00:00, 42307.47it/s]

BOW == Shape of Train Data Text encoding (train_text_bow)  (24500, 902
3)
BOW == Shape of Train Data Title encoding (train_titles_bow)  (24500, 1
190)

  0%|          | 0/24500 [00:00<?, ?it/s]

TFID == Shape of Train Data Text encoding (train_text_tfidf)  (24500, 9
023)
TFIDF == Shape of Train Data Title encoding (train_title_tfidf)  (2450
```

In [46]:
```
# Data size of encoded essays and titles on Train, CV and Test datas

print ("="*75)
print("Train Data")
print ("="*75)

print ("BOW on Essay (train_text_bow) ",train_text_bow.shape)
print ("BOW on title (train_titles_bow) ",train_titles_bow.shape)

print ("TFIDF on Essay (train_text_tfidf)", train_text_tfidf.shape)
print ("TFIDF on Title (train_title_tfidf)", train_title_tfidf.shape)

print ("Avg W2V on Essay (avg_w2v_train_essay_vectors)",len (avg_w2v_train_
print ("Avg W2V on Title (avg_w2v_train_title_vectors)",len (avg_w2v_train_

print ("TFIDF W2V on Essay (tfidf_w2v_train_essay_vectors)",len (tfidf_w2v_
print ("TFIDF W2V on Title (tfidf_w2v_train_title_vectors)",len (tfidf_w2v_

print ("="*75)
print("CV Data")
print ("="*75)

print ("BOW on Essay (cv_text_bow) ",cv_text_bow.shape)
print ("BOW on Title (cv_titles_bow) ",cv_titles_bow.shape)

print ("TFIDF on Essay (cv_text_tfidf)", cv_text_tfidf.shape)
print ("TFIDF on Title (cv_title_tfidf)", cv_title_tfidf.shape)

print ("Avg W2V on Essay (avg_w2v_cv_essay_vectors)",len (avg_w2v_cv_essay_
print ("Avg W2V on Title (avg_w2v_cv_title_vectors)",len (avg_w2v_cv_title_

print ("TFIDF W2V on Essay (tfidf_w2v_cv_essay_vectors)",len (tfidf_w2v_cv_
print ("TFIDF W2V on Title (tfidf_w2v_cv_title_vectors)",len (tfidf_w2v_cv_

print ("="*75)
print("TEST Data")
print ("="*75)

print ("BOW on Essay (test_text_bow) ",test_text_bow.shape)
print ("BOW on title (test_titles_bow) ",test_titles_bow.shape)

print ("TFIDF on Essay (test_text_tfidf)", test_text_tfidf.shape)
print ("TFIDF on Title (test_title_tfidf)", test_title_tfidf.shape)

print ("Avg W2V on Essay (avg_w2v_test_essay_vectors)",len (avg_w2v_test_es
print ("Avg W2V on Title (avg_w2v_test_title_vectors)",len (avg_w2v_test_ti

print ("TFIDF W2V on Essay (tfidf_w2v_test_essay_vectors)",len (tfidf_w2v_t
print ("TFIDF W2V on Title (tfidf_w2v_test_title_vectors)",len (tfidf_w2v_t
```

```
=========================================================================
==
Train Data
=========================================================================
==
BOW on Essay (train_text_bow)  (24500, 9023)
BOW on title (train_titles_bow)  (24500, 1190)
TFIDF on Essay (train_text_tfidf) (24500, 9023)
TFIDF on Title (train_title_tfidf) (24500, 1190)
Avg W2V on Essay (avg_w2v_train_essay_vectors) 24500 300
Avg W2V on Title (avg_w2v_train_title_vectors) 24500 300
TFIDF W2V on Essay (tfidf_w2v_train_essay_vectors) 24500 300
TFIDF W2V on Title (tfidf_w2v_train_title_vectors) 24500 300
=========================================================================
==
CV Data
=========================================================================
==
BOW on Essay (cv_text_bow)  (10500, 9023)
BOW on Title (cv_titles_bow)  (10500, 1190)
TFIDF on Essay (cv_text_tfidf) (10500, 9023)
TFIDF on Title (cv_title_tfidf) (10500, 1190)
Avg W2V on Essay (avg_w2v_cv_essay_vectors) 10500 300
Avg W2V on Title (avg_w2v_cv_title_vectors) 10500 300
TFIDF W2V on Essay (tfidf_w2v_cv_essay_vectors) 10500 300
TFIDF W2V on Title (tfidf_w2v_cv_title_vectors) 10500 300
=========================================================================
==
TEST Data
=========================================================================
==
BOW on Essay (test_text_bow)  (15000, 9023)
BOW on title (test_titles_bow)  (15000, 1190)
TFIDF on Essay (test_text_tfidf) (15000, 9023)
TFIDF on Title (test_title_tfidf) (15000, 1190)
Avg W2V on Essay (avg_w2v_test_essay_vectors) 15000 300
Avg W2V on Title (avg_w2v_test_title_vectors) 15000 300
TFIDF W2V on Essay (tfidf_w2v_test_essay_vectors) 15000 300
TFIDF W2V on Title (tfidf_w2v_test_title_vectors) 15000 300
```

## 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 [57]:

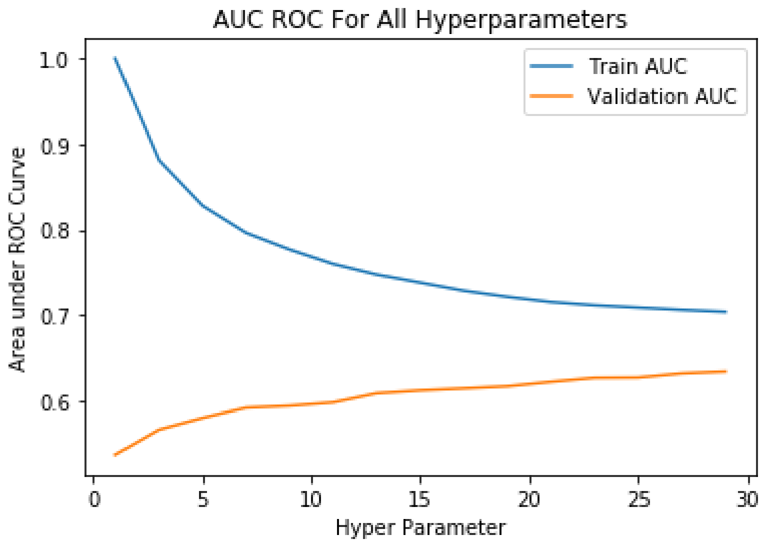
```
# please write all the code with proper documentation, and proper titles fo
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in de

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

# Preparing the data matrix with the given features
from scipy.sparse import hstack
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn import metrics

# method to plot the graph for Hyperparameter Vs AUC
def plot_hyper_vs_auc(train_auc_score, cv_auc_score, hyper_parameters):

    plt.plot(hyper_parameters ,train_auc_score, label= "Train AUC")
    plt.plot(hyper_parameters,cv_auc_score, label="Validation AUC")
    plt.title("AUC ROC For All Hyperparameters")
    plt.xlabel("Hyper Parameter")
    plt.ylabel("Area under ROC Curve")
    plt.legend()
    plt.show()

# method to get auc score based on prediction
def get_auc_score (train_data, train_val, cv_data, cv_val):

    auc_score = []
    for i in range(1,50,2):
        # instantiate learning model (k = 30)
        knn = KNeighborsClassifier(n_neighbors=i)

        # fitting the model on crossvalidation train
        knn.fit(train_data, train_val)

        # predict the response on the given data (train/cv)
        pred = knn.predict_proba(cv_data)
        postv_class_test_prob = [item[1] for item in pred]

        # Appending the score to a list
        auc_score.append(roc_auc_score(cv_val, postv_class_test_prob))

    return auc_score

def plot_roc (fpr_test,tpr_test,fpr_train,tpr_train):

    plt.plot(fpr_test,tpr_test, label="ROC Curve for Test Data")
    plt.plot(fpr_train,tpr_train, label="ROC Curve for Train Data")
    plt.title("Area under ROC Curve")
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.legend()
    plt.show()
```

## 2.4.1 Applying KNN brute force on BOW, SET 1

In [218]:

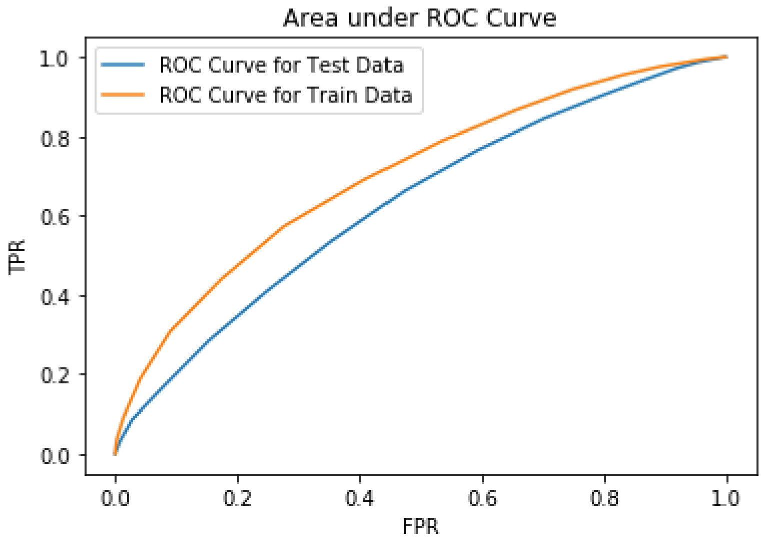
```
# Applying KNN on Set 1: categorical, numerical features + project_title(BC

set1_train_data = hstack((categories_one_hot_train,sub_categories_one_hot_t
print ("Shape of BOW Train Dataset ",set1_train_data.shape)

set1_cv_data = hstack((categories_one_hot_cv,sub_categories_one_hot_cv,stat
print ("Shape of BOW CV Dataset ",set1_cv_data.shape)

set1_test_data = hstack((categories_one_hot_test,sub_categories_one_hot_tes
print ("Shape of BOW Test Dataset ",set1_test_data.shape)

# get CV AUC Score
cv_auc_score = get_auc_score (set1_train_data,project_data_tr["project_is_a

# Get Train AUC Score
train_auc_score = get_auc_score (set1_train_data,project_data_tr["project_i

hyper_parameters = list(range (1,30,2))

# Plot Graph for Hyperparameter VS AUC
plot_hyper_vs_auc (train_auc_score,cv_auc_score,hyper_parameters)
```

```
Shape of BOW Train Dataset  (24500, 10322)
Shape of BOW CV Dataset  (10500, 10322)
Shape of BOW Test Dataset  (15000, 10322)
```



In [223]:

```
# training the model with best K
knn = KNeighborsClassifier(n_neighbors=30)
knn.fit(set1_train_data, project_data_tr["project_is_approved"])

# predict probabilities for test data
pred_prob = knn.predict_proba(set1_test_data)
postv_class_test_prob = [item[1] for item in pred_prob]

fpr_test, tpr_test, thresholds = metrics.roc_curve(project_data_test["proje

# predict probabilities for train data
pred_prob = knn.predict_proba(set1_train_data)
postv_class_train_prob = [item[1] for item in pred_prob]

fpr_train, tpr_train, thresholds = metrics.roc_curve(project_data_tr["proje

# Plot ROC Curve for Train and Test data
plot_roc (fpr_test,tpr_test,fpr_train,tpr_train)
```



In [221]:

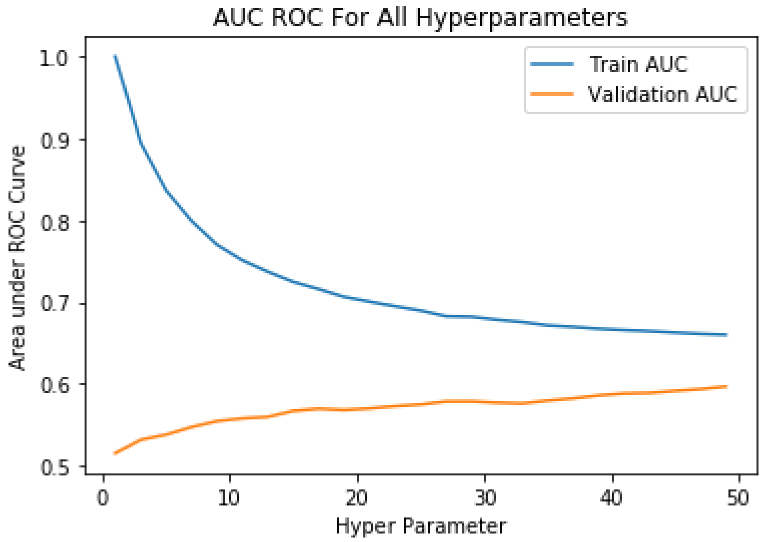
```
# Confusion matrix for testData
from sklearn.metrics import confusion_matrix

y_pred = knn.predict(set1_test_data)

conf_mtrx = confusion_matrix(project_data_test["project_is_approved"], y_pr

print (conf_mtrx)
```

```
[[   138  2301]
 [   245 12316]]
```

## 2.4.2 Applying KNN brute force on TFIDF, SET 2

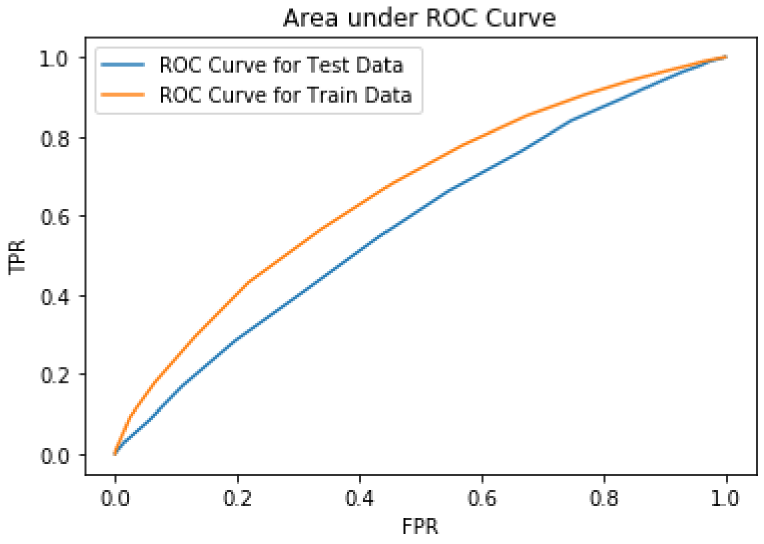
```
In [229]: # Please write all the code with proper documentation
          # Applying KNN on Set 2: categorical, numerical features + project_title(TF

          set2_train_data = hstack((categories_one_hot_train,sub_categories_one_hot_t
          print ("Shape of TFIDF Train Dataset ",set2_train_data.shape)

          set2_cv_data = hstack((categories_one_hot_cv,sub_categories_one_hot_cv,stat
          print ("Shape of TFIDF CV Dataset ",set2_cv_data.shape)

          set2_test_data = hstack((categories_one_hot_test,sub_categories_one_hot_tes
          print ("Shape of TFIDF Test Dataset ",set2_test_data.shape)

          # get CV AUC Score
          cv_auc_score = get_auc_score (set2_train_data,project_data_tr["project_is_a

          # Get Train AUC Score
          train_auc_score = get_auc_score (set2_train_data,project_data_tr["project_i

          hyper_parameters = list(range (1,50,2))

          # Plot Graph for Hyperparameter VS AUC
          plot_hyper_vs_auc (train_auc_score,cv_auc_score,hyper_parameters)
```

```
Shape of TFIDF Train Dataset  (24500, 10322)
Shape of TFIDF CV Dataset  (10500, 10322)
Shape of TFIDF Test Dataset  (15000, 10322)
```



In [230]:

```
# training the model with best K
knn = KNeighborsClassifier(n_neighbors=49)
knn.fit(set2_train_data, project_data_tr["project_is_approved"])

# predict probabilities for test data
pred_prob = knn.predict_proba(set2_test_data)
postv_class_test_prob = [item[1] for item in pred_prob]

fpr_test, tpr_test, thresholds = metrics.roc_curve(project_data_test["proje

# predict probabilities for train data
pred_prob = knn.predict_proba(set2_train_data)
postv_class_train_prob = [item[1] for item in pred_prob]

fpr_train, tpr_train, thresholds = metrics.roc_curve(project_data_tr["proje

# Plot ROC Curve for Train and Test data
plot_roc (fpr_test,tpr_test,fpr_train,tpr_train)
```



In [ ]:

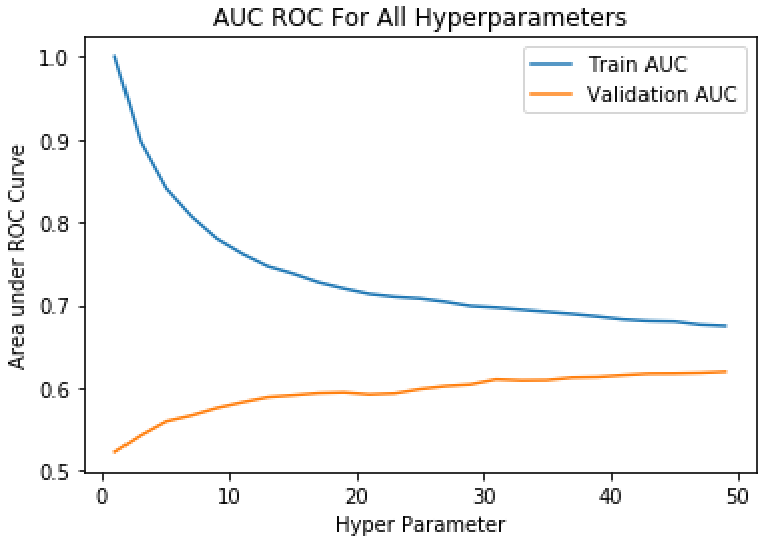
```
# Confusion matrix for testData

y_pred = knn.predict(set2_test_data)

conf_mtrx = confusion_matrix(project_data_test["project_is_approved"], y_pr

print (conf_mtrx)
```

## 2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [232]:

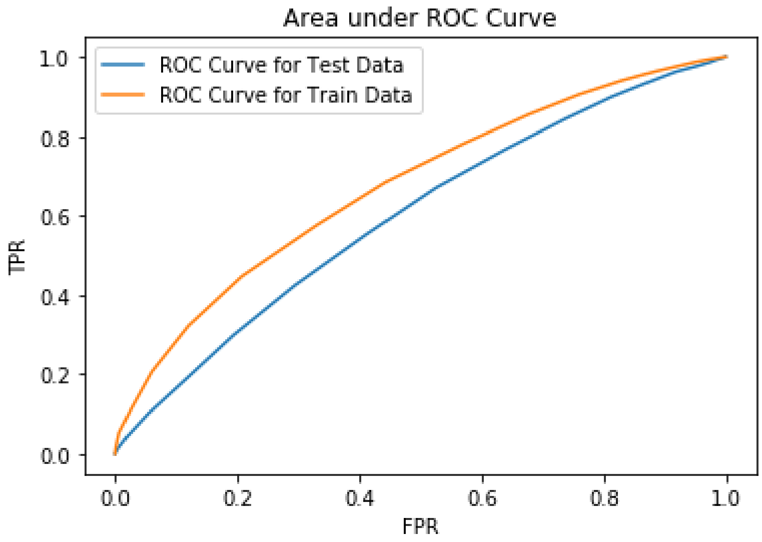
```
# Applying KNN on Set 3: categorical, numerical features + project_title(AV

set3_train_data = hstack((categories_one_hot_train,sub_categories_one_hot_t
print ("Shape of TFIDF Train Dataset ",set3_train_data.shape)

set3_cv_data = hstack((categories_one_hot_cv,sub_categories_one_hot_cv,stat
print ("Shape of TFIDF CV Dataset ",set3_cv_data.shape)

set3_test_data = hstack((categories_one_hot_test,sub_categories_one_hot_tes
print ("Shape of TFIDF Test Dataset ",set3_test_data.shape)

# get CV AUC Score
cv_auc_score = get_auc_score (set3_train_data,project_data_tr["project_is_a

# Get Train AUC Score
train_auc_score = get_auc_score (set3_train_data,project_data_tr["project_i

hyper_parameters = list(range (1,50,2))

# Plot Graph for Hyperparameter VS AUC
plot_hyper_vs_auc (train_auc_score,cv_auc_score,hyper_parameters)
```

```
Shape of TFIDF Train Dataset  (24500, 700)
Shape of TFIDF CV Dataset  (10500, 700)
Shape of TFIDF Test Dataset  (15000, 700)
```



In [233]:

```
# training the model with best K
knn = KNeighborsClassifier(n_neighbors=49)
knn.fit(set3_train_data, project_data_tr["project_is_approved"])

# predict probabilities for test data
pred_prob = knn.predict_proba(set3_test_data)
postv_class_test_prob = [item[1] for item in pred_prob]

fpr_test, tpr_test, thresholds = metrics.roc_curve(project_data_test["proje

# predict probabilities for train data
pred_prob = knn.predict_proba(set3_train_data)
postv_class_train_prob = [item[1] for item in pred_prob]

fpr_train, tpr_train, thresholds = metrics.roc_curve(project_data_tr["proje

# Plot ROC Curve for Train and Test data
plot_roc (fpr_test,tpr_test,fpr_train,tpr_train)
```



In [234]:

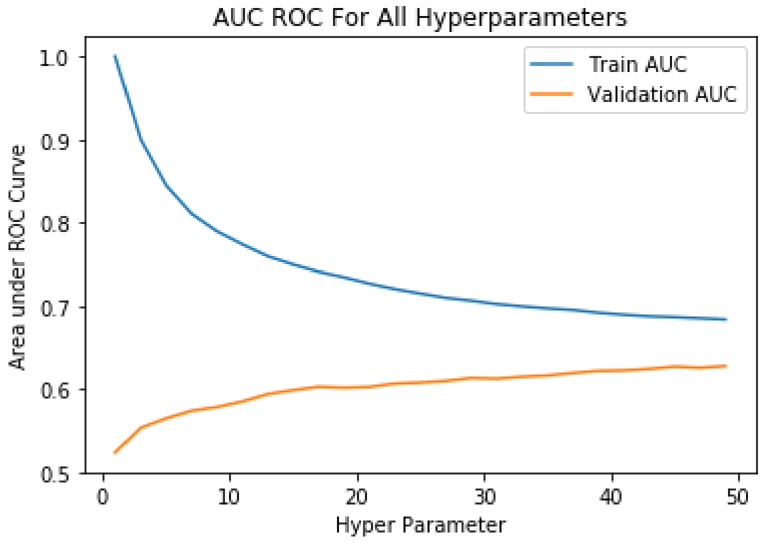
```
y_pred = knn.predict(set3_test_data)

conf_mtrx = confusion_matrix(project_data_test["project_is_approved"], y_pr

print (conf_mtrx)
```

```
[[    0  2439]
 [    1 12560]]
```

## 2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

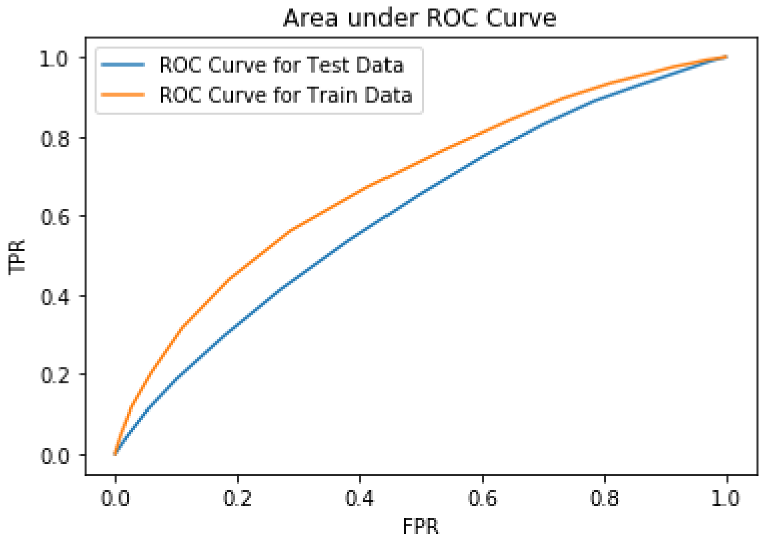
In [235]:
```
ying KNN on Set 4: categorical, numerical features + project_title(TFIDF W2V

rain_data = hstack((categories_one_hot_train,sub_categories_one_hot_train,st
("Shape of TFIDF Train Dataset ",set4_train_data.shape)

v_data = hstack((categories_one_hot_cv,sub_categories_one_hot_cv,states_one_
("Shape of TFIDF CV Dataset ",set4_cv_data.shape)

est_data = hstack((categories_one_hot_test,sub_categories_one_hot_test,state
("Shape of TFIDF Test Dataset ",set4_test_data.shape)

CV AUC Score
_score = get_auc_score (set4_train_data,project_data_tr["project_is_approved

Train AUC Score
auc_score = get_auc_score (set4_train_data,project_data_tr["project_is_appro

parameters = list(range (1,50,2))

 Graph for Hyperparameter VS AUC
yper_vs_auc (train_auc_score,cv_auc_score,hyper_parameters)
```

```
Shape of TFIDF Train Dataset  (24500, 700)
Shape of TFIDF CV Dataset  (10500, 700)
Shape of TFIDF Test Dataset  (15000, 700)
```



In [236]:

```
# training the model with best K
knn = KNeighborsClassifier(n_neighbors=49)
knn.fit(set4_train_data, project_data_tr["project_is_approved"])

# predict probabilities for test data
pred_prob = knn.predict_proba(set4_test_data)
postv_class_test_prob = [item[1] for item in pred_prob]

fpr_test, tpr_test, thresholds = metrics.roc_curve(project_data_test["proje

# predict probabilities for train data
pred_prob = knn.predict_proba(set4_train_data)
postv_class_train_prob = [item[1] for item in pred_prob]

fpr_train, tpr_train, thresholds = metrics.roc_curve(project_data_tr["proje

# Plot ROC Curve for Train and Test data
plot_roc (fpr_test,tpr_test,fpr_train,tpr_train)
```



In [237]:

```
y_pred = knn.predict(set4_test_data)

conf_mtrx = confusion_matrix(project_data_test["project_is_approved"], y_pr

print (conf_mtrx)
```

```
[[    0  2439]
 [    0 12561]]
```

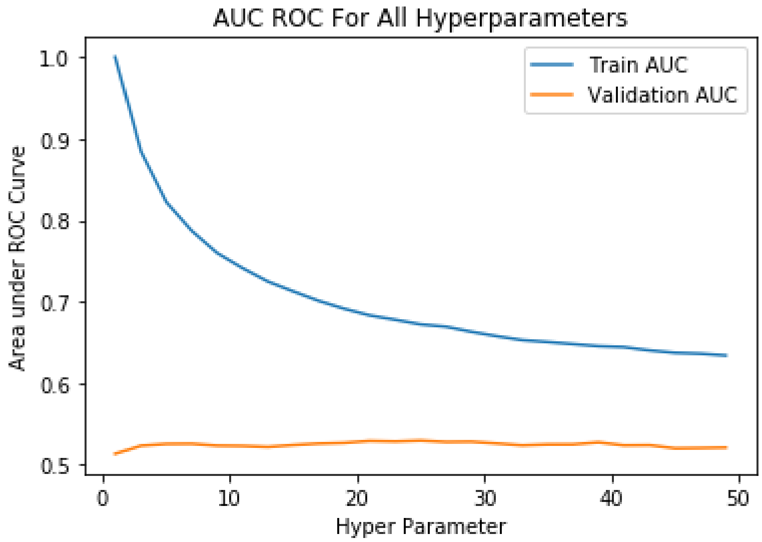
## 2.5 Feature selection with `SelectKBest`

In [55]:
```
from sklearn.datasets import load_digits
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.preprocessing import minmax_scale

# Noramalizing numerical feature since standardization won't supported by s

# Train Data
price_normalized_tr_data = minmax_scale(project_data_tr['price']).reshape(-
# price_normalized_data = price_normalized_data.reshape(-1,1)

# CV Data
price_normalized_cv_data = minmax_scale(project_data_cv['price']).reshape(-


# Test Data
price_normalized_test_data = minmax_scale(project_data_test['price']).resha

# Preparing the Data with normalized price
set2_train_data_norm = hstack((categories_one_hot_train,sub_categories_one_
set2_cv_data_norm = hstack((categories_one_hot_cv,sub_categories_one_hot_cv
set2_test_data_norm = hstack((categories_one_hot_test,sub_categories_one_hc

selectKBest_train = SelectKBest(chi2, k=2000).fit(set2_train_data_norm,proj

set2_train_data_kbest = selectKBest_train.transform(set2_train_data_norm)
print ("Shape of TFIDF Train Dataset ",set2_train_data_kbest.shape)

set2_cv_data_kbest = selectKBest_train.transform(set2_cv_data_norm)
print ("Shape of TFIDF CV Dataset ",set2_cv_data_kbest.shape)

set2_test_data_kbest = selectKBest_train.transform(set2_test_data_norm)
print ("Shape of TFIDF Test Dataset ",set2_test_data_kbest.shape)
```

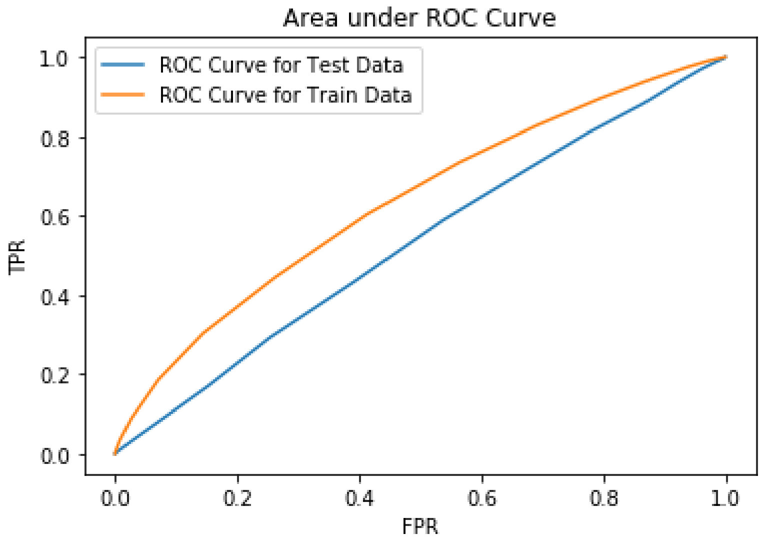
```
Shape of TFIDF Train Dataset  (24500, 2000)
Shape of TFIDF CV Dataset  (10500, 2000)
Shape of TFIDF Test Dataset  (15000, 2000)
```

In [58]:

```
# Applying KNN on Set 2 with selectKBest: categorical, numerical features +

# get CV AUC Score
cv_auc_score = get_auc_score (set2_train_data_kbest,project_data_tr["projec

# Get Train AUC Score
train_auc_score = get_auc_score (set2_train_data_kbest,project_data_tr["pro

hyper_parameters = list(range (1,50,2))

# Plot Graph for Hyperparameter VS AUC
plot_hyper_vs_auc (train_auc_score,cv_auc_score,hyper_parameters)
```



In [59]:
```
# training the model with best K
knn = KNeighborsClassifier(n_neighbors=49)
knn.fit(set2_train_data_kbest, project_data_tr["project_is_approved"])

# predict probabilities for test data
pred_prob = knn.predict_proba(set2_test_data_kbest)
postv_class_test_prob = [item[1] for item in pred_prob]

fpr_test, tpr_test, thresholds = metrics.roc_curve(project_data_test["proje

# predict probabilities for train data
pred_prob = knn.predict_proba(set2_train_data_kbest)
postv_class_train_prob = [item[1] for item in pred_prob]

fpr_train, tpr_train, thresholds = metrics.roc_curve(project_data_tr["proje

# Plot ROC Curve for Train and Test data
plot_roc (fpr_test,tpr_test,fpr_train,tpr_train)
```



In [60]:
```
# confusion matrix
y_pred = knn.predict(set2_test_data_kbest)

conf_mtrx = confusion_matrix(project_data_test["project_is_approved"], y_pr

print (conf_mtrx)
```

```
[[    0  2439]
 [    0 12561]]
```

# 3. Conclusions

In [279]:

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

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]

x.add_row(["BOW", "Brute", 49, 0.57])
x.add_row(["TFIDF", "Brute", 49, 0.57])
x.add_row(["W2V", "Brute", 49, 0.59])
x.add_row(["TFIDFW2V", "Brute", 49, 0.61])


print(x)
```

```
+------------+-------+-----------------+------+
| Vectorizer | Model | Hyper Parameter | AUC  |
+------------+-------+-----------------+------+
|    BOW     | Brute |        49       | 0.57 |
|   TFIDF    | Brute |        49       | 0.57 |
|    W2V     | Brute |        49       | 0.59 |
|  TFIDFW2V  | Brute |        49       | 0.61 |
+------------+-------+-----------------+------+
```

In [ ]: