DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

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

Feature	Description
`project_essay_3`	Third application essay*
`project_essay_4`	Fourth application essay*
`project_submitted_datetime`	Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245`
`teacher_id`	A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56`
`teacher_prefix`	Teacher's title. One of the following enumerated values: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `Teacher.`
`teacher_number_of_previously_posted_projects`	Number of project applications previously submitted by the same teacher. **Example:** `2`

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

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

```
import 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/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns)]


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


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

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

Out[4]:

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

In [5]:

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

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

Out[5]:

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

1.2 preprocessing of `project_subject_categories`

In [6]:

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

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [7]:

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

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

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

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

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

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

1.3 Text preprocessing

In [8]:

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

In [9]:

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

Out[9]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	projec
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Engin STE/ the F Clas
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	S To

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	projec

In [10]:

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

In [11]:

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

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classroom as well as the STEM journa
ls, which my students really enjoyed.  I would 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 socioeconomic status.  Many of them don't have a
lot of experience in science and engineering and these kits give me the materials to provide these exci
ting opportunities for my students.Each month I try to do several science or STEM/STEAM projects.  I wo
uld use the kits and robot to help guide my science instruction in engaging and meaningful ways.  I can
adapt the kits to my current language arts pacing guide where we already teach some of the material in
the kits like tall tales (Paul Bunyan) or Johnny Appleseed.  The following units will be taught in the
next school year where I will implement these kits: magnets, motion, sink vs. float, robots.  I often g
et to these units and don't know If I am teaching the right way or using the right materials.    The ki
ts will give me additional ideas, strategies, and lessons to prepare my students in science.It is chall
enging to develop high quality science activities.  These kits give me the materials I need to provide
my students with science activities that will go along with the curriculum in my classroom.  Although I
have some things (like magnets) in my classroom, I don't know how to use them effectively.  The kits wi
ll provide me with the right amount of materials and show me how to use them in an appropriate way.
==================================================
I teach high school English to students with learning and behavioral disabilities. My students all vary
in their ability level. However, the ultimate goal is to increase all students literacy levels. This in
cludes their reading, writing, and communication levels.I teach a really dynamic group of students. How
ever, my students face a lot of challenges. My students all live in poverty and in a dangerous neighbor
hood. Despite these challenges, I have students who have the the desire to defeat these challenges. My
students all have learning disabilities and currently all are performing below grade level. My students
are visual learners and will benefit from a classroom that fulfills their preferred learning style.The
materials I am requesting will allow my students to be prepared for the classroom with the necessary su
pplies.  Too often I am challenged with students who come to school unprepared for class due to economi
c challenges.  I want my students to be able to focus on learning and not how they will be able to get
school supplies.  The supplies will last all year.  Students will be able to complete written assignmen
ts and maintain a classroom journal.  The chart paper will be used to make learning more visual in clas
s and to create posters to aid students in their learning.  The students have access to a classroom pri
nter.  The toner will be used to print student work that is completed on the classroom Chromebooks.I wa
nt to try and remove all barriers for the students learning and create opportunities for learning. One
of the biggest barriers is the students not having the resources to get pens, paper, and folders. My st
udents will be able to increase their literacy skills because of this project.
==================================================
\"Life moves pretty fast. If you don't stop and look around once in awhile, you could miss it.\"  from
the movie, Ferris Bueller's Day Off.  Think back...what do you remember about your grandparents?  How a
mazing would it be to be able to flip through a book to see a day in their lives?My second graders are
voracious readers! They love to read both fiction and nonfiction books.  Their favorite characters incl
ude Pete the Cat, Fly Guy, Piggie and Elephant, and Mercy Watson. They also love to read about insects,
space and plants. My students are hungry bookworms! My students are eager to learn and read about the w
orld around them. My kids love to be at school and are like little sponges absorbing everything around
them. Their parents work long hours and usually do not see their children. My students are usually care
d for by their grandparents or a family friend. Most of my students do not have someone who speaks Engl
ish at home. Thus it is difficult for my students to acquire language.Now think forward... wouldn't it
mean a lot to your kids, nieces or nephews or grandchildren, to be able to see a day in your life today
30 years from now? Memories are so precious to us and being able to share these memories with future ge
nerations will be a rewarding experience.  As part of our social studies curriculum, students will be l
earning about changes over time.  Students will be studying photos to learn about how their community h
as changed over time.  In particular, we will look at photos to study how the land, buildings, clothing
, and schools have changed over time.  As a culminating activity, my students will capture a slice of t
heir history and preserve it through scrap booking. Key important events in their young lives will be d
ocumented with the date, location, and names.  Students will be using photos from home and from school
```

ocumented with the date, location, and names. Students will be using photos from home and from school to create their second grade memories. Their scrap books will preserve their unique stories for futur e generations to enjoy.Your donation to this project will provide my second graders with an opportunity to learn about social studies in a fun and creative manner. Through their scrapbooks, children will sh are their story with others and have a historical document for the rest of their lives.
==========
\"A person's a person, no matter how small.\" (Dr.Seuss) I teach the smallest students with the biggest enthusiasm for learning. My students learn in many different ways using all of our senses and multiple intelligences. I use a wide range of techniques to help all my students succeed. \r\nStudents in my cla ss come from a variety of different backgrounds which makes for wonderful sharing of experiences and cu ltures, including Native Americans.\r\nOur school is a caring community of successful learners which ca n be seen through collaborative student project based learning in and out of the classroom. Kindergarte ners in my class love to work with hands-on materials and have many different opportunities to practice a skill before it is mastered. Having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curriculum.Montana is the perfect place to learn about agriculture and nutri tion. My students love to role play in our pretend kitchen in the early childhood classroom. I have had several kids ask me, \"Can we try cooking with REAL food?\" I will take their idea and create \"Common Core Cooking Lessons\" where we learn important math and writing concepts while cooking delicious healt hy food for snack time. My students will have a grounded appreciation for the work that went into makin g the food and knowledge of where the ingredients came from as well as how it's healthy for their bodie s. This project would expand our learning of nutrition and agricultural cooking recipes by having us pe el our own apples to make homemade applesauce, make our own bread, and mix up healthy plants from our c lassroom garden in the spring. We will also create our own cookbooks to be printed and shared with fami lies. \r\nStudents will gain math and literature skills as well as a life long enjoyment for healthy co oking.nannan
==========
My classroom consists of twenty-two amazing sixth graders from different cultures and backgrounds. They are a social bunch who enjoy working in partners and working with groups. They are hard-working and eag er to head to middle school next year. My job is to get them ready to make this transition and make it as smooth as possible. In order to do this, my students need to come to school every day and feel safe and ready to learn. Because they are getting ready to head to middle school, I give them lots of choice - choice on where to sit and work, the order to complete assignments, choice of projects, etc. Part of the students feeling safe is the ability for them to come into a welcoming, encouraging environment. My room is colorful and the atmosphere is casual. I want them to take ownership of the classroom because w e ALL share it together. Because my time with them is limited, I want to ensure they get the most of th is time and enjoy it to the best of their abilities.Currently, we have twenty-two desks of differing si zes, yet the desks are similar to the ones the students will use in middle school. We also have a kidne y table with crates for seating. I allow my students to choose their own spots while they are working i ndependently or in groups. More often than not, most of them move out of their desks and onto the crate s. Believe it or not, this has proven to be more successful than making them stay at their desks! It is because of this that I am looking toward the “Flexible Seating” option for my classroom.\r\n The studen ts look forward to their work time so they can move around the room. I would like to get rid of the con stricting desks and move toward more “fun” seating options. I am requesting various seating so my stude nts have more options to sit. Currently, I have a stool and a papasan chair I inherited from the previo us sixth-grade teacher as well as five milk crate seats I made, but I would like to give them more opti ons and reduce the competition for the “good seats”. I am also requesting two rugs as not only more sea ting options but to make the classroom more welcoming and appealing. In order for my students to be abl e to write and complete work without desks, I am requesting a class set of clipboards. Finally, due to curriculum that requires groups to work together, I am requesting tables that we can fold up when we ar e not using them to leave more room for our flexible seating options.\r\nI know that with more seating options, they will be that much more excited about coming to school! Thank you for your support in maki ng my classroom one students will remember forever!nannan
==========

In [12]:

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

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

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

In [13]:

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

```
\"A person is a person, no matter how small.\" (Dr.Seuss) I teach the smallest students with the bigges
t enthusiasm for learning. My students learn in many different ways using all of our senses and multipl
e intelligences. I use a wide range of techniques to help all my students succeed. \r\nStudents in my c
lass come from a variety of different backgrounds which makes for wonderful sharing of experiences and
cultures, including Native Americans.\r\nOur school is a caring community of successful learners which
can be seen through collaborative student project based learning in and out of the classroom. Kindergar
teners in my class love to work with hands-on materials and have many different opportunities to practi
ce a skill before it is mastered. Having the social skills to work cooperatively with friends is a cruc
ial aspect of the kindergarten curriculum.Montana is the perfect place to learn about agriculture and n
utrition. My students love to role play in our pretend kitchen in the early childhood classroom. I have
had several kids ask me, \"Can we try cooking with REAL food?\" I will take their idea and create \"Com
mon Core Cooking Lessons\" where we learn important math and writing concepts while cooking delicious h
ealthy food for snack time. My students will have a grounded appreciation for the work that went into m
aking the food and knowledge of where the ingredients came from as well as how it is healthy for their
bodies. This project would expand our learning of nutrition and agricultural cooking recipes by having
us peel our own apples to make homemade applesauce, make our own bread, and mix up healthy plants from
our classroom garden in the spring. We will also create our own cookbooks to be printed and shared with
families. \r\nStudents will gain math and literature skills as well as a life long enjoyment for health
y cooking.nannan
==================================================
```

In [14]:

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

```
 A person is a person, no matter how small.  (Dr.Seuss) I teach the smallest students with the biggest
enthusiasm for learning. My students learn in many different ways using all of our senses and multiple
intelligences. I use a wide range of techniques to help all my students succeed.  Students in my class
come from a variety of different backgrounds which makes for wonderful sharing of experiences and cultu
res, including Native Americans.  Our school is a caring community of successful learners which can be
seen through collaborative student project based learning in and out of the classroom. Kindergarteners
in my class love to work with hands-on materials and have many different opportunities to practice a sk
ill before it is mastered. Having the social skills to work cooperatively with friends is a crucial asp
ect of the kindergarten curriculum.Montana is the perfect place to learn about agriculture and nutritio
n. My students love to role play in our pretend kitchen in the early childhood classroom. I have had se
veral kids ask me,  Can we try cooking with REAL food?  I will take their idea and create  Common Core
Cooking Lessons  where we learn important math and writing concepts while cooking delicious healthy foo
d for snack time. My students will have a grounded appreciation for the work that went into making the
food and knowledge of where the ingredients came from as well as how it is healthy for their bodies. Th
is project would expand our learning of nutrition and agricultural cooking recipes by having us peel ou
r own apples to make homemade applesauce, make our own bread, and mix up healthy plants from our classr
oom garden in the spring. We will also create our own cookbooks to be printed and shared with families.
Students will gain math and literature skills as well as a life long enjoyment for healthy cooking.nann
an
```

In [15]:

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

```
 A person is a person no matter how small Dr Seuss I teach the smallest students with the biggest enthu
siasm for learning My students learn in many different ways using all of our senses and multiple intell
igences I use a wide range of techniques to help all my students succeed Students in my class come from
a variety of different backgrounds which makes for wonderful sharing of experiences and cultures includ
ing Native Americans Our school is a caring community of successful learners which can be seen through
collaborative student project based learning in and out of the classroom Kindergarteners in my class lo
ve to work with hands on materials and have many different opportunities to practice a skill before it
is mastered Having the social skills to work cooperatively with friends is a crucial aspect of the kind
ergarten curriculum Montana is the perfect place to learn about agriculture and nutrition My students l
ove to role play in our pretend kitchen in the early childhood classroom I have had several kids ask me
```

```
Can we try cooking with REAL food I will take their idea and create Common Core Cooking Lessons where w
e learn important math and writing concepts while cooking delicious healthy food for snack time My stud
ents will have a grounded appreciation for the work that went into making the food and knowledge of whe
re the ingredients came from as well as how it is healthy for their bodies This project would expand ou
r learning of nutrition and agricultural cooking recipes by having us peel our own apples to make homem
ade applesauce make our own bread and mix up healthy plants from our classroom garden in the spring We
will also create our own cookbooks to be printed and shared with families Students will gain math and l
iterature skills as well as a life long enjoyment for healthy cooking nannan
```

In [16]:

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

In [17]:

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

```
100%|██████████| 109248/109248 [02:12<00
:00, 825.10it/s]
```

In [18]:

```
# after preprocesing
preprocessed_essays[20000]
```

Out[18]:

```
'person person no matter small dr seuss teach smallest students biggest enthusiasm learning students le
arn many different ways using senses multiple intelligences use wide range techniques help students suc
ceed students class come variety different backgrounds makes wonderful sharing experiences cultures inc
luding native americans school caring community successful learners seen collaborative student project
```

```
based learning classroom kindergarteners class love work hands materials many different opportunities p
ractice skill mastered social skills work cooperatively friends crucial aspect kindergarten curriculum
montana perfect place learn agriculture nutrition students love role play pretend kitchen early childho
od classroom several kids ask try cooking real food take idea create common core cooking lessons learn
important math writing concepts cooking delicious healthy food snack time students grounded appreciatio
n work went making food knowledge ingredients came well healthy bodies project would expand learning nu
trition agricultural cooking recipes us peel apples make homemade applesauce make bread mix healthy pla
nts classroom garden spring also create cookbooks printed shared families students gain math literature
skills well life long enjoyment healthy cooking nannan'
```

In [19]:

```
# Updating dataframe for clean project title and remove old project title
project_data['clean_essay'] = preprocessed_essays
project_data.drop(['essay'], axis=1, inplace=True)
project_data.head(2)
```

Out[19]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	projec
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Engin STE/ the F Clas
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	S Tc

1.4 Preprocessing of `project_title`

In [20]:

```
# similarly you can preprocess the titles also
# Combining all the above stundents
from tqdm import tqdm
preprocessed_title = []
# 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_title.append(sent.lower().strip())
```

```
100%|██████████| 109248/109248 [00:05<00:0
0, 18628.19it/s]
```

In [21]:

```
# after preprocesing
preprocessed_title[20000]
```

Out[21]:

```
'health nutritional cooking kindergarten'
```

In [22]:

```
# Updating dataframe for clean project title and remove old project title
project_data['clean_project_title'] = preprocessed_title
```

```
project_data.drop(['project_title'], axis=1, inplace=True)
project_data.head(2)
```

Out[22]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	projec
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	fortun to use
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Imagi You're

1.5 Preparing data for models

In [23]:

```
project_data.columns
```

Out[23]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'clean_essay',
       'clean_project_title'],
      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/

In [0]:

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

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

In [0]:

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular', 'Civics_
Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger', 'SocialSciences', 'Perf
ormingArts', 'CharacterEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_Geogr
aphy', 'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness', 'EnvironmentalScience', 'VisualA
rts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Liter
acy']
Shape of matrix after one hot encodig  (109248, 30)
```

In [0]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [0]:

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

```
Shape of matrix after one hot encodig  (109248, 16623)
```

In [0]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
```

1.5.2.2 TFIDF vectorizer

In [0]:

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

```
Shape of matrix after one hot encodig  (109248, 16623)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [0]:

```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
```

```
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

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

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

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

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

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

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

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


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-an
d-load-variables-in-python/

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


'''
```

Out[0]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef loadGloveModel(
gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    mod
el = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n        word = splitLine[0]\n
embedding = np.array([float(val) for val in splitLine[1:]])\n        model[word] = embedding\n    print
("Done.",len(model)," words loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.txt\')
\n\n# ============================\nOutput:\n    \nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\
nDone. 1917495  words loaded!\n\n# ============================\n\nwords = []\nfor i in preproced_texts
:\n    words.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\npr
int("all the words in the coupus", len(words))\nwords = set(words)\nprint("the unique words in the coup
us", len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",       len(inter_words),"(",np.round(len(inter_word
s)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove = set(model.keys())\nfor i in words:\n
if i in words_glove:\n        words_courpus[i] = model[i]\nprint("word 2 vec length", len(words_courpus
))\n\n\n# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to
-save-and-load-variables-in-python/\n\nimport pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n    p
ickle.dump(words_courpus, f)\n\n\n'
```

In [0]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-an
d-load-variables-in-python/
```

```
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [0]:

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

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

```
100%|| 109248/109248 [00:59<00:00, 1830.39it/s]
```

```
109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [0]:

```
# 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 [0]:

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

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

```
100%|| 109248/109248 [07:23<00:00, 246.23it/s]
```

```
109248
300
```

In [0]:

```
# Similarly you can vectorize for title also
```

1.5.3 Vectorizing Numerical features

In [0]:

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

In [0]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73 5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

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

In [0]:

```
price_standardized
```

Out[0]:

```
array([[4.63560392e-03, 1.36200635e-03, 2.10346002e-03, ...,
        2.55100471e-03, 1.83960046e-03, 3.51642253e-05]])
```

1.5.4 Merging all the above features

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

In [0]:

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

```
(109248, 9)
(109248, 30)
(109248, 16623)
(109248, 1)
```

In [0]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
```

```
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
X.shape
```

Out[0]:

```
(109248, 16663)
```

Assignment 3: Apply KNN

1. **[Task-1] Apply KNN(brute force version) on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)
2. **Hyper paramter tuning to find best K**
   - Find the best hyper parameter which results in the maximum AUC value
   - Find the best hyper paramter using k-fold cross validation (or) simple cross validation data
   - Use gridsearch-cv or randomsearch-cv or write your own for loops to do this task
3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, as shown in the figure
   - Once you find the best hyper parameter, you need to train your model-M using the best hyper-param. Now, find the AUC on test data and plot the ROC curve on both train and test using model-M.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points
4. **[Task-2]**
   - Select top 2000 features from feature Set 2 using `SelectKBest` and then apply KNN on top of these features

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

   - Repeat the steps 2 and 3 on the data matrix after feature selection
5. **Conclusion**
   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

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

2. K Nearest Neighbor

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

In [27]:

```
# Combine the train.csv and resource.csv
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in-one-step
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
# join two dataframes in python:
project_data = pd.merge(project_data, price_data, on='id', how='left')

# Take 50k dataset ----> Tried but memory error
# Take 40k dataset
from sklearn.model_selection import train_test_split

# remove unnecessary column: https://cmdlinetips.com/2018/04/how-to-drop-one-or-more-columns-in-pandas-dataframe/
project_data = project_data.drop(['Unnamed: 0','id','teacher_id','Date'], axis=1)
# https://www.geeksforgeeks.org/python-pandas-dataframe-sample/
project_data = project_data.sample(n=50000)
project_data = project_data[pd.notnull(project_data['teacher_prefix'])]
project_data.shape
```

Out[27]:

```
(49998, 16)
```

In [28]:

```
project_data.head()
```

Out[28]:

	teacher_prefix	school_state	project_grade_category	project_essay_1	project_essay_2	project_essay_3	project_essay_4	proj
37015	Ms.	IA	Grades 3-5	I have the pleasure of working with a class of...	One of my goals as an educator is to make sure...	NaN	NaN	N
62924	Mrs.	FL	Grades 3-5	The students in my classroom are in grades 3-5...	On a daily basis, my students sit in the tradi...	NaN	NaN	
32460	Ms.	WI	Grades 3-5	As a teacher in a Title I school, my students ...	Having many different options for alternative ...	NaN	NaN	My
77686	Ms.	CA	Grades PreK-2	Most students I teach do not have more than tw...	My students are on the cusp of falling in love...	NaN	NaN	
16920	Mrs.	MO	Grades PreK-2	My students live in a high poverty neighborhoo...	Donations to this project will help my student...	NaN	NaN	My

In [29]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
```

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

# Split train and test with both 50%
tr_X, ts_X, tr_y, ts_y, = train_test_split(project_data, project_data['project_is_approved'], test_size
=0.33, random_state=1, stratify=project_data['project_is_approved'].values)
tr_X = tr_X.reset_index(drop=True)
ts_X = ts_X.reset_index(drop=True)

# # Split train data further with 70% train data and 30% cv data
tr_X, cv_X, tr_y, cv_y = train_test_split(tr_X, tr_y, test_size=0.33, random_state=1, stratify=tr_y)
tr_X = tr_X.reset_index(drop=True)
ts_X = ts_X.reset_index(drop=True)
cv = cv_X.reset_index(drop=True)
tr_X.drop(['project_is_approved'], axis=1, inplace=True)
ts_X.drop(['project_is_approved'], axis=1, inplace=True)
cv_X.drop(['project_is_approved'], axis=1, inplace=True)

print('Shape of train data:', tr_X.shape)
print('Shape of test data:', ts_X.shape)
print('Shape of CV data', cv_X.shape)
```

```
Shape of train data: (22443, 15)
Shape of test data: (16500, 15)
Shape of CV data (11055, 15)
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [27]:

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

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

# For Numerical with train data
### 1) quantity

# We are going to represent the quantity, as numerical values within the range 0-1
# normalization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Standa
rdScaler.html

# quantity_normalized = standardScalar.fit(project_data['quantity'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73
5.5 ].
# Reshape your data either using array.reshape(-1, 1)
from sklearn.preprocessing import StandardScaler

quantity_scalar = StandardScaler()
quantity_scalar.fit(tr_X['quantity'].values.reshape(-1,1)) # finding the mean and standard deviation of
this data
print(f"for quantity -> Mean : {quantity_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scala
r.var_[0])}")

# Now standardize the data with above mean and variance.
quantity_normalized = quantity_scalar.transform(tr_X['quantity'].values.reshape(-1, 1))

quantity_normalized.shape

### 2) price

# the cost feature is already in numerical values,  we are going to represent the money, as numerical v
alues within the range 0-1
```

```
# normalization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

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

# Now standardize the data with above mean and variance.
price_normalized = price_scalar.transform(tr_X['price'].values.reshape(-1, 1))

price_normalized.shape

### 3) For teacher_number_of_previously_projects

# We are going to represent the teacher_number_of_previously_posted_projects, as numerical values within the range 0-1
# normalization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

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

# Now standardize the data with above mean and variance.
teacher_number_of_previously_posted_projects_normalized = teacher_number_of_previously_posted_projects_scalar.transform(tr_X['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
```

```
for quantity -> Mean : 16.853234717519157, Standard deviation : 26.183192140503532
for price: Mean -> 297.4918321154874, Standard deviation : 350.29882668094234
for teacher_number_of_previously_posted_projects -> Mean : 10.79544644448405, Standard deviation : 26.393199080451556
```

In [28]:

```
print('Shape of quantity:', quantity_normalized.shape)
print('Shape of price:', price_normalized.shape)
print('Shape of teacher_number_of_previously_posted_projects:', teacher_number_of_previously_posted_projects_normalized.shape)
```

```
Shape of quantity: (22444, 1)
Shape of price: (22444, 1)
Shape of teacher_number_of_previously_posted_projects: (22444, 1)
```

In [29]:

```
# Transform numerical attributes for test data
ts_price = price_scalar.transform(ts_X['price'].values.reshape(-1,1))
ts_quantity = quantity_scalar.transform(ts_X['quantity'].values.reshape(-1,1))
ts_teacher_number_of_previously_posted_projects = \
teacher_number_of_previously_posted_projects_scalar.transform(ts_X['teacher_number_of_previously_posted_projects'].\
                                                              values.reshape(-1,1))

# tranform nmerical attributes for cv data
cv_price = price_scalar.transform(cv_X['price'].values.reshape(-1,1))
cv_quantity = quantity_scalar.transform(cv_X['quantity'].values.reshape(-1,1))
cv_teacher_number_of_previously_posted_projects = \
teacher_number_of_previously_posted_projects_scalar.transform(cv_X['teacher_number_of_previously_posted_projects'].\
                                                              values.reshape(-1,1))
```

In [30]:

```
print('--------------Test data--------------')
print('Shape of quantity:', ts_quantity.shape)
print('Shape of price:', ts_price.shape)
```

```
print('Shape of teacher_number_of_previously_posted_projects:', ts_teacher_number_of_previously_posted_
projects.shape)

print('--------------CV data--------------')
print('Shape of quantity:', cv_quantity.shape)
print('Shape of price:', cv_price.shape)
print('Shape of teacher_number_of_previously_posted_projects:', cv_teacher_number_of_previously_posted_
projects.shape)
```

```
--------------Test data--------------
Shape of quantity: (16500, 1)
Shape of price: (16500, 1)
Shape of teacher_number_of_previously_posted_projects: (16500, 1)
--------------CV data--------------
Shape of quantity: (11055, 1)
Shape of price: (11055, 1)
Shape of teacher_number_of_previously_posted_projects: (11055, 1)
```

In [30]:

```
# For categorical with train data
# Please do the similar feature encoding with state, teacher_prefix and project_grade_category also
# One hot encoding for school state

### 1) school_state
print('==========================================================================\n')
# Count Vectorize with vocuabulary contains unique code of school state and we are doing boolen BoW
vectorizer_school_state = CountVectorizer(vocabulary=tr_X['school_state'].unique(), lowercase=False, bi
nary=True)
vectorizer_school_state.fit(tr_X['school_state'].values)
print('List of feature in school_state',vectorizer_school_state.get_feature_names())


school_state_one_hot = vectorizer_school_state.transform(tr_X['school_state'].values)
print("\nShape of school_state matrix after one hot encoding ",school_state_one_hot.shape)

### 2) project_subject_categories
print('==========================================================================\n')
vectorizer_categories = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binar
y=True)
vectorizer_categories.fit(tr_X['clean_categories'].values)
print('List of features in project_subject_categories',vectorizer_categories.get_feature_names())


categories_one_hot = vectorizer_categories.transform(tr_X['clean_categories'].values)
print("\nShape of project_subject_categories matrix after one hot encodig ",categories_one_hot.shape)

### 3) project_subject_subcategories
print('==========================================================================\n')
vectorizer_subcategories = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False
, binary=True)
vectorizer_subcategories.fit(tr_X['clean_categories'].values)
print('List of features in project_subject_categories',vectorizer_subcategories.get_feature_names())


subcategories_one_hot = vectorizer_subcategories.transform(tr_X['clean_categories'].values)
print("\nShape of project_subject_subcategories matrix after one hot encodig ",subcategories_one_hot.sh
ape)

### 4) project_grade_category
print('==========================================================================\n')
# One hot encoding for project_grade_category

# Count Vectorize with vocuabulary contains unique code of project_grade_category and we are doing bool
en BoW
vectorizer_grade_category = CountVectorizer(vocabulary=tr_X['project_grade_category'].unique(), lowerca
se=False, binary=True)
vectorizer_grade_category.fit(tr_X['project_grade_category'].values)
print('List of features in project_grade_category',vectorizer_grade_category.get_feature_names())


project_grade_category_one_hot = vectorizer_grade_category.transform(tr_X['project_grade_category'].val
ues)
print("\nShape of project_grade_category matrix after one hot encodig ",project_grade_category_one_hot.
```

```
shape)

### 5) teacher_prefix
print('==========================================================\n')
# One hot encoding for teacher_prefix

# Count Vectorize with vocuabulary contains unique code of teacher_prefix and we are doing boolen BoW
# Since some of the data is filled with nan. So we update the nan to 'None' as a string
tr_X['teacher_prefix'] = tr_X['teacher_prefix'].fillna('None')
vectorizer_teacher_prefix = CountVectorizer(vocabulary=tr_X['teacher_prefix'].unique(), lowercase=False
, binary=True)
vectorizer_teacher_prefix.fit(tr_X['teacher_prefix'].values)
print('List of features in teacher_prefix',vectorizer_teacher_prefix.get_feature_names())


teacher_prefix_one_hot = vectorizer_teacher_prefix.transform(tr_X['teacher_prefix'].values)
print("\nShape of teacher_prefix matrix after one hot encoding ",teacher_prefix_one_hot.shape)
```

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

List of feature in school_state ['IN', 'GA', 'LA', 'NY', 'CA', 'FL', 'WA', 'WV', 'MO', 'CO', 'IL', 'NJ'
, 'AL', 'NM', 'OH', 'OK', 'NC', 'WI', 'PA', 'SC', 'TN', 'MS', 'SD', 'TX', 'AZ', 'NE', 'KY', 'NV', 'MI',
'VA', 'MN', 'IA', 'UT', 'CT', 'OR', 'KS', 'MD', 'ND', 'MA', 'AR', 'RI', 'HI', 'ID', 'NH', 'DC', 'MT', '
DE', 'ME', 'AK', 'VT', 'WY']

Shape of school_state matrix after one hot encoding  (22443, 51)
==========================================================

List of features in project_subject_categories ['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts'
, 'AppliedLearning', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']

Shape of project_subject_categories matrix after one hot encodig  (22443, 9)
==========================================================

List of features in project_subject_categories ['Economics', 'CommunityService', 'FinancialLiteracy', '
ParentInvolvement', 'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducation', '
Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other'
, 'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeed
s', 'Literature_Writing', 'Mathematics', 'Literacy']

Shape of project_subject_subcategories matrix after one hot encodig  (22443, 30)
==========================================================

List of features in project_grade_category ['Grades 9-12', 'Grades 3-5', 'Grades PreK-2', 'Grades 6-8']

Shape of project_grade_category matrix after one hot encodig  (22443, 4)
==========================================================

List of features in teacher_prefix ['Mrs.', 'Ms.', 'Mr.', 'Teacher', 'Dr.']

Shape of teacher_prefix matrix after one hot encoding  (22443, 5)
```

In [31]:

```
# Transform categorical for test data
ts_school_state = vectorizer_school_state.transform(ts_X['school_state'].values)
ts_project_subject_category = vectorizer_categories.transform(ts_X['clean_categories'].values)
ts_project_subject_subcategory = vectorizer_subcategories.transform(ts_X['clean_subcategories'].values)
ts_project_grade_category = vectorizer_grade_category.transform(ts_X['project_grade_category'].values)
ts_teacher_prefix = vectorizer_teacher_prefix.transform(ts_X['teacher_prefix'].values)

# Transform categorical for cv data
cv_school_state = vectorizer_school_state.transform(cv_X['school_state'].values)
cv_project_subject_category = vectorizer_categories.transform(cv_X['clean_categories'].values)
cv_project_subject_subcategory = vectorizer_subcategories.transform(cv_X['clean_subcategories'].values)
cv_project_grade_category = vectorizer_grade_category.transform(cv_X['project_grade_category'].values)
cv_teacher_prefix = vectorizer_teacher_prefix.transform(cv_X['teacher_prefix'].values)
```

In [32]:

```
print('--------------Test data--------------')
print('Shape of school_state:', ts_school_state.shape)
```

```
print('Shape of project_subject_categories:', ts_project_subject_category.shape)
print('Shape of project_subject_subcategories:', ts_project_subject_subcategory.shape)
print('Shape of project_grade_category:', ts_project_grade_category.shape)
print('Shape of teacher_prefix:', ts_teacher_prefix.shape)

print('--------------CV data--------------')
print('Shape of school_state:', cv_school_state.shape)
print('Shape of project_subject_categories:', cv_project_subject_category.shape)
print('Shape of project_subject_subcategories:', cv_project_subject_subcategory.shape)
print('Shape of project_grade_category:', cv_project_grade_category.shape)
print('Shape of teacher_prefix:', cv_teacher_prefix.shape)
```

```
--------------Test data--------------
Shape of school_state: (16500, 51)
Shape of project_subject_categories: (16500, 9)
Shape of project_subject_subcategories: (16500, 30)
Shape of project_grade_category: (16500, 4)
Shape of teacher_prefix: (16500, 5)
--------------CV data--------------
Shape of school_state: (11055, 51)
Shape of project_subject_categories: (11055, 9)
Shape of project_subject_subcategories: (11055, 30)
Shape of project_grade_category: (11055, 4)
Shape of teacher_prefix: (11055, 5)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [34]:

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

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

Note:

> We already have preprocessed both essay and project_title in Text processing section (1.3 and 1.4) above

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 [82]:

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

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

BoW

In [36]:

```
### BoW in Essay and Title on Train

# # We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer_bow = CountVectorizer(min_df=20)
text_bow = vectorizer_bow.fit_transform(tr_X['clean_essay'].values)
print("Shape of essay matrix after one hot encodig on train",text_bow.shape)

# # Similarly you can vectorize for title also
vectorizer_bowt = CountVectorizer(min_df=20, max_features=5000)
title_bow = vectorizer_bowt.fit_transform(tr_X['clean_project_title'])
print("Shape of title matrix after one hot encodig ",title_bow.shape)

### BoW in Essay and Title on CV
print('============================================================\n')
cv_essay = vectorizer_bow.transform(cv_X['clean_essay'])
print("Shape of essay matrix after one hot encodig on cv",cv_essay.shape)

cv_title = vectorizer_bowt.transform(cv_X['clean_project_title'])
print("Shape of title matrix after one hot encodig on cv",cv_title.shape)

### BoW in Essay and Title on Test
print('============================================================\n')
ts_essay = vectorizer_bow.transform(ts_X['clean_essay'])
print("Shape of essay matrix after one hot encodig on test",ts_essay.shape)

ts_title = vectorizer_bowt.transform(ts_X['clean_project_title'])
print("Shape of title matrix after one hot encodig on test",ts_title.shape)
```

```
Shape of essay matrix after one hot encodig on train (22444, 6411)
Shape of title matrix after one hot encodig  (22444, 638)
============================================================

Shape of essay matrix after one hot encodig on cv (11055, 6411)
Shape of title matrix after one hot encodig on cv (11055, 638)
============================================================

Shape of essay matrix after one hot encodig on test (16500, 6411)
Shape of title matrix after one hot encodig on test (16500, 638)
```

In [37]:

```
## Convert them into dense and standardize it

text_bow = text_bow.toarray()

# For essay in train data
text_scalar = StandardScaler()
text_scalar.fit(text_bow)
print(f"for essay in train data -> Mean : {text_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantit
y_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
text_normalized = text_scalar.transform(text_bow)

# For title in train data
title_bow = title_bow.toarray()
title_scalar = StandardScaler()
title_scalar.fit(title_bow)
print(f"for title in train data -> Mean : {title_scalar.mean_[0]}, Standard deviation : {np.sqrt(quanti
ty_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
title_normalized = title_scalar.transform(title_bow)

# Transform essay and title in cv data from prefit in train data
cv_essay = cv_essay.toarray()
cv_title = cv_title.toarray()
cv_essay_normalized = text_scalar.transform(cv_essay)
cv_title_normalized = title_scalar.transform(cv_title)
```

```
# Transform essay and title in test data from prefit in train data
ts_essay = ts_essay.toarray()
ts_title = ts_title.toarray()
ts_essay_normalized = text_scalar.transform(ts_essay)
ts_title_normalized = title_scalar.transform(ts_title)
```

```
for essay in train data -> Mean : 0.0023614328996613794, Standard deviation : 26.100176112712074
for title in train data -> Mean : 0.0010247727677775798, Standard deviation : 26.100176112712074
```

In [38]:

```
print('Shape of normalized essay in train data', text_normalized.shape)
print('Shape of normalized title in train data', title_normalized.shape)
print('=====================================================\n')
print('Shape of normalized essay in cv data', cv_essay_normalized.shape)
print('Shape of normalized title in cv data', cv_title_normalized.shape)
print('=====================================================\n')
print('Shape of normalized essay in test data', ts_essay_normalized.shape)
print('Shape of normalized title in test data', ts_title_normalized.shape)
```

```
Shape of normalized essay in train data (22444, 6411)
Shape of normalized title in train data (22444, 638)
=====================================================

Shape of normalized essay in cv data (11055, 6411)
Shape of normalized title in cv data (11055, 638)
=====================================================

Shape of normalized essay in test data (16500, 6411)
Shape of normalized title in test data (16500, 638)
```

TFIDF

In [60]:

```
# # We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer_tfidf = TfidfVectorizer(min_df=20)
text_tfidf = vectorizer_tfidf.fit_transform(tr_X['clean_essay'].values)
print("Shape of essay matrix after one hot encodig on train",text_tfidf.shape)

# # Similarly you can vectorize for title also
vectorizer_tfidft = TfidfVectorizer(min_df=20)
title_tfidf = vectorizer_tfidft.fit_transform(tr_X['clean_project_title'])
print("Shape of title matrix after one hot encodig on train ",title_tfidf.shape)

### TFIDF in Essay and Title on CV
print('=============================================================\n')
cv_essay = vectorizer_tfidf.transform(cv_X['clean_essay'])
print("Shape of essay matrix after one hot encodig on cv",cv_essay.shape)

cv_title = vectorizer_tfidft.transform(cv_X['clean_project_title'])
print("Shape of title matrix after one hot encodig on cv",cv_title.shape)

### TFIDF in Essay and Title on Test
print('=============================================================\n')
ts_essay = vectorizer_tfidf.transform(ts_X['clean_essay'])
print("Shape of essay matrix after one hot encodig on test",ts_essay.shape)

ts_title = vectorizer_tfidft.transform(ts_X['clean_project_title'])
print("Shape of title matrix after one hot encodig on test",ts_title.shape)
```

```
Shape of essay matrix after one hot encodig on train (22444, 6411)
Shape of title matrix after one hot encodig on train  (22444, 638)
=============================================================

Shape of essay matrix after one hot encodig on cv (11055, 6411)
Shape of title matrix after one hot encodig on cv (11055, 638)
=============================================================

Shape of essay matrix after one hot encodig on test (16500, 6411)
```

```
Shape of essay matrix after one hot encodig on test (16500, 6411)
Shape of title matrix after one hot encodig on test (16500, 638)
```

In [61]:

```
## Convert them into dense and standardize it

text_tfidf = text_tfidf.toarray()

# For essay and title in train data
text_scalar = StandardScaler()
text_scalar.fit(text_tfidf)
print(f"on Essay-> Mean : {text_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scalar.var_[0]
)}")

# Now standardize the data with above mean and variance.
text_normalized = text_scalar.transform(text_tfidf)

title_tfidf = title_tfidf.toarray()
title_scalar = StandardScaler()
title_scalar.fit(title_tfidf)
print(f"on Title-> Mean : {title_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scalar.var_[0
])}")

# Now standardize the data with above mean and variance.
title_normalized = title_scalar.transform(title_tfidf)

# Transform essay and title in cv data from prefit in train data
cv_essay = cv_essay.toarray()
cv_title = cv_title.toarray()
cv_essay_normalized = text_scalar.transform(cv_essay)
cv_title_normalized = title_scalar.transform(cv_title)

# Transform essay and title in test data from prefit in train data
ts_essay = ts_essay.toarray()
ts_title = ts_title.toarray()
ts_essay_normalized = text_scalar.transform(ts_essay)
ts_title_normalized = title_scalar.transform(ts_title)
```

```
on Essay-> Mean : 0.00029055486219502674, Standard deviation : 26.100176112712074
on Title-> Mean : 0.000702522396140204, Standard deviation : 26.100176112712074
```

In [62]:

```
print('Shape of normalized essay in train data', text_normalized.shape)
print('Shape of normalized title in train data', title_normalized.shape)
print('=====================================================\n')
print('Shape of normalized essay in cv data', cv_essay_normalized.shape)
print('Shape of normalized title in cv data', cv_title_normalized.shape)
print('=====================================================\n')
print('Shape of normalized essay in test data', ts_essay_normalized.shape)
print('Shape of normalized title in test data', ts_title_normalized.shape)
```

```
Shape of normalized essay in train data (22444, 6411)
Shape of normalized title in train data (22444, 638)
=====================================================

Shape of normalized essay in cv data (11055, 6411)
Shape of normalized title in cv data (11055, 638)
=====================================================

Shape of normalized essay in test data (16500, 6411)
Shape of normalized title in test data (16500, 638)
```

Avgw2v

In [37]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-an
d-load-variables-in-python/
```

```
# load variables in python)
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())

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

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

# average Word2Vec for train
# compute average word2vec for each title.
avg_w2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(tr_X['clean_project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_title.append(vector)

print(len(avg_w2v_title))
print(len(avg_w2v_title[0]))
```

```
100%|| 22443/22443 [00:12<00:00, 1742.58it/s]
```

```
22443
300
```

```
100%|| 22443/22443 [00:00<00:00, 34875.74it/s]
```

```
22443
300
```

In [38]:

```
# average Word2Vec for cv
# compute average word2vec for each essay
avg_w2v_cv_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(cv_X['clean_essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_cv_vectors.append(vector)

print(len(avg_w2v_cv_vectors))
print(len(avg_w2v_cv_vectors[0]))

# average Word2Vec for cv
```

```
# compute average word2vec for each title
avg_w2v_cv_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(cv_X['clean_project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_cv_title.append(vector)

print(len(avg_w2v_cv_title))
print(len(avg_w2v_cv_title[0]))
```

```
100%|██████████| 11055/11055 [00:06<00:00, 1775.81it/s]
```

```
11055
300
```

```
100%|██████████| 11055/11055 [00:00<00:00, 40500.16it/s]
```

```
11055
300
```

In [39]:

```
# average Word2Vec for test
# compute average word2vec for each essay
avg_w2v_ts_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(ts_X['clean_essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_ts_vectors.append(vector)

print(len(avg_w2v_ts_vectors))
print(len(avg_w2v_ts_vectors[0]))

# average Word2Vec for test
# compute average word2vec for each title
avg_w2v_ts_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(ts_X['clean_project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_ts_title.append(vector)

print(len(avg_w2v_ts_title))
print(len(avg_w2v_ts_title[0]))
```

```
100%|██████████| 16500/16500 [00:09<00:00, 1758.27it/s]
```

```
16500
300
```

```
100%|██████████| 16500/16500 [00:00<00:00, 41158.19it/s]
```

```
16500
300
```

In [40]:

```
avg_w2v_vectors = np.array(avg_w2v_vectors)

# For essay and title in train data
text_scalar = StandardScaler()
text_scalar.fit(avg_w2v_vectors)
print(f"on Essay-> Mean : {text_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
text_normalized = text_scalar.transform(avg_w2v_vectors)

avg_w2v_title = np.array(avg_w2v_title)
title_scalar = StandardScaler()
title_scalar.fit(avg_w2v_title)
print(f"on Title-> Mean : {title_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scalar.var_[0])}")

# Now standardize the data with above mean and variance.
title_normalized = title_scalar.transform(avg_w2v_title)

# Tranform CV and Test data
avg_w2v_cv_vectors = np.array(avg_w2v_cv_vectors)
avg_w2v_cv_title = np.array(avg_w2v_cv_title)
cv_essay_normalized = text_scalar.transform(avg_w2v_cv_vectors)
cv_title_normalized = title_scalar.transform(avg_w2v_cv_title)

avg_w2v_ts_vectors = np.array(avg_w2v_ts_vectors)
avg_w2v_ts_title = np.array(avg_w2v_ts_title)
ts_essay_normalized = text_scalar.transform(avg_w2v_ts_vectors)
ts_title_normalized = title_scalar.transform(avg_w2v_ts_title)
```

```
on Essay-> Mean : 0.014661379201408935, Standard deviation : 26.732288128099512
on Title-> Mean : -0.03810307021204848, Standard deviation : 26.732288128099512
```

In [41]:

```
print('Shape of normalized essay in train data', text_normalized.shape)
print('Shape of normalized title in train data', title_normalized.shape)
print('='*100+'\n')
print('Shape of normalized essay in cv data', cv_essay_normalized.shape)
print('Shape of normalized title in cv data', cv_title_normalized.shape)
print('='*100+'\n')
print('Shape of normalized essay in test data', ts_essay_normalized.shape)
print('Shape of normalized title in test data', ts_title_normalized.shape)
```

```
Shape of normalized essay in train data (22443, 300)
Shape of normalized title in train data (22443, 300)
====================================================

Shape of normalized essay in cv data (11055, 300)
Shape of normalized title in cv data (11055, 300)
====================================================

Shape of normalized essay in test data (16500, 300)
Shape of normalized title in test data (16500, 300)
```

TFIDF W2V

In [35]:

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

# Tfidf weighted w2v on essay in train
tfidf_model = TfidfVectorizer()
tfidf_model.fit(tr_X['clean_essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())

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

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

# Tfidf weighted w2v on title in train
tfidf_model2 = TfidfVectorizer()
tfidf_model2.fit(tr_X['clean_project_title'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary2 = dict(zip(tfidf_model2.get_feature_names(), list(tfidf_model2.idf_)))
tfidf_words2 = set(tfidf_model2.get_feature_names())

# tfidf Word2Vec
# compute average word2vec for each title
tfidf_w2v_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(tr_X['clean_project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_title.append(vector)

print(len(tfidf_w2v_title))
print(len(tfidf_w2v_title[0]))
```

```
100%|██████████| 22444/22444 [00:32<00:00, 684.32it/s]
```

```
22444
300
```

```
100%|██████████| 22444/22444 [00:00<00:0
```

```
0, 45830.64it/s]
```

```
22444
300
```

In [36]:

```
# tfidf Word2Vec in essay on cv
# compute average word2vec for each essay
tfidf_w2v_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(cv_X['clean_essay']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word
)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf
value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_cv.append(vector)

print(len(tfidf_w2v_vectors_cv))
print(len(tfidf_w2v_vectors_cv[0]))

# tfidf Word2Vec on title on cv
# compute average word2vec for each title
tfidf_w2v_title_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(cv_X['clean_project_title']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word
)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf
value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_title_cv.append(vector)

print(len(tfidf_w2v_title_cv))
print(len(tfidf_w2v_title_cv[0]))
```

```
100%|| 11055/11055 [00:16<00
:00, 659.32it/s]
```

```
11055
300
```

```
100%|| 11055/11055 [00:00<00:0
0, 43799.70it/s]
```

```
11055
300
```

In [37]:

```
# average Word2Vec for test
# compute average word2vec for each essay
tfidf_w2v_ts_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(ts_X['clean_essay'].values): # for each review/sentence
```

```
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word
)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf
value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_ts_vectors.append(vector)

print(len(tfidf_w2v_ts_vectors))
print(len(tfidf_w2v_ts_vectors[0]))

# average Word2Vec for test
# compute average word2vec for each title
tfidf_w2v_ts_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(ts_X['clean_project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word
)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf
value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_ts_title.append(vector)

print(len(tfidf_w2v_ts_title))
print(len(tfidf_w2v_ts_title[0]))
```

```
100%|██████████| 16500/16500 [00:24<00
:00, 668.31it/s]
```

```
16500
300
```

```
100%|██████████| 16500/16500 [00:00<00:0
0, 45951.12it/s]
```

```
16500
300
```

In [38]:

```
tfidf_w2v_vectors = np.array(tfidf_w2v_vectors)

# For essay and title in train data
text_scalar = StandardScaler()
text_scalar.fit(tfidf_w2v_vectors)
print(f"on Essay-> Mean : {text_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scalar.var_[0]
)}")

# Now standardize the data with above mean and variance.
text_normalized = text_scalar.transform(tfidf_w2v_vectors)

tfidf_w2v_title = np.array(tfidf_w2v_title)
title_scalar = StandardScaler()
title_scalar.fit(tfidf_w2v_title)
print(f"on Title-> Mean : {title_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scalar.var_[0
])}")
```

```
# Now standardize the data with above mean and variance.
title_normalized = title_scalar.transform(tfidf_w2v_title)

# Now tranform test and cv and then standard them.
tfidf_w2v_vectors_cv = np.array(tfidf_w2v_vectors_cv)
tfidf_w2v_title_cv = np.array(tfidf_w2v_title_cv)
cv_essay_normalized = text_scalar.transform(tfidf_w2v_vectors_cv)
cv_title_normalized = title_scalar.transform(tfidf_w2v_title_cv)
tfidf_w2v_ts_vectors = np.array(tfidf_w2v_ts_vectors)
tfidf_w2v_ts_title = np.array(tfidf_w2v_ts_title)
ts_essay_normalized = text_scalar.transform(tfidf_w2v_ts_vectors)
ts_title_normalized = title_scalar.transform(tfidf_w2v_ts_title)
```

```
on Essay-> Mean : 0.01682446363430252, Standard deviation : 26.183192140503532
on Title-> Mean : -0.046626825468811965, Standard deviation : 26.183192140503532
```

In [39]:

```
print('Shape of normalized essay in train data', text_normalized.shape)
print('Shape of normalized title in train data', title_normalized.shape)
print('=====================================================\n')
print('Shape of normalized essay in cv data', cv_essay_normalized.shape)
print('Shape of normalized title in cv data', cv_title_normalized.shape)
print('=====================================================\n')
print('Shape of normalized essay in test data', ts_essay_normalized.shape)
print('Shape of normalized title in test data', ts_title_normalized.shape)
```

```
Shape of normalized essay in train data (22444, 300)
Shape of normalized title in train data (22444, 300)
=====================================================

Shape of normalized essay in cv data (11055, 300)
Shape of normalized title in cv data (11055, 300)
=====================================================

Shape of normalized essay in test data (16500, 300)
Shape of normalized title in test data (16500, 300)
```

In [ ]:

Merge them

In [40]:

```
# for train data
from scipy.sparse import hstack
tr_X = hstack((quantity_normalized, price_normalized, teacher_number_of_previously_posted_projects_normalized, \
               school_state_one_hot, categories_one_hot, subcategories_one_hot, project_grade_category_one_hot, \
               teacher_prefix_one_hot, text_normalized, title_normalized))
tr_X.shape
```

Out[40]:

```
(22444, 702)
```

In [41]:

```
tr_X = tr_X.toarray()
```

In [42]:

```
# for cv data
```

```
cv_X = hstack((cv_quantity, cv_price, cv_teacher_number_of_previously_posted_projects, cv_school_state,
\
              cv_project_subject_category, cv_project_subject_subcategory,cv_project_grade_category, \
              cv_teacher_prefix, cv_essay_normalized, cv_title_normalized))
cv_X.shape
```

Out[42]:

```
(11055, 702)
```

In [43]:

```
cv_X = cv_X.toarray()
```

In [44]:

```
# for test data
# for cv data
ts_X = hstack((ts_quantity, ts_price, ts_teacher_number_of_previously_posted_projects, ts_school_state,
\
              ts_project_subject_category, ts_project_subject_subcategory,ts_project_grade_category, \
              ts_teacher_prefix, ts_essay_normalized, ts_title_normalized))
ts_X.shape
```

Out[44]:

```
(16500, 702)
```

In [45]:

```
ts_X = ts_X.toarray()
```

Let define plot function so that we can use as reusibility

In [24]:

```
from sklearn.neighbors import KNeighborsClassifier
import tqdm

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

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred

def knnbrutealgo(X, y, cv_X, cv_y):
    """
    Parameters:
    X - train feature data
    y- train class data
    cv_X - valid feature data
    cv_y - valid class data

    Return:
    Print the AUC score of CV data

    """
    tr_score = []
```

```
    cv_score = []
    index = 0
    for i in tqdm.tqdm_notebook([1,5,11,25,51,101]):
        # Create knn model instance
        knn = KNeighborsClassifier(n_neighbors=i, algorithm='brute')
        # Fit the model with train data
        knn.fit(X,y)
        # Predict the cv data
        predict_cv = batch_predict(knn, cv_X)
        predict_tr = batch_predict(knn, X)
        # Evaluate accuracy to see how much it corrected class label
        tr_score.append(metrics.roc_auc_score(y, predict_tr))
        cv_score.append(metrics.roc_auc_score(cv_y, predict_cv))
        print('\nTrain AUC and CV AUC score for k:{0} is {1} , {2}'.format(i,tr_score[index],cv_score[i
ndex]))
        index += 1
    return tr_score, cv_score


def plotauc_tr_cv(feature_name, n_list, X_score, cv_X_score):
    """
    Parameters:
    k - number of neighbors
    X_score - Train AUC score
    cv_X_score - CV AUC score
    Return:
    Save FPR, TRP and ROC for train data and Plot the graph of Train and CV data
    """
    plt.plot(n_list, X_score, label='Train AUC')
    plt.plot(n_list, cv_X_score, label='CV AUC')
    plt.scatter(n_list, X_score)
    plt.scatter(n_list, cv_X_score)

    plt.legend()
    plt.xlabel('Hyperparameter(k) ')
    plt.ylabel('AUC Score')
    plt.title('Train AUC vs CV AUC plot with {0} features'.format(feature_name))
    plt.show()

def plotauc_tr_ts(k, feature_name, X, y, ts_X, ts_y):
    """
    Parameters:
    k = number of neigbhors
    feature_name - (string) Write feature to print the plot title
    X - train feature data
    y - train class data
    fpr - FPR value for train data
    tpr - TPR value for train data
    roc_auc - AUC value of train data

    Return:
    Save the prediction of test data and plot the graph for Train and Test data
    """
    knn = KNeighborsClassifier(n_neighbors=k, algorithm='brute')
    # Fit the model with train data
    knn.fit(X,y)

    tr_predict = batch_predict(knn, X)
    ts_predict = batch_predict(knn, ts_X)

    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    fpr, tpr, tr_thre = roc_curve(y, tr_predict)
    roc_auc = auc(fpr, tpr)

    fpr_t = dict()
    tpr_t = dict()
    roc_auc_t = dict()
    fpr_t, tpr_t, _ = roc_curve(ts_y, ts_predict)
    roc_auc_t = auc(fpr_t, tpr_t)

    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
```

```
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot(fpr_t, tpr_t, color='blue',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_t)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC With Maximum AUC on KNN Classifier for k={0} on {1} features'.format(k,feature_name)
)
    plt.legend(loc="lower right")
    plt.show()

    return tr_thre, fpr, tpr, tr_predict, ts_predict
```

In [25]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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


def plot_cm(feature_names, tr_thresholds, train_fpr, train_tpr, y_train, y_train_pred, y_test, y_test_p
red):
    """
    Parameters:
    k = number of neigbhors
    feature_name - (string) Write feature to print the plot title
    y_true - test class data
    y_pred - test prediction value

    Return:
    Plot the confusion matrix
    """
    best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
    print("Train confusion matrix")
#     print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t)))
    cm = metrics.confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
    plt.figure(figsize = (10,7))
    sns.heatmap(cm, annot=True, fmt="d")
    plt.xlabel('Predicted Class')
    plt.ylabel('True Class')
    plt.title('Confusion matrix for Train Data when KNN with {0} features'.format(feature_names))

    print("Test confusion matrix")
#     print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
    cm = metrics.confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
    plt.figure(figsize = (10,7))
    sns.heatmap(cm, annot=True, fmt="d")
    plt.xlabel('Predicted Class')
    plt.ylabel('True Class')
    plt.title('Confusion matrix for Test Data when KNN with {0} features'.format(feature_names))
```

2.4.1 Applying KNN brute force on BOW, SET 1

In [47]:

```
# Please write all the code with proper documentation
tr_score, cv_score = knnbrutealgo(tr_X, tr_y, cv_X, cv_y)
```

```
Train AUC and CV AUC score for k:1 is 1.0 , 0.49775563607213086

Train AUC and CV AUC score for k:5 is 0.8922648077871687 , 0.5229818842782452

Train AUC and CV AUC score for k:11 is 0.8044556798151272 , 0.5591860712614423

Train AUC and CV AUC score for k:25 is 0.7608875755992128 , 0.5380287877147506

Train AUC and CV AUC score for k:51 is 0.7246087311085125 , 0.5673851770443061

Train AUC and CV AUC score for k:101 is 0.704552949200877 , 0.5886356214303946
```

> Observation: We found that k=101 got the maximum AUC score for kNN
> **Note: I performed only 6 values because of taking long computation time.**

In [85]:

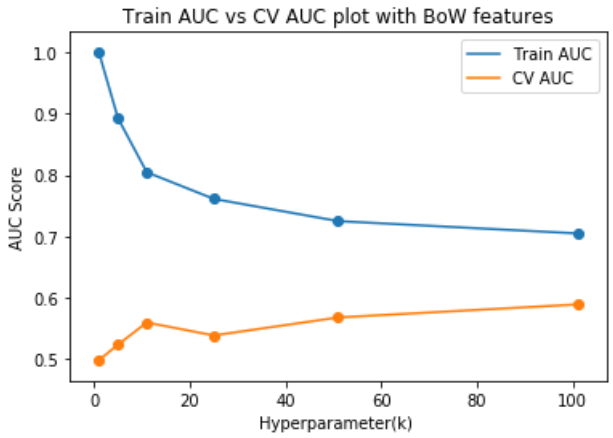
```
len(tr_score), len(cv_score)
```

Out[85]:

```
(6, 6)
```

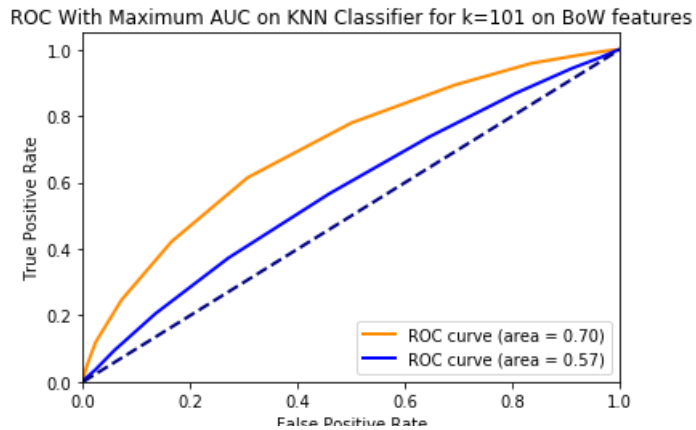
In [48]:

```
plotauc_tr_cv('BoW', [1,5,11,25,51,101], tr_score, cv_score)
```



In [49]:

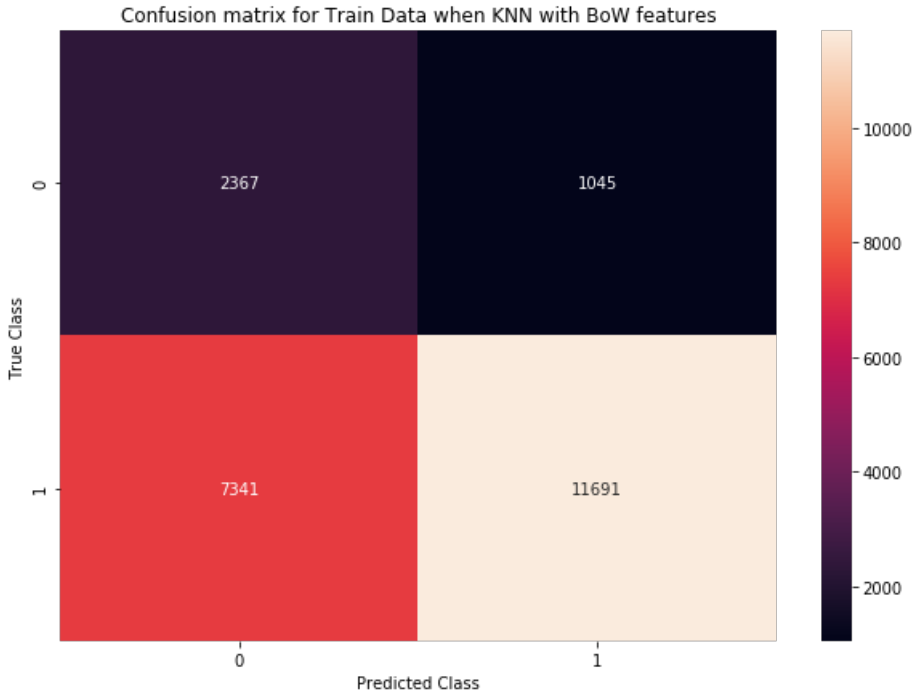
```
tr_thre, fpr, tpr, tr_predict, ts_predict = plotauc_tr_ts(101, 'BoW', tr_X, tr_y, ts_X, ts_y)
```

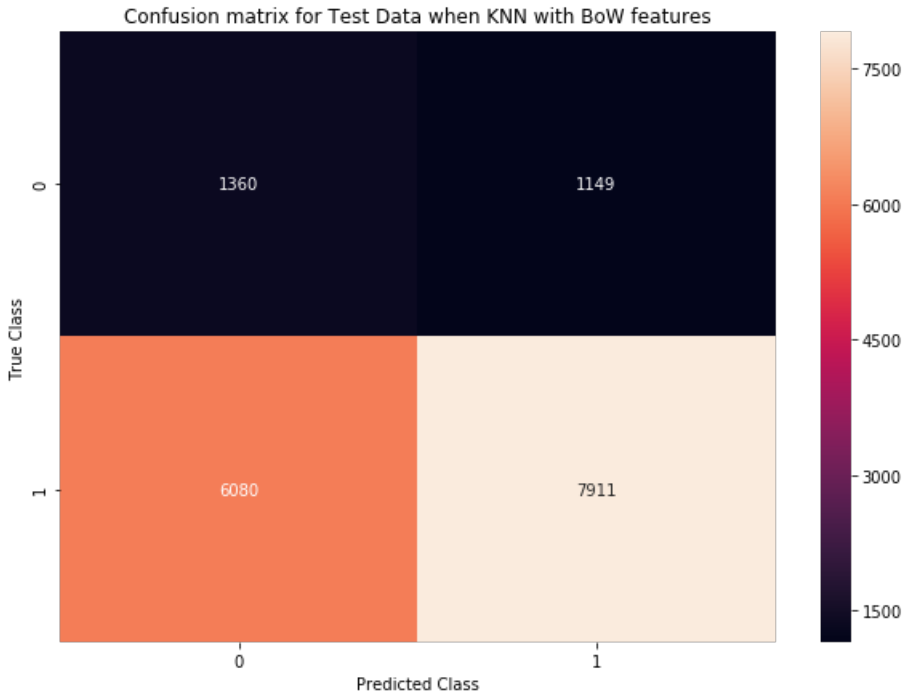


In [50]:

```
plot_cm('BoW', tr_thre, fpr, tpr, tr_y, tr_predict, ts_y, ts_predict)
```

```
the maximum value of tpr*(1-fpr) 0.4261440871843165 for threshold 0.772
Train confusion matrix
Test confusion matrix
```





2.4.2 Applying KNN brute force on TFIDF, SET 2

In [69]:

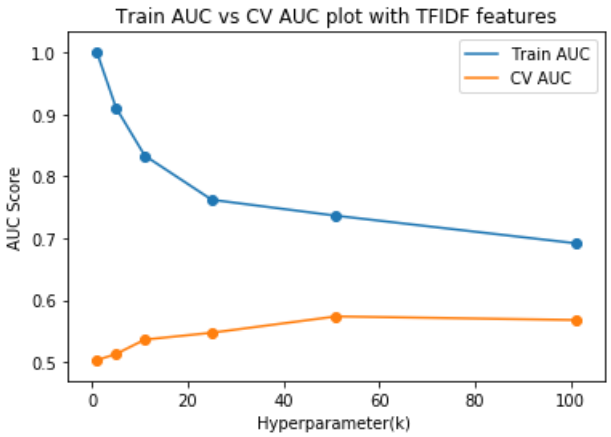
```
# Please write all the code with proper documentation
tr_score, cv_score = knnbrutealgo(tr_X, tr_y, cv_X, cv_y)
```

```
Train AUC and CV AUC score for k:1 is 1.0 , 0.503109845894964

Train AUC and CV AUC score for k:5 is 0.9104986212521935 , 0.5130136110017113

Train AUC and CV AUC score for k:11 is 0.8332544879063435 , 0.5362354415563597

Train AUC and CV AUC score for k:25 is 0.7620352616460855 , 0.5473192968463533

Train AUC and CV AUC score for k:51 is 0.736242381560617 , 0.5736983787094736

Train AUC and CV AUC score for k:101 is 0.691833795872639 , 0.5678501245169502
```

> Observation: we observe that for k=51 in kNN got the maximum AUC score.

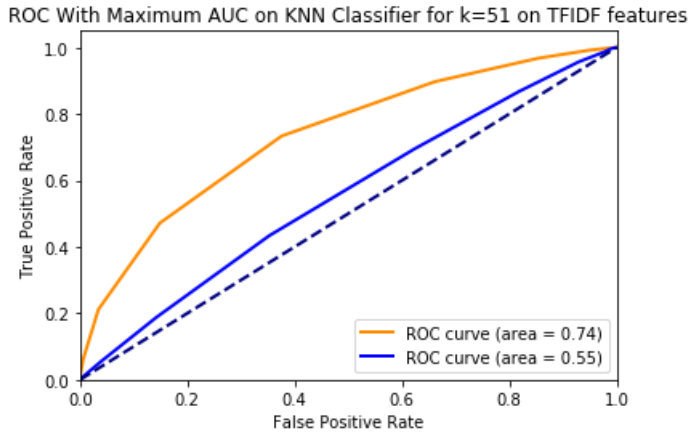
In [70]:

```
plotauc_tr_cv('TFIDF', [1,5,11,25,51,101], tr_score, cv_score)
```



In [71]:

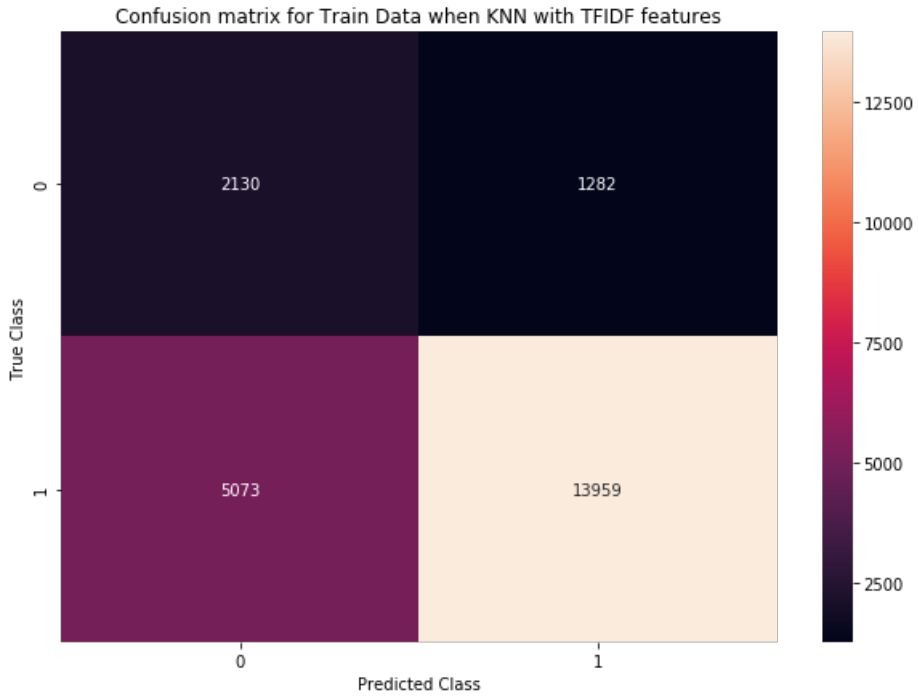
```
tr_thre, fpr, tpr, tr_predict, ts_predict = plotauc_tr_ts(51, 'TFIDF', tr_X, tr_y, ts_X, ts_y)
```

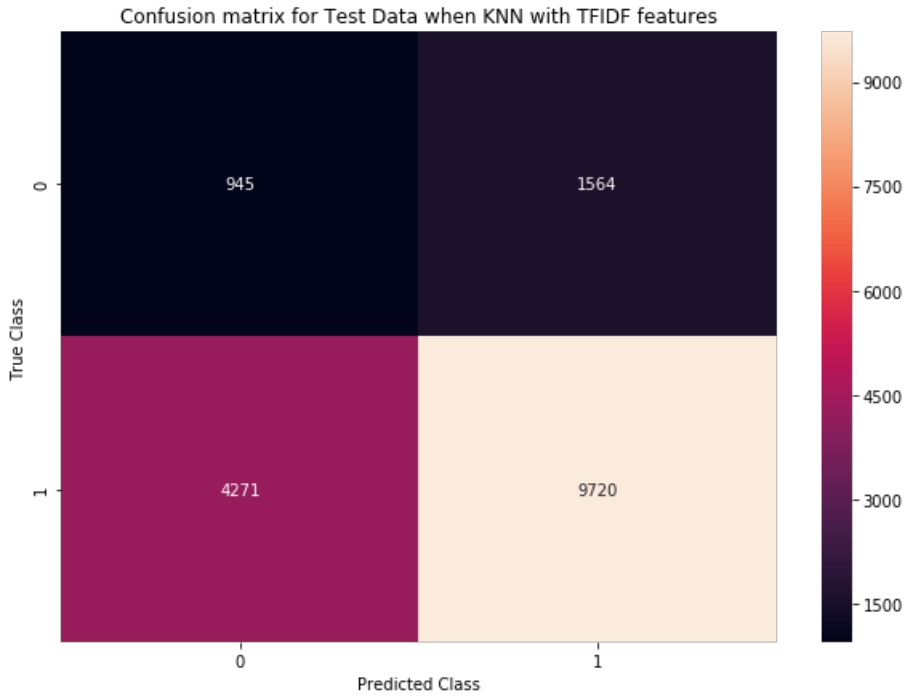


In [72]:

```
plot_cm('TFIDF', tr_thre, fpr, tpr, tr_y, tr_predict, ts_y, ts_predict)
```

```
the maximum value of tpr*(1-fpr) 0.45786817611308794 for threshold 0.824
Train confusion matrix
Test confusion matrix
```





2.4.3 Applying KNN brute force on AVG W2V, SET 3

In [51]:

```
# Please write all the code with proper documentation
tr_score, cv_score = knnbrutealgo(tr_X, tr_y, cv_X, cv_y)
```

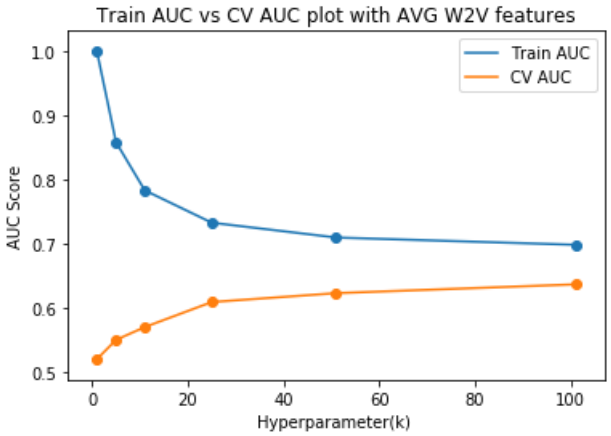
```
Train AUC and CV AUC score for k:1 is 1.0 , 0.5193989420818714

Train AUC and CV AUC score for k:5 is 0.858071543377767 , 0.549481451186435

Train AUC and CV AUC score for k:11 is 0.7829660427190236 , 0.5694212668456928

Train AUC and CV AUC score for k:25 is 0.7324374948189665 , 0.6085454941174531

Train AUC and CV AUC score for k:51 is 0.7092965321574118 , 0.6224060443401154
```

```
Train AUC and CV AUC score for k:101 is 0.697958315381071 , 0.6361381972743773
```

> Observation: we observe that for k=101 in kNN got the maximum AUC score.

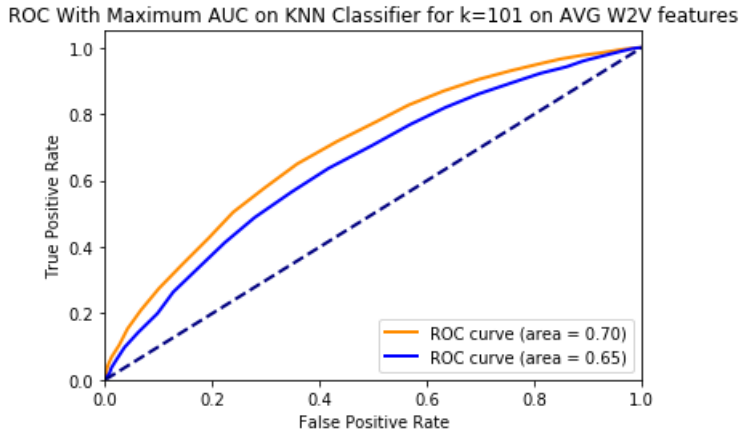
In [52]:

```
plotauc_tr_cv('AVG W2V', [1,5,11,25,51,101], tr_score, cv_score)
```



In [57]:

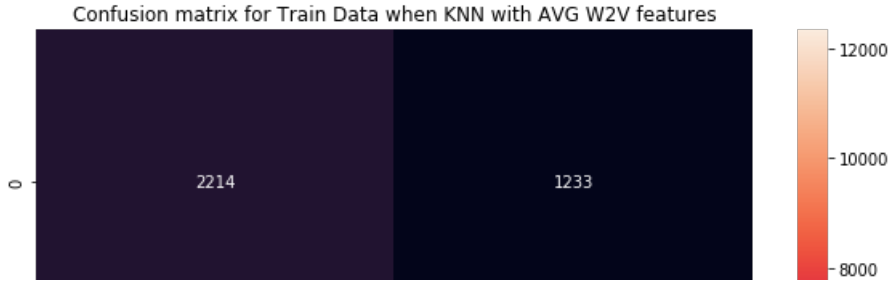
```
tr_thre, fpr, tpr, tr_predict, ts_predict = plotauc_tr_ts(101, 'AVG W2V', tr_X, tr_y, ts_X, ts_y)
```

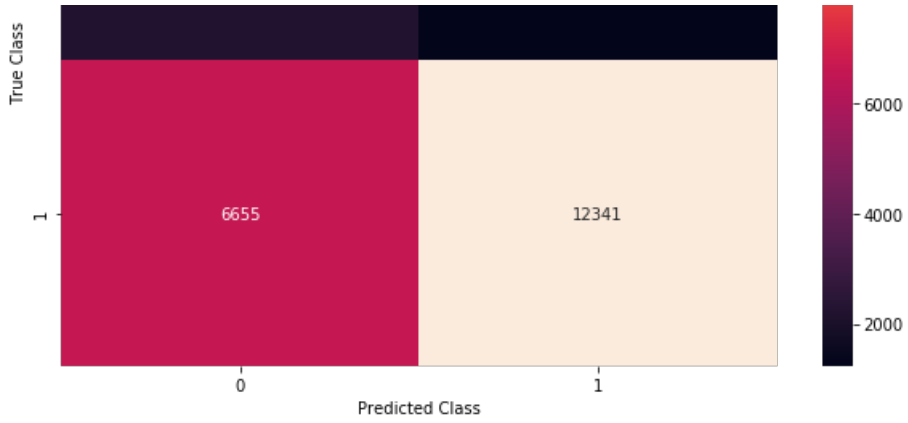


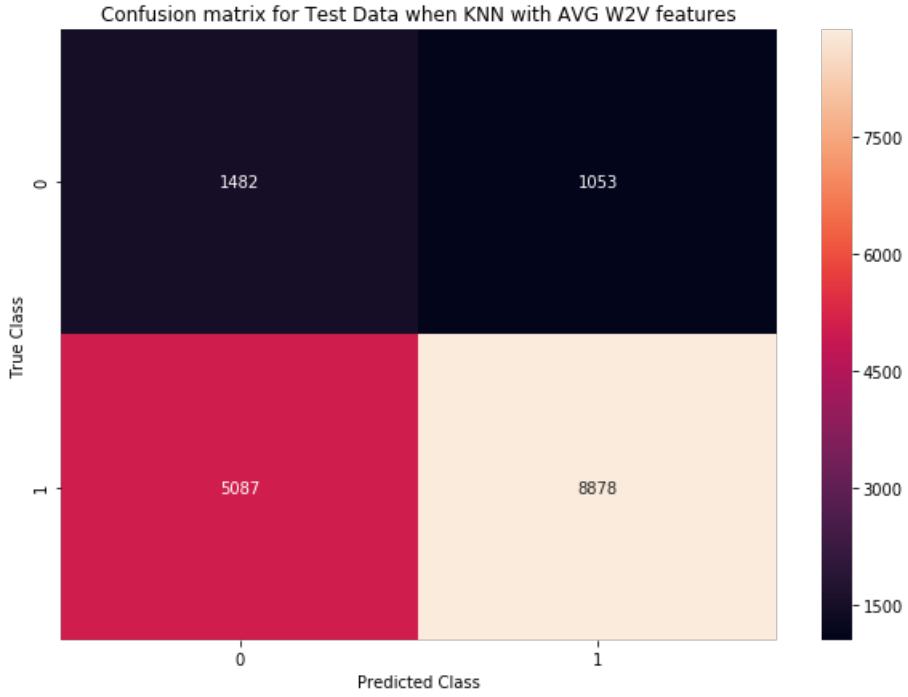
In [58]:

```
plot_cm('AVG W2V', tr_thre, fpr, tpr, tr_y, tr_predict, ts_y, ts_predict)
```

```
the maximum value of tpr*(1-fpr) 0.41727707413461235 for threshold 0.851
Train confusion matrix
Test confusion matrix
```







2.4.4 Applying KNN brute force on TFIDF W2V, SET 4

In [48]:

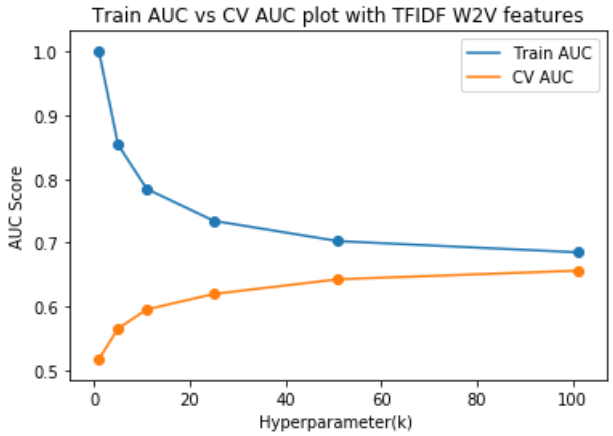
```
# Please write all the code with proper documentation
tr_score, cv_score = knnbrutealgo(tr_X, tr_y, cv_X, cv_y)
```

```
Train AUC and CV AUC score for k:1 is 1.0 , 0.5176652452025586

Train AUC and CV AUC score for k:5 is 0.8538605180572544 , 0.5655940234859815

Train AUC and CV AUC score for k:11 is 0.7843711466943424 , 0.5955390319192948

Train AUC and CV AUC score for k:25 is 0.7343999267949146 , 0.6197741145021163

Train AUC and CV AUC score for k:51 is 0.7026077844496337 , 0.642814117048022

Train AUC and CV AUC score for k:101 is 0.6850830661502156 , 0.6562650924482066
```

> Observation: we observe that for k=101 in kNN got the maximum AUC score.

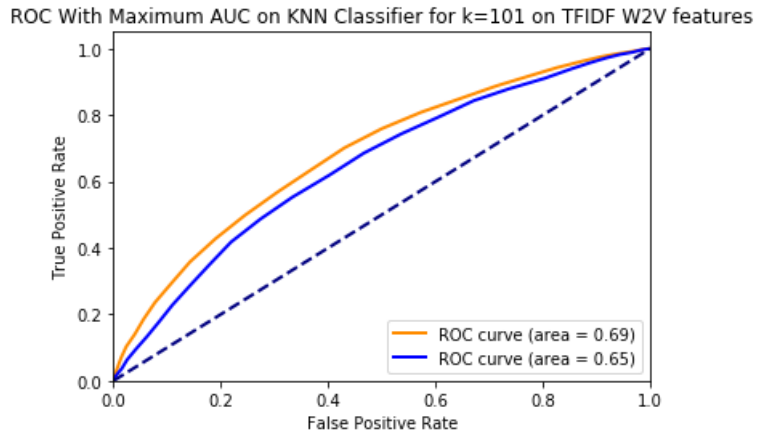
In [49]:

```
plotauc_tr_cv('TFIDF W2V', [1,5,11,25,51,101], tr_score, cv_score)
```



In [50]:

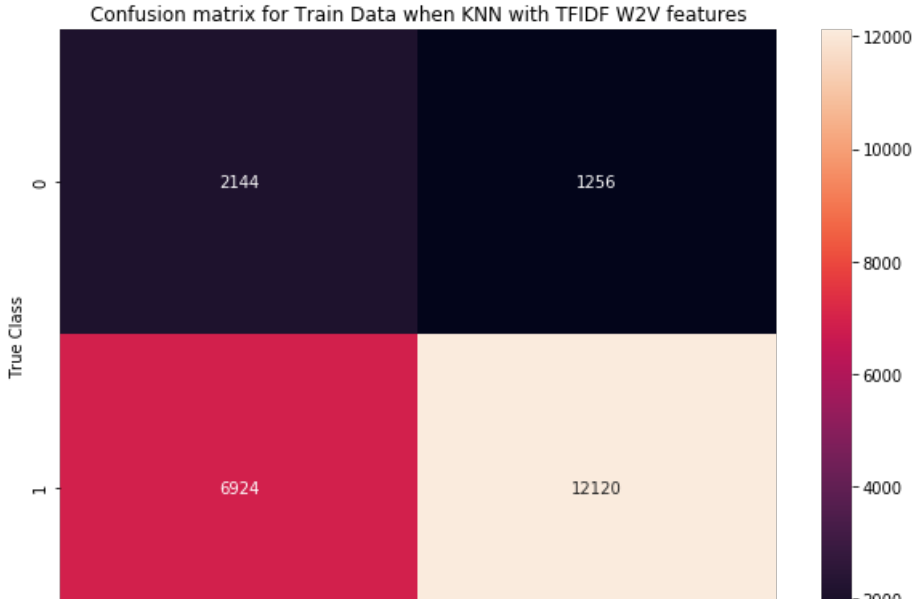
```
tr_thre, fpr, tpr, tr_predict, ts_predict = plotauc_tr_ts(101, 'TFIDF W2V', tr_X, tr_y, ts_X, ts_y)
```

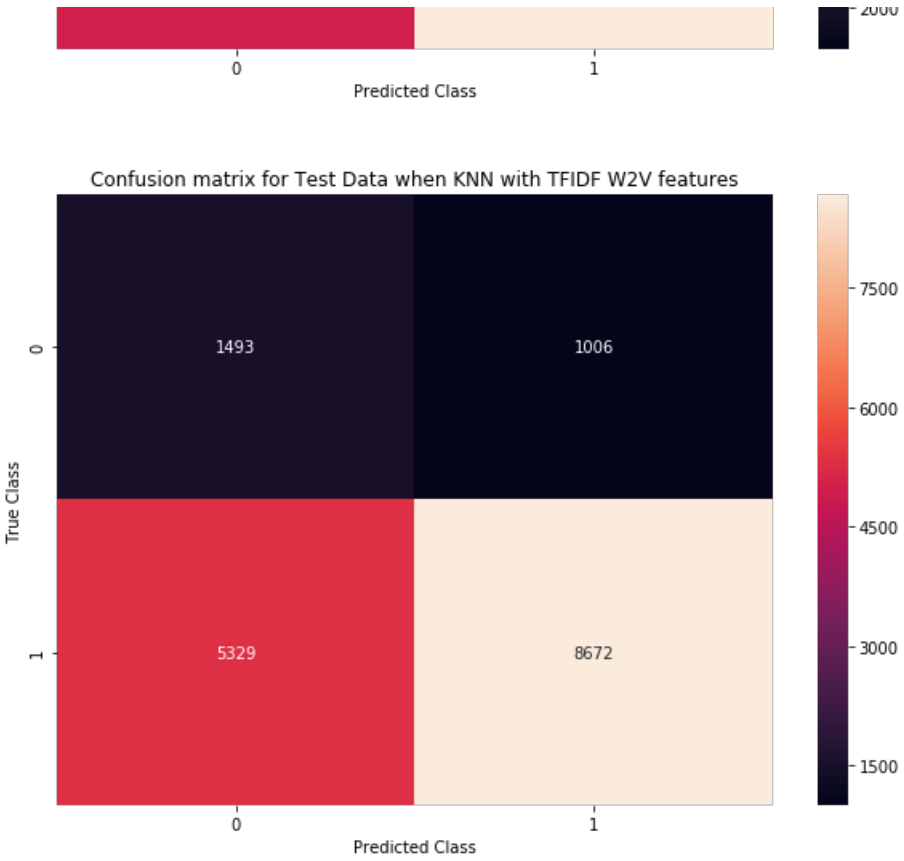


In [51]:

```
plot_cm('TFIDF W2V', tr_thre, fpr, tpr, tr_y, tr_predict, ts_y, ts_predict)
```

```
the maximum value of tpr*(1-fpr) 0.401319544831165 for threshold 0.842
Train confusion matrix
Test confusion matrix
```





2.5 Feature selection with `SelectKBest`

In [33]:

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

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

# We are going to represent the quantity, as numerical values within the range 0-1
# normalization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

from sklearn.preprocessing import MinMaxScaler

quantity_scalar = MinMaxScaler()
quantity_scalar.fit(tr_X['quantity'].values.reshape(-1,1)) # finding the mean and standard deviation of this data

quantity_normalized = quantity_scalar.transform(tr_X['quantity'].values.reshape(-1, 1))

### 2) price

# the cost feature is already in numerical values,  we are going to represent the money, as numerical values within the range 0-1
# normalization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

price_scalar = MinMaxScaler()
price_scalar.fit(tr_X['price'].values.reshape(-1,1)) # finding the mean and standard deviation of this data

price_normalized = price_scalar.transform(tr_X['price'].values.reshape(-1, 1))

### 3) For teacher number of previously projects
```

```
### 3) for teacher_number_of_previously_projects

# We are going to represent the teacher_number_of_previously_posted_projects, as numerical values withi
n the range 0-1
# normalization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Standa
rdScaler.html

teacher_number_of_previously_posted_projects_scalar = MinMaxScaler()
teacher_number_of_previously_posted_projects_scalar.fit(tr_X['teacher_number_of_previously_posted_proje
cts'].values.reshape(-1,1))

teacher_number_of_previously_posted_projects_normalized = teacher_number_of_previously_posted_projects_
scalar.transform(tr_X['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
```

In [39]:

```
# Transform numerical attributes for test data
ts_price = price_scalar.transform(ts_X['price'].values.reshape(-1,1))
ts_quantity = quantity_scalar.transform(ts_X['quantity'].values.reshape(-1,1))
ts_teacher_number_of_previously_posted_projects = \
teacher_number_of_previously_posted_projects_scalar.transform(ts_X['teacher_number_of_previously_posted
_projects'].\
                                                              values.reshape(-1,1))

# tranform nmerical attributes for cv data
cv_price = price_scalar.transform(cv_X['price'].values.reshape(-1,1))
cv_quantity = quantity_scalar.transform(cv_X['quantity'].values.reshape(-1,1))
cv_teacher_number_of_previously_posted_projects = \
teacher_number_of_previously_posted_projects_scalar.transform(cv_X['teacher_number_of_previously_posted
_projects'].\
                                                              values.reshape(-1,1))

print('--------------Test data--------------')
print('Shape of quantity:', ts_quantity.shape)
print('Shape of price:', ts_price.shape)
print('Shape of teacher_number_of_previously_posted_projects:', ts_teacher_number_of_previously_posted_
projects.shape)

print('--------------CV data--------------')
print('Shape of quantity:', cv_quantity.shape)
print('Shape of price:', cv_price.shape)
print('Shape of teacher_number_of_previously_posted_projects:', cv_teacher_number_of_previously_posted_
projects.shape)
```

```
--------------Test data--------------
Shape of quantity: (16500, 1)
Shape of price: (16500, 1)
Shape of teacher_number_of_previously_posted_projects: (16500, 1)
--------------CV data--------------
Shape of quantity: (11055, 1)
Shape of price: (11055, 1)
Shape of teacher_number_of_previously_posted_projects: (11055, 1)
```

In [34]:

```
# # We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer_tfidf = TfidfVectorizer(min_df=10)
text_tfidf = vectorizer_tfidf.fit_transform(tr_X['clean_essay'].values)
print("Shape of essay matrix after one hot encodig on train",text_tfidf.shape)

# # Similarly you can vectorize for title also
vectorizer_tfidft = TfidfVectorizer(min_df=10)
title_tfidf = vectorizer_tfidft.fit_transform(tr_X['clean_project_title'])
print("Shape of title matrix after one hot encodig on train ",title_tfidf.shape)

### TFIDF in Essay and Title on CV
print('==========================================================\n')
cv_essay = vectorizer_tfidf.transform(cv_X['clean_essay'])
print("Shape of essay matrix after one hot encodig on cv",cv_essay.shape)

cv_title = vectorizer_tfidft.transform(cv_X['clean_project_title'])
print("Shape of title matrix after one hot encodig on cv",cv_title.shape)

### TFIDF in Essay and Title on Test
```

```
print('==========================================================\n')
ts_essay = vectorizer_tfidf.transform(ts_X['clean_essay'])
print("Shape of essay matrix after one hot encodig on test",ts_essay.shape)

ts_title = vectorizer_tfidft.transform(ts_X['clean_project_title'])
print("Shape of title matrix after one hot encodig on test",ts_title.shape)
```

```
Shape of essay matrix after one hot encodig on train (22443, 8802)
Shape of title matrix after one hot encodig on train  (22443, 1158)
==========================================================

Shape of essay matrix after one hot encodig on cv (11055, 8802)
Shape of title matrix after one hot encodig on cv (11055, 1158)
==========================================================

Shape of essay matrix after one hot encodig on test (16500, 8802)
Shape of title matrix after one hot encodig on test (16500, 1158)
```

In [35]:

```
from sklearn.preprocessing import MinMaxScaler

text_tfidf = text_tfidf.toarray()

# For essay and title in train data
text_scalar = MinMaxScaler()
text_scalar.fit(text_tfidf)
# print(f"on Essay-> Mean : {text_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scalar.var_[
0])}")

# Now standardize the data with above mean and variance.
text_normalized = text_scalar.transform(text_tfidf)

title_tfidf = title_tfidf.toarray()
title_scalar = MinMaxScaler()
title_scalar.fit(title_tfidf)
# print(f"on Title-> Mean : {title_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scalar.var_
[0])}")

# Now standardize the data with above mean and variance.
title_normalized = title_scalar.transform(title_tfidf)

cv_essay = cv_essay.toarray()
cv_title = cv_title.toarray()
cv_essay_normalized = text_scalar.transform(cv_essay)
cv_title_normalized = title_scalar.transform(cv_title)

# Transform essay and title in test data from prefit in train data
ts_essay = ts_essay.toarray()
ts_title = ts_title.toarray()
ts_essay_normalized = text_scalar.transform(ts_essay)
ts_title_normalized = title_scalar.transform(ts_title)
```

In [36]:

```
print('Shape of normalized essay in train data', text_normalized.shape)
print('Shape of normalized title in train data', title_normalized.shape)
print('==========================================================\n')
print('Shape of normalized essay in cv data', cv_essay_normalized.shape)
print('Shape of normalized title in cv data', cv_title_normalized.shape)
print('==========================================================\n')
print('Shape of normalized essay in test data', ts_essay_normalized.shape)
print('Shape of normalized title in test data', ts_title_normalized.shape)
```

```
Shape of normalized essay in train data (22443, 8802)
Shape of normalized title in train data (22443, 1158)
==========================================================

Shape of normalized essay in cv data (11055, 8802)
Shape of normalized title in cv data (11055, 1158)
==========================================================
```

```
Shape of normalized essay in test data (16500, 8802)
Shape of normalized title in test data (16500, 1158)
```

In [37]:

```
# Merge Them
# for train data
from scipy.sparse import hstack
tr_X = hstack((quantity_normalized, price_normalized, teacher_number_of_previously_posted_projects_norm
alized, \
               school_state_one_hot, categories_one_hot, subcategories_one_hot, project_grade_category_o
ne_hot, \
               teacher_prefix_one_hot, text_normalized, title_normalized))
tr_X = tr_X.toarray()
tr_X.shape
```

Out[37]:

```
(22443, 10062)
```

In [40]:

```
# for cv data
cv_X = hstack((cv_quantity, cv_price, cv_teacher_number_of_previously_posted_projects, cv_school_state,
\
               cv_project_subject_category, cv_project_subject_subcategory,cv_project_grade_category, \
               cv_teacher_prefix, cv_essay_normalized, cv_title_normalized))
cv_X = cv_X.toarray()
cv_X.shape
```

Out[40]:

```
(11055, 10062)
```

In [41]:

```
# for test data
# for cv data
ts_X = hstack((ts_quantity, ts_price, ts_teacher_number_of_previously_posted_projects, ts_school_state,
\
               ts_project_subject_category, ts_project_subject_subcategory,ts_project_grade_category, \
               ts_teacher_prefix, ts_essay_normalized, ts_title_normalized))
ts_X = ts_X.toarray()
ts_X.shape
```

Out[41]:

```
(16500, 10062)
```

In [42]:

```
from sklearn.feature_selection import SelectKBest, chi2
kbest = SelectKBest(chi2, k=2000)
kbest.fit(tr_X, tr_y)
tr_X_new = kbest.transform(tr_X)
```

In [43]:

```
cv_X_new = kbest.transform(cv_X)
```

In [44]:

```
ts_X_new = kbest.transform(ts_X)
```

In [45]:

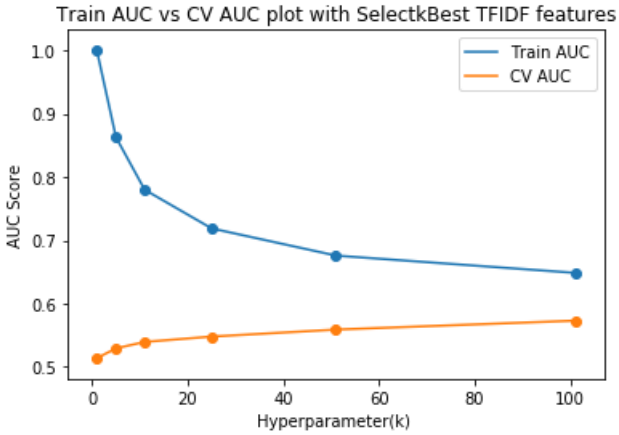
```
# Please write all the code with proper documentation
tr_score, cv_score = knnbrutealgo(tr_X_new, tr_y, cv_X_new, cv_y)
```

```
Train AUC and CV AUC score for k:1 is 1.0 , 0.5138785714285714

Train AUC and CV AUC score for k:5 is 0.8627202065148127 , 0.5292531746031746

Train AUC and CV AUC score for k:11 is 0.7798221212458419 , 0.5394854920634922

Train AUC and CV AUC score for k:25 is 0.7187977793945829 , 0.5479113015873016

Train AUC and CV AUC score for k:51 is 0.6758321617041717 , 0.5589473015873015

Train AUC and CV AUC score for k:101 is 0.6485699793179571 , 0.5730077460317462
```

> Observation: we observe that for k=101 in kNN got the maximum AUC score.

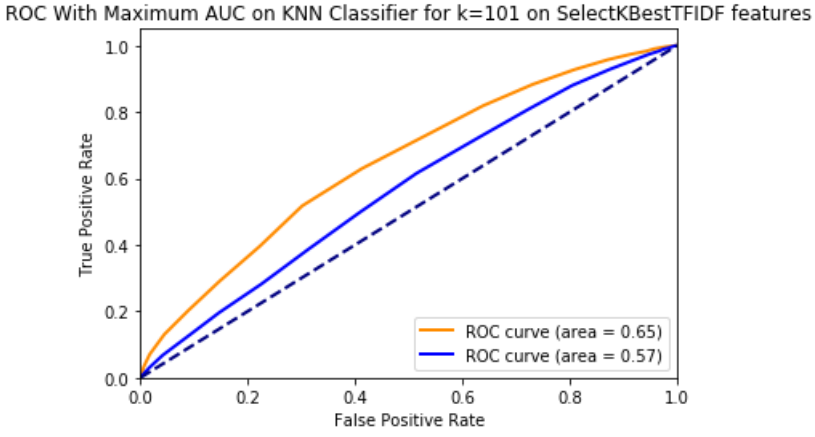
In [46]:

```
plotauc_tr_cv('SelectkBest TFIDF', [1,5,11,25,51,101], tr_score, cv_score)
```



In [48]:

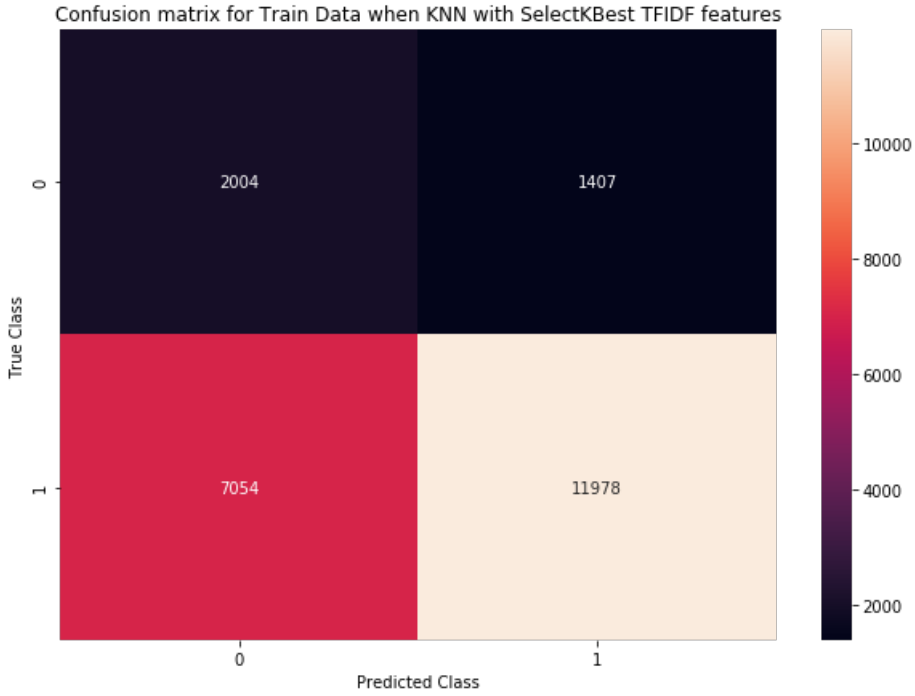
```
tr_thre, fpr, tpr, tr_predict, ts_predict = plotauc_tr_ts(101, 'SelectKBestTFIDF', tr_X_new, tr_y, ts_X_new, ts_y)
```

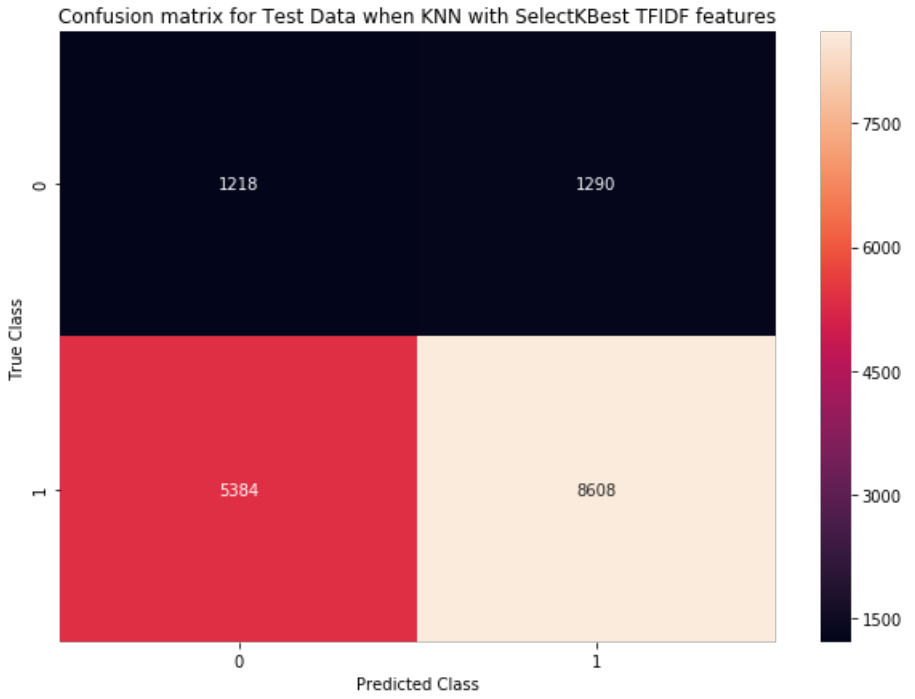


In [49]:

```
plot_cm('SelectKBest TFIDF', tr_thre, fpr, tpr, tr_y, tr_predict, ts_y, ts_predict)
```

```
the maximum value of tpr*(1-fpr) 0.36975655129554524 for threshold 0.871
Train confusion matrix
Test confusion matrix
```





3. Conclusions

In [0]:

```
# Please compare all your models using Prettytable library
```

In [50]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

x = PrettyTable()
```

```
x.field_names = ["Features", "Model", "Hyperparameter 'k'", "Maximum AUC score",]

x.add_row(["BoW","Brute", 101, 0.5886356214303946])
x.add_row(["TFIDF","Brute", 51, 0.5736983787094736])
x.add_row(["AvgW2V","Brute", 101, 0.6361381972743773])
x.add_row(["TDIDFW2V","Brute", 101, 0.6562650924482066])
x.add_row(["2000 SelectKbest from TFIDF","Brute",101,0.5730077460317462])

print(x)
```

```
+-----------------------------+-------+--------------------+--------------------+
|           Features          | Model | Hyperparameter 'k' | Maximum AUC score  |
+-----------------------------+-------+--------------------+--------------------+
|             BoW             | Brute |        101         | 0.5886356214303946 |
|            TFIDF            | Brute |         51         | 0.5736983787094736 |
|            AvgW2V           | Brute |        101         | 0.6361381972743773 |
|           TDIDFW2V          | Brute |        101         | 0.6562650924482066 |
| 2000 SelectKbest from TFIDF | Brute |        101         | 0.5730077460317462 |
+-----------------------------+-------+--------------------+--------------------+
```

In [ ]:

```
```