Donors choose data analysis

Table of contents

Little History about Data Set

Founded in 2000 by a high school teacher in the Bronx, DonorsChoose.org empowers public school teachers from across the country to request much-needed materials and experiences for their students. At any given time, there are thousands of classroom requests that can be brought to life with a gift of any amount.

Answers to What and Why Questions on Data Set

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`
	Grade level of students for which the project is targeted. One of the following enumerated values:

Feature	Description
project_grade_category	• Grades PreK-2 • Grades 3-5 • Grades 6-8 • Grades 9-12
project_subject_categories	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • Applied Learning • Care & Hunger • Health & Sports • History & Civics • Literacy & Language • Math & Science • Music & The Arts • Special Needs • Warmth **Examples:** • Music & The Arts • Literacy & Language, Math & Science
school_state	State where school is located (Two-letter U.S. postal code). **Example:** WY
project_subject_subcategories	One or more (comma-separated) subject subcategories for the project. **Examples:** • Literacy • Literature & Writing, Social Sciences
project_resource_summary	An explanation of the resources needed for the project. **Example:** • My students need hands on literacy materials to manage sensory needs!
project_essay_1	First application essay*
project_essay_2	Second application essay*
project_essay_3	Third application essay*
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_4:__ "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.

Importing required libraries

In [217]:

```
# numpy for easy numerical computations
import numpy as np
# pandas for dataframes and filterings
import pandas as pd
# sqlite3 library for performing operations on sqlite file
import sqlite3
# matplotlib for plotting graphs
import matplotlib.pyplot as plt
# seaborn library for easy plotting
import seaborn as sbrn
# warnings library for specific settings
import warnings
# regularlanguage for regex operations
import re
# For loading precomputed models
import pickle

# tqdm for tracking progress of loops
from tqdm import tqdm_notebook as tqdm
# For creating dictionary of words
from collections import Counter
# For creating BagOfWords Model
from sklearn.feature_extraction.text import CountVectorizer
# For creating TfidfModel
from sklearn.feature_extraction.text import TfidfVectorizer
# For standardizing values
from sklearn.preprocessing import StandardScaler
# For merging sparse matrices along row direction
```

```
from scipy.sparse import hstack
# For merging sparse matrices along column direction
from scipy.sparse import vstack
# For calculating TSNE values
from sklearn.manifold import TSNE
# For calculating the accuracy score on cross validate data
from sklearn.metrics import accuracy_score
# For performing the k-fold cross validation
from sklearn.model_selection import cross_val_score
# For splitting the data set into test and train data
from sklearn import cross_validation
# KNeighbors classifier for classification
from sklearn.neighbors import KNeighborsClassifier
# For creating samples for making dataset balanced
from sklearn.utils import resample
# For shuffling the dataframes
from sklearn.utils import shuffle
# For calculating roc_curve parameters
from sklearn.metrics import roc_curve
# For calculating auc value
from sklearn.metrics import auc
# For displaying results in table format
from prettytable import PrettyTable
# For generating confusion matrix
from sklearn.metrics import confusion_matrix
# For selecting most useful features
from sklearn.feature_selection import SelectKBest, f_classif

warnings.filterwarnings('ignore')
```

Reading and Storing Data

In [2]:

```
projectsData = pd.read_csv('train_data.csv');
resourcesData = pd.read_csv('resources.csv');
```

In [3]:

```
projectsData.head(3)
```

Out[3]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Gra

In [4]:

```
projectsData.tail(3)
```

Out[4]:

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime
109245	143653	p155633	cdbfd04aa041dc6739e9e576b1fb1478	Mrs.	NJ	2016-08-25 17:11:32
109246	164599	p206114	6d5675dbfafa1371f0e2f6f1b716fe2d	Mrs.	NY	2016-07-29 17:53:15
109247	128381	p191189	ca25d5573f2bd2660f7850a886395927	Ms.	VA	2016-06-29 09:17:01

In [5]:

```
resourcesData.head(3)
```

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
2	p069063	Cory Stories: A Kid's Book About Living With Adhd	1	8.45

In [6]:

```
resourcesData.tail(3)
```

Out[6]:

	id	description	quantity	price
1541269	p031981	Black Electrical Tape (GIANT 3 PACK) Each Roll...	6	8.99
1541270	p031981	Flormoon DC Motor Mini Electric Motor 0.5-3V 1...	2	8.14
1541271	p031981	WAYLLSHINE 6PCS 2 x 1.5V AAA Battery Spring Cl...	2	7.39

Helper functions and classes

In [7]:

```
def equalsBorder(numberOfEqualSigns):
    """
    This function prints passed number of equal signs
    """
    print("="* numberOfEqualSigns);
```

In [8]:

```
# Citation link: https://stackoverflow.com/questions/8924173/how-do-i-print-bold-text-in-python
class color:
   PURPLE = '\033[95m'
   CYAN = '\033[96m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
   GREEN = '\033[92m'
```

```
    YELLOW = '\033[93m'
    RED = '\033[91m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'
    END = '\033[0m'
```

In [9]:

```
def printStyle(text, style):
    "This function prints text with the style passed to it"
    print(style + text + color.END);
```

Shapes of projects data and resources data

In [10]:

```
printStyle("Number of data points in projects data: {}".format(projectsData.shape[0]), color.BOLD)
;
printStyle("Number of attributes in projects data:{}".format(projectsData.shape[1]), color.BOLD);
equalsBorder(60);
printStyle("Number of data points in resources data: {}".format(resourcesData.shape[0]),
color.BOLD);
printStyle("Number of attributes in resources data: {}".format(resourcesData.shape[1]), color.BOLD)
;
```

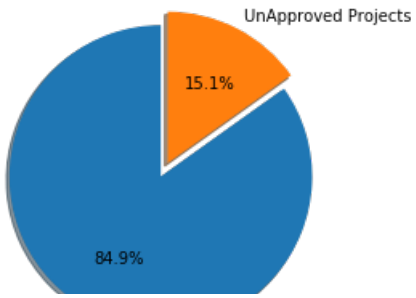
```
Number of data points in projects data: 109248
Number of attributes in projects data:17
============================================================
Number of data points in resources data: 1541272
Number of attributes in resources data: 4
```

Univariate data analysis

In [11]:

```
approvedProjects = projectsData[projectsData.project_is_approved == 1].shape[0];
unApprovedProjects = projectsData[projectsData.project_is_approved == 0].shape[0];
totalProjects = projectsData.shape[0];
print("Number of projects approved for funding: {}, ({})".format(approvedProjects,
(approvedProjects / totalProjects) * 100));
print("Number of projects not approved for funding: {}, ({})".format(unApprovedProjects, (unApprov
edProjects / totalProjects) * 100));
# Pie chart representation
# Citation: https://matplotlib.org/gallery/pie_and_polar_charts/pie_features.html
labels = ["Approved Projects", "UnApproved Projects"];
explode = (0, 0.1);
sizes = [approvedProjects, unApprovedProjects];
figure, ax = plt.subplots();
ax.pie(sizes, labels = labels, explode = explode, autopct = '%1.1f%%', shadow = True, startangle =
90);
ax.axis('equal');
plt.rcParams['figure.figsize'] = (7, 7);
plt.show();
```

```
Number of projects approved for funding: 92706, (84.85830404217927)
Number of projects not approved for funding: 16542, (15.141695957820739)
```



Observation:

1. There are more number of approved projects compared to rejected projects. So this is a imbalanced dataset.

Univariate Analysis : 'school_state'

Project proposal percentage in different states

In [12]:

```
groupedByStatesData = pd.DataFrame(projectsData.groupby(['school_state'])['project_is_approved'].a
pply(np.mean)).reset_index();
groupedByStatesData.columns = ['state_code', 'number_of_proposals'];
groupedByStatesData = groupedByStatesData.sort_values(by=['number_of_proposals'], ascending = True)
;
printStyle("5 States with lowest percentage of project approvals:", color.BOLD);
equalsBorder(60);
groupedByStatesData.head(5)
```

```
5 States with lowest percentage of project approvals:
============================================================
```

Out[12]:

	state_code	number_of_proposals
46	VT	0.800000
7	DC	0.802326
43	TX	0.813142
26	MT	0.816327
18	LA	0.831245

In [13]:

```
printStyle("5 states with highest percentage of project approvals: ", color.BOLD);
equalsBorder(60);
groupedByStatesData.tail(5).iloc[::-1]
```

```
5 states with highest percentage of project approvals:
============================================================
```

Out[13]:

	state_code	number_of_proposals
8	DE	0.897959
28	ND	0.888112
47	WA	0.876178
35	OH	0.875152
30	NH	0.873563

In [14]:

```
def univariateBarPlots(data, col1, col2 = 'project_is_approved', orientation = 'vertical', plot = T
rue):
    groupedData = data.groupby(col1);
    # Count number of zeros in dataframe python: https://stackoverflow.com/a/51540521/4084039
```

```
    tempData = pd.DataFrame(groupedData[col2].agg(lambda x: x.eq(1).sum())).reset_index();
    tempData['total'] = pd.DataFrame(groupedData[col2].agg({'total': 'count'})).reset_index()['tota
l'];
    tempData['approval_rate'] = pd.DataFrame(groupedData[col2].agg({'approval_rate': 'mean'})).rese
t_index()['approval_rate'];
    tempData.sort_values(by=['total'], inplace = True, ascending = False);
    tempDataWithTotalAndCol2 = tempData[['total', col2, col1]]
    if plot:
        if(orientation == 'vertical'):
            tempDataWithTotalAndCol2.plot(x = col1, align= 'center', kind = 'bar', title = "Number
of projects approved vs rejected", figsize = (20, 6), stacked = True, rot = 0);
        else:
            tempDataWithTotalAndCol2.plot(x = col1, align= 'center', kind = 'barh', title = "Number
of projects approved vs rejected", width = 0.8, figsize = (23, 20), stacked = True);
    return tempData;
```

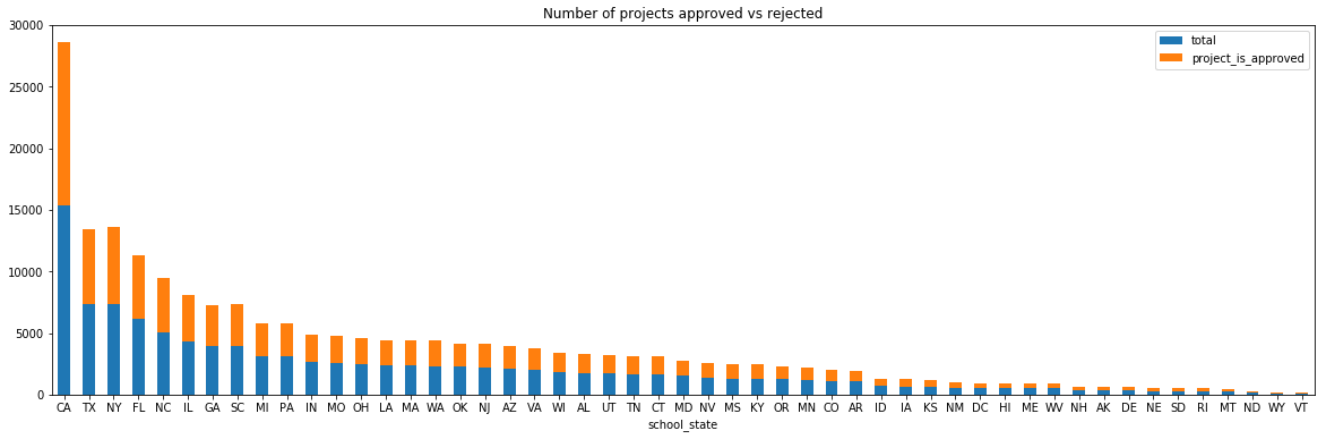
In [15]:

```
statesCharacteristicsData = univariateBarPlots(projectsData, 'school_state', 'project_is_approved'
, orientation = 'vertical');
printStyle("Top 5 states with high project proposals", color.BOLD)
equalsBorder(60);
statesCharacteristicsData.head(5)
```

```
Top 5 states with high project proposals
============================================================
```

Out[15]:

	school_state	project_is_approved	total	approval_rate
4	CA	13205	15388	0.858136
43	TX	6014	7396	0.813142
34	NY	6291	7318	0.859661
9	FL	5144	6185	0.831690
27	NC	4353	5091	0.855038



In [16]:

```
printStyle("Top 5 states with least project proposals", color.BOLD)
equalsBorder(60);
statesCharacteristicsData.tail(5)
```

```
Top 5 states with least project proposals
============================================================
```

Out[16]:

	school_state	project_is_approved	total	approval_rate
39	RI	243	285	0.852632

	school_state	project_is_approved	total	approval_rate
26	MT	200	245	0.816327
28	ND	127	143	0.888112
50	WY	82	98	0.836735
46	VT	64	80	0.800000

Observation:

1. Highest number of project proposals are from CA(California) and it was almost about 16000 projects
2. Every state has more than 80% approval rate.

Univariate Analysis: teacher_prefix

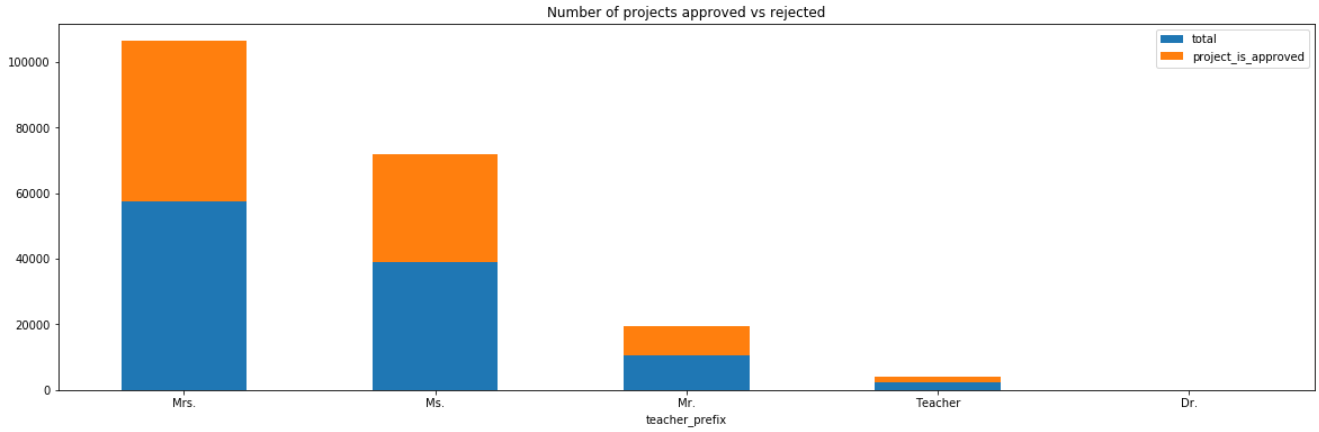
In [17]:

```
teacherPrefixCharacteristicsData = univariateBarPlots(projectsData, 'teacher_prefix',
'project_is_approved', orientation = 'vertical', plot = True);
printStyle("Project proposals characteristics based on types of persons", color.BOLD);
equalsBorder(60);
teacherPrefixCharacteristicsData
```

```
Project proposals characteristics based on types of persons
============================================================
```

Out[17]:

	teacher_prefix	project_is_approved	total	approval_rate
2	Mrs.	48997	57269	0.855559
3	Ms.	32860	38955	0.843537
1	Mr.	8960	10648	0.841473
4	Teacher	1877	2360	0.795339
0	Dr.	9	13	0.692308



Observataion:

1. When compared to others Dr.'s have proposed very less number of projects.
2. Women have proposed more number of projects than men.

Univariate Analysis: project_grade

In [18]:

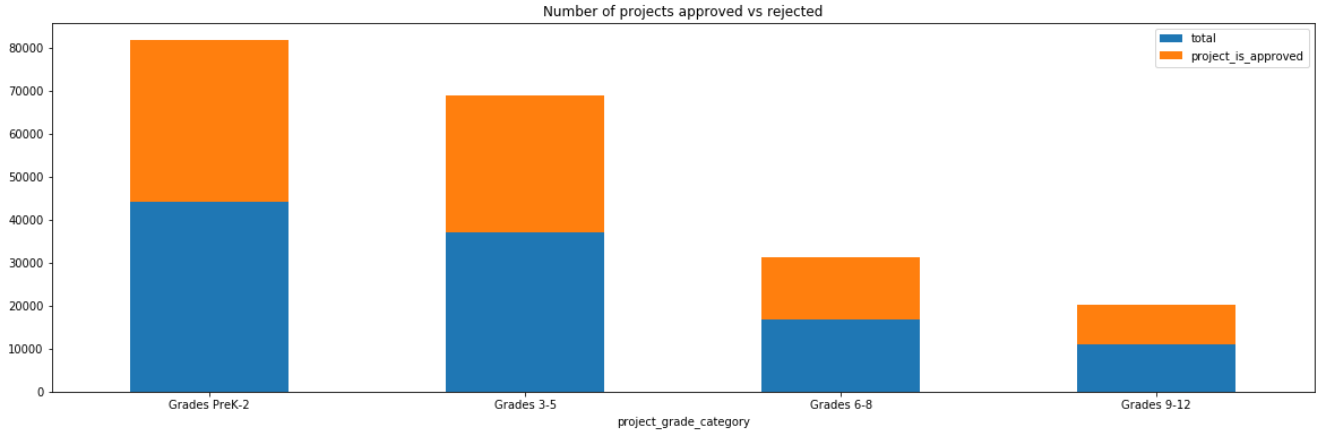
```
gradeCharacteristicsData = univariateBarPlots(projectsData, 'project_grade_category',
```

```
'project_is_approved', orientation = 'vertical', plot = True);
printStyle("Project proposal characteristics based on grades", color.BOLD);
equalsBorder(60);
gradeCharacteristicsData
```

```
Project proposal characteristics based on grades
============================================================
```

Out[18]:

	project_grade_category	project_is_approved	total	approval_rate
3	Grades PreK-2	37536	44225	0.848751
0	Grades 3-5	31729	37137	0.854377
1	Grades 6-8	14258	16923	0.842522
2	Grades 9-12	9183	10963	0.837636



Observation:

1. Most number of projects proposed are for students less than grade-5 (for primary school students) which means that children are being taught with project oriented teaching which is great.

Univariate Analysis: project_subject_categories

In [19]:

```
# 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
def cleanCategories(subjectCategories):
    cleanedCategories = []
    for subjectCategory in tqdm(subjectCategories):
        tempCategory = ""
        for category in subjectCategory.split(","):
            if 'The' in category.split(): # this will split each of the catogory based on space "Ma
th & Science"=> "Math","&", "Science"
                category = category.replace('The','') # if we have the words "The" we are going to
replace it with ''(i.e removing 'The')
            category = category.replace(' ','') # we are placeing all the ' '(space) with ''(empty)
ex:"Math & Science"=>"Math&Science"
            tempCategory += category.strip()+" "#" abc ".strip() will return "abc", remove the
trailing spaces
            tempCategory = tempCategory.replace('&','_')
        cleanedCategories.append(tempCategory)
    return cleanedCategories
```

In [20]:

```
# projectDataWithCleanedCategories = pd.DataFrame(projectsData);
subjectCategories = list(projectsData.project_subject_categories);
cleanedCategories = cleanCategories(subjectCategories);
printStyle("Sample categories: ", color.BOLD);
equalsBorder(60);
print(subjectCategories[0:5]);
equalsBorder(60);
printStyle("Sample cleaned categories: ", color.BOLD);
equalsBorder(60);
print(cleanedCategories[0:5]);
projectsData['cleaned_categories'] = cleanedCategories;
projectsData.head(5)
```

```
Sample categories:
============================================================
['Literacy & Language', 'History & Civics, Health & Sports', 'Health & Sports', 'Literacy & Langua
ge, Math & Science', 'Math & Science']
============================================================
Sample cleaned categories:
============================================================
['Literacy_Language ', 'History_Civics Health_Sports ', 'Health_Sports ', 'Literacy_Language
Math_Science ', 'Math_Science ']
```

Out[20]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Gra
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	2016-10-06 21:16:17	Gra
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	2016-07-11 01:10:09	Gra

In [21]:

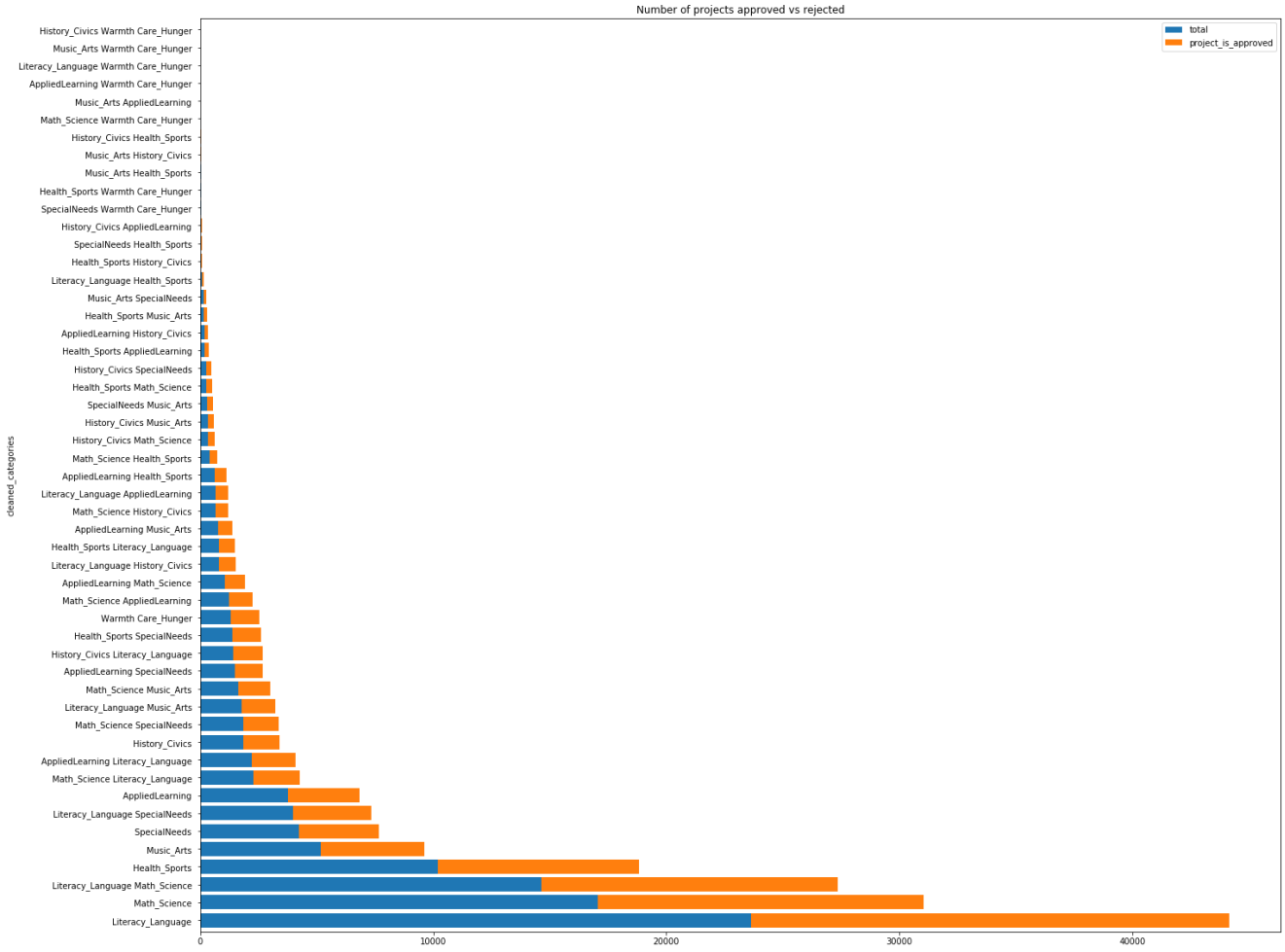
```
categoriesCharacteristicsData = univariateBarPlots(projectsData, 'cleaned_categories',
'project_is_approved', orientation = 'horizontal', plot = True);
print("Project proposals characteristics based on subject categories");
equalsBorder(60);
categoriesCharacteristicsData.head(5)
```

```
Project proposals characteristics based on subject categories
============================================================
```

Out[21]:

	cleaned_categories	project_is_approved	total	approval_rate

	cleaned_categories	project_is_approved	total	approval_rate
24	Literacy_Language	20520	23655	0.867470
32	Math_Science	13991	17072	0.819529
28	Literacy_Language Math_Science	12725	14636	0.869432
8	Health_Sports	8640	10177	0.848973
40	Music_Arts	4429	5180	0.855019



In [22]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
categoriesCounter = Counter()
for subjectCategory in projectsData.cleaned_categories.values:
    categoriesCounter.update(subjectCategory.split());
categoriesCounter
```

Out[22]:

```
Counter({'Literacy_Language': 52239,
         'History_Civics': 5914,
         'Health_Sports': 14223,
         'Math_Science': 41421,
         'SpecialNeeds': 13642,
         'AppliedLearning': 12135,
         'Music_Arts': 10293,
         'Warmth': 1388,
         'Care_Hunger': 1388})
```

In [23]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
categoriesDictionary = dict(categoriesCounter);
sortedCategoriesDictionary = dict(sorted(categoriesDictionary.items(), key = lambda keyValue: keyVa
lue[1]));
sortedCategoriesData = pd.DataFrame.from_dict(sortedCategoriesDictionary, columns =
['subject_categories'], orient = 'index');
```

```
printStyle("Number of projects by Subject Categories: ", color.BOLD);
equalsBorder(60);
sortedCategoriesData
```

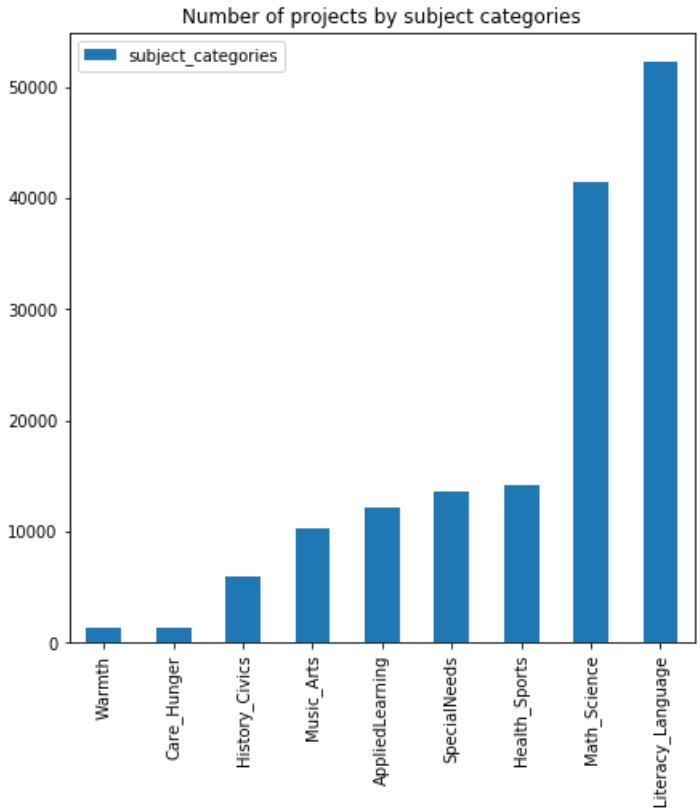
```
Number of projects by Subject Categories:
============================================================
```

Out[23]:

	subject_categories
Warmth	1388
Care_Hunger	1388
History_Civics	5914
Music_Arts	10293
AppliedLearning	12135
SpecialNeeds	13642
Health_Sports	14223
Math_Science	41421
Literacy_Language	52239

In [24]:

```
sortedCategoriesData.plot(kind = 'bar', title = 'Number of projects by subject categories');
```



Observation:

1. Many number of projects proposed belong to multiple subject categories.
2. When compared to others literacy_language & math_science have large number of project proposals.

Univariate Analysis: project_subject_subcategories

In [25]:

```
subjectSubCategories = projectsData.project_subject_subcategories;
cleanedSubCategories = cleanCategories(subjectSubCategories);
printStyle("Sample subject sub categories: ", color.BOLD);
equalsBorder(70);
print(subjectSubCategories[0:5]);
equalsBorder(70);
printStyle("Sample cleaned subject sub categories: ", color.BOLD);
equalsBorder(70);
print(cleanedSubCategories[0:5]);
projectsData['cleaned_sub_categories'] = cleanedSubCategories;
```

```
Sample subject sub categories:
======================================================================
0                       ESL, Literacy
1    Civics & Government, Team Sports
2      Health & Wellness, Team Sports
3               Literacy, Mathematics
4                        Mathematics
Name: project_subject_subcategories, dtype: object
======================================================================
Sample cleaned subject sub categories:
======================================================================
['ESL Literacy ', 'Civics_Government TeamSports ', 'Health_Wellness TeamSports ', 'Literacy
Mathematics ', 'Mathematics ']
```

In [26]:

```
projectsData.head(5)
```

Out[26]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Gra
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	2016-10-06 21:16:17	Gra
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	2016-07-11 01:10:09	Gra

In [27]:

```
subCategoriesCharacteristicsData = univariateBarPlots(projectsData, 'cleaned_sub_categories',
'project_is_approved', plot = False);
print("Project proposals characteristics based on subject sub categories");
equalsBorder(60);
subCategoriesCharacteristicsData.head(5)
```

```
Project proposals characteristics based on subject sub categories
============================================================
```

Out[27]:

	cleaned_sub_categories	project_is_approved	total	approval_rate
317	Literacy	8371	9486	0.882458
319	Literacy Mathematics	7260	8325	0.872072
331	Literature_Writing Mathematics	5140	5923	0.867803
318	Literacy Literature_Writing	4823	5571	0.865733
342	Mathematics	4385	5379	0.815207

In [28]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
subjectsSubCategoriesCounter = Counter();
for subCategory in projectsData.cleaned_sub_categories:
    subjectsSubCategoriesCounter.update(subCategory.split());
subjectsSubCategoriesCounter
```

Out[28]:

```
Counter({'ESL': 4367,
         'Literacy': 33700,
         'Civics_Government': 815,
         'TeamSports': 2192,
         'Health_Wellness': 10234,
         'Mathematics': 28074,
         'Literature_Writing': 22179,
         'SpecialNeeds': 13642,
         'ParentInvolvement': 677,
         'EnvironmentalScience': 5591,
         'Health_LifeScience': 4235,
         'AppliedSciences': 10816,
         'EarlyDevelopment': 4254,
         'Music': 3145,
         'ForeignLanguages': 890,
         'Other': 2372,
         'Economics': 269,
         'FinancialLiteracy': 568,
         'Gym_Fitness': 4509,
         'VisualArts': 6278,
         'Warmth': 1388,
         'Care_Hunger': 1388,
         'SocialSciences': 1920,
         'College_CareerPrep': 2568,
         'CharacterEducation': 2065,
         'PerformingArts': 1961,
         'CommunityService': 441,
         'History_Geography': 3171,
         'NutritionEducation': 1355,
         'Extracurricular': 810})
```

In [29]:

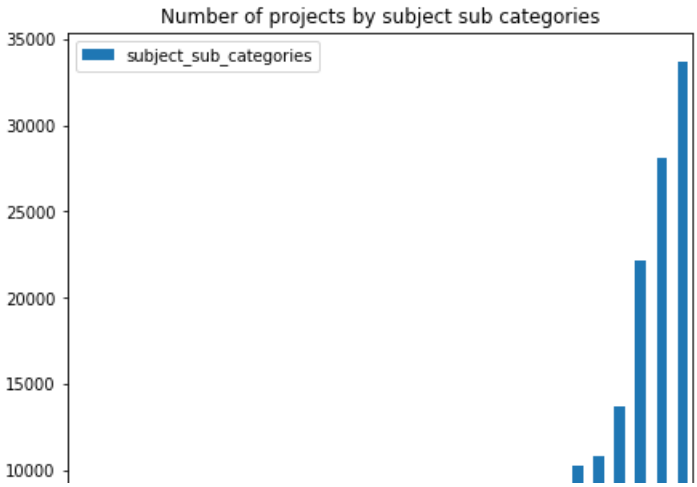
```
# dict sort by value python: https://stackoverflow.com/a/613218/4084039
dictionarySubCategories = dict(subjectsSubCategoriesCounter);
sortedDictionarySubCategories = dict(sorted(dictionarySubCategories.items(), key = lambda keyValue:
keyValue[1]));
sortedSubCategoriesData = pd.DataFrame.from_dict(sortedDictionarySubCategories, columns =
['subject_sub_categories'], orient = 'index');
sortedSubCategoriesData.plot(kind = 'bar', title = "Number of projects by subject sub categories")
;
printStyle("Number of projects sorted by subject sub categories: ", color.BOLD);
equalsBorder(70);
sortedSubCategoriesData
```

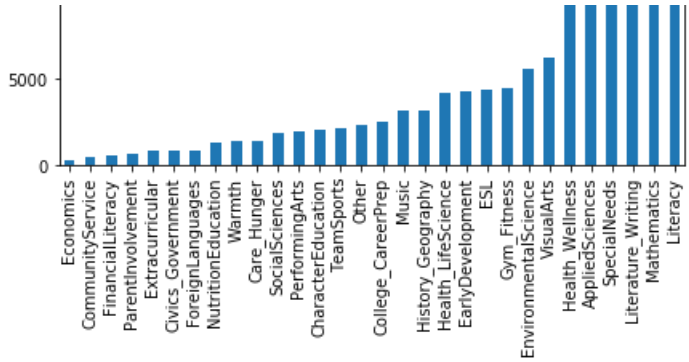
```
Number of projects sorted by subject sub categories:
```

==========

Out[29]:

	subject_sub_categories
Economics	269
CommunityService	441
FinancialLiteracy	568
ParentInvolvement	677
Extracurricular	810
Civics_Government	815
ForeignLanguages	890
NutritionEducation	1355
Warmth	1388
Care_Hunger	1388
SocialSciences	1920
PerformingArts	1961
CharacterEducation	2065
TeamSports	2192
Other	2372
College_CareerPrep	2568
Music	3145
History_Geography	3171
Health_LifeScience	4235
EarlyDevelopment	4254
ESL	4367
Gym_Fitness	4509
EnvironmentalScience	5591
VisualArts	6278
Health_Wellness	10234
AppliedSciences	10816
SpecialNeeds	13642
Literature_Writing	22179
Mathematics	28074
Literacy	33700





Observation:

1. There are more number of subject subcategories than subject categories.
2. Even more number of projects proposed belong to multiple subject sub categories.

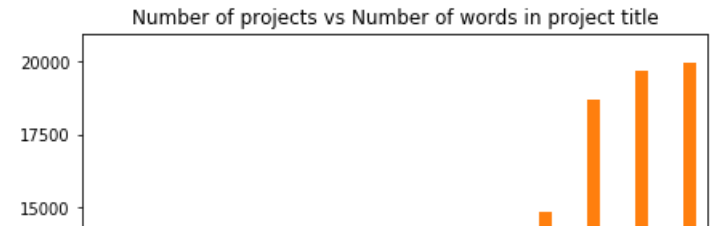
Univariate Analysis : project_title

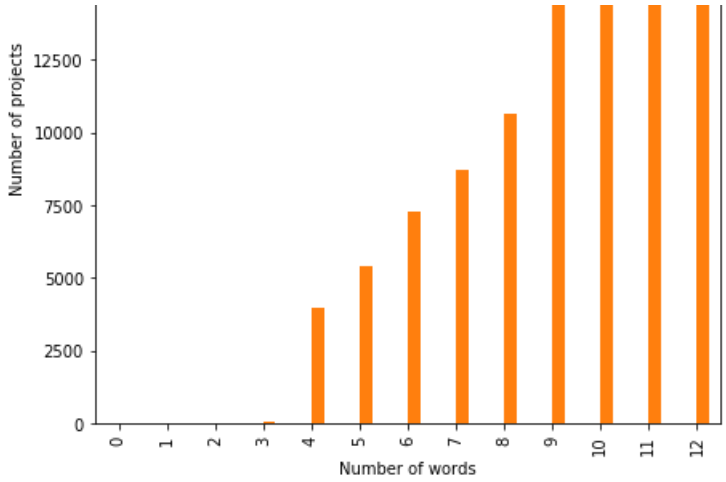
In [30]:

```
#How to calculate number of words in a string in DataFrame:
https://stackoverflow.com/a/37483537/4084039
wordCounts = projectsData['project_title'].str.split().apply(len).value_counts();
dictionaryWordCounts = dict(wordCounts);
dictionaryWordCounts = dict(sorted(dictionaryWordCounts.items(), key = lambda kv: kv[1]));
wordCountsData = pd.DataFrame.from_dict({'number_of_words': list(dictionaryWordCounts.keys()), 'nu
mber_of_projects': list(dictionaryWordCounts.values())}).sort_values(by = ['number_of_projects']);
wordCountsData.plot(kind = 'bar', title = "Number of projects vs Number of words in project title"
, legend = False);
plt.xlabel('Number of words');
plt.ylabel('Number of projects');
wordCountsData
```

Out[30]:

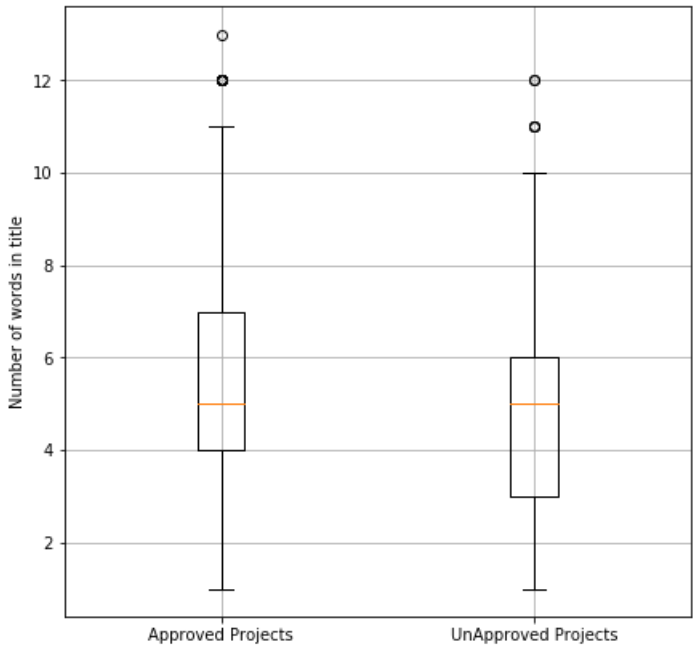
	number_of_words	number_of_projects
0	13	1
1	12	11
2	11	30
3	1	31
4	10	3968
5	9	5383
6	8	7289
7	2	8733
8	7	10631
9	6	14824
10	3	18691
11	5	19677
12	4	19979





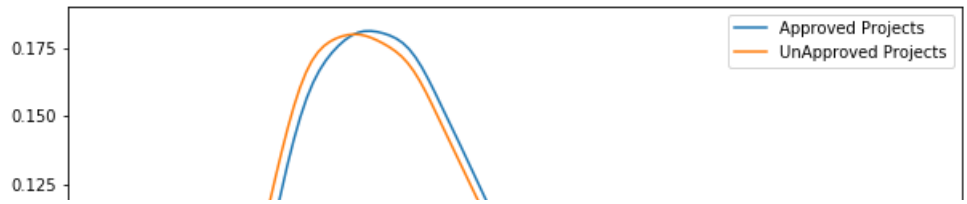
In [31]:

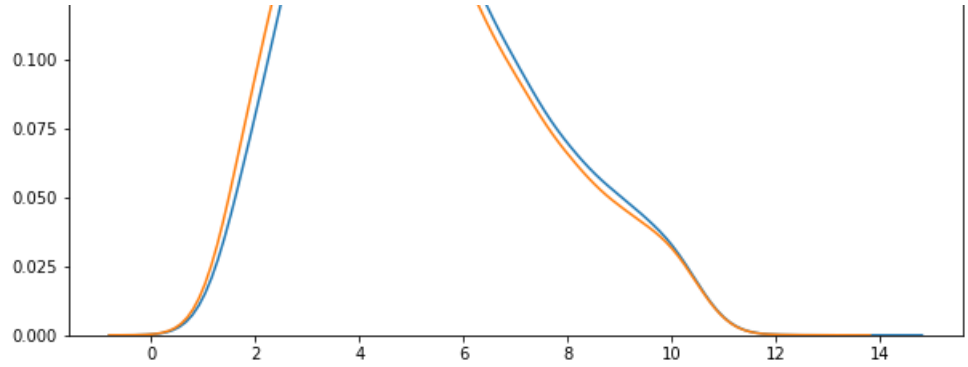
```
approvedNumberOfProjects = projectsData[projectsData.project_is_approved == 1]['project_title'].str
.split().apply(len);
approvedNumberOfProjects = approvedNumberOfProjects.values
unApprovedNumberOfProjects = projectsData[projectsData.project_is_approved == 0]['project_title'].s
tr.split().apply(len);
unApprovedNumberOfProjects = unApprovedNumberOfProjects.values
plt.boxplot([approvedNumberOfProjects, unApprovedNumberOfProjects]);
plt.grid();
plt.xticks([1, 2], ['Approved Projects', 'UnApproved Projects']);
plt.ylabel('Number of words in title');
plt.show();
```



In [32]:

```
plt.figure(figsize = (10, 6));
sbrn.kdeplot(approvedNumberOfProjects, label = "Approved Projects", bw = 0.6);
sbrn.kdeplot(unApprovedNumberOfProjects, label = "UnApproved Projects", bw = 0.6);
plt.legend();
plt.show();
```





Observations:

1. Most of the approved projects have between 4 to 8 number of words in their project_title.
2. Most of the rejected projects have between 3 to 6 number of words in their project_title.

Univariate Analysis: project_essay_1,2,3,4

In [33]:

```
projectsData['project_essay'] = projectsData['project_essay_1'].map(str) + projectsData['project_es
say_2'].map(str) + \
                                projectsData['project_essay_3'].map(str) + projectsData['project_es
ay_4'].map(str);
projectsData.head(5)
```

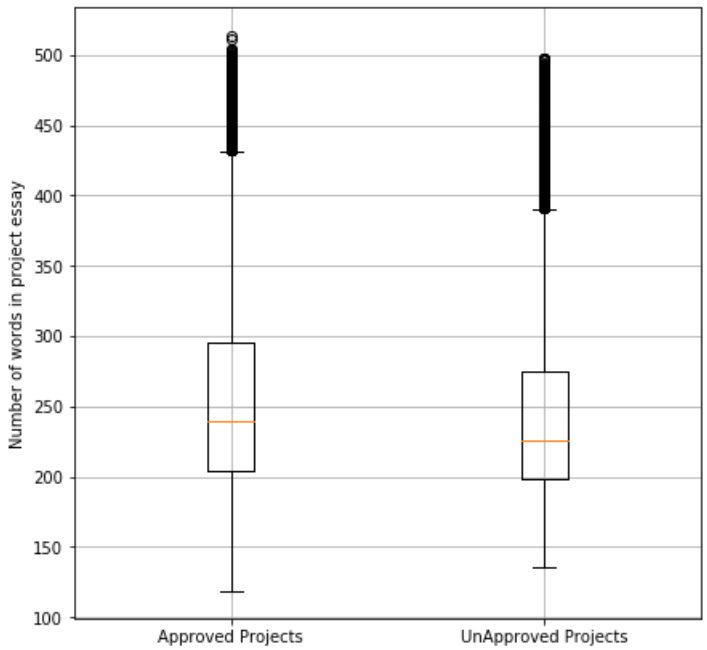
Out[33]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Gra
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	2016-10-06 21:16:17	Gra
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	2016-07-11 01:10:09	Gra

In [34]:

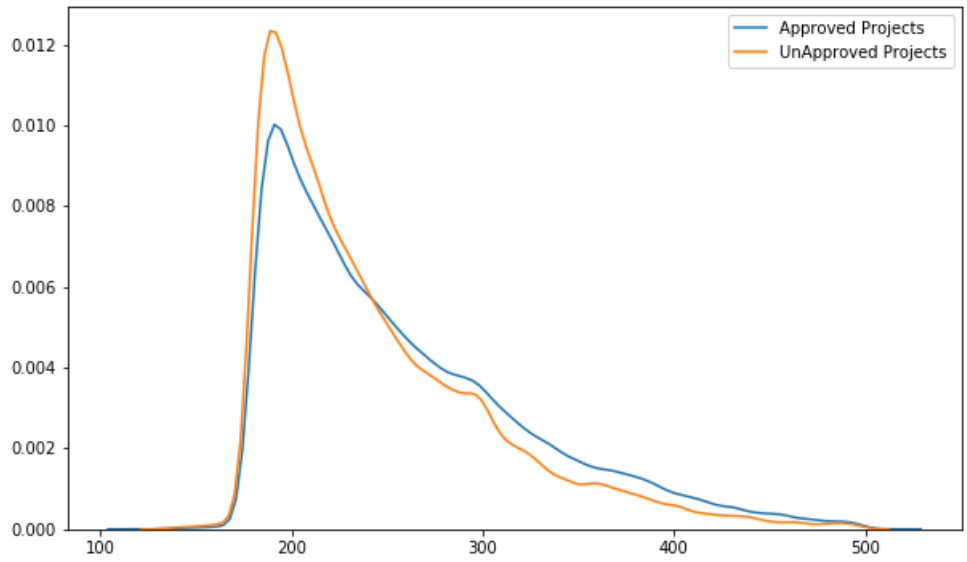
```
approvedNumberOfProjects = projectsData[projectsData.project_is_approved == 1]['project_essay'].str
.split().apply(len);
approvedNumberOfProjects = approvedNumberOfProjects.values
unApprovedNumberOfProjects = projectsData[projectsData.project_is_approved == 0]['project_essay'].s
```

```
tr.split().apply(len);
unApprovedNumberOfProjects = unApprovedNumberOfProjects.values
plt.boxplot([approvedNumberOfProjects, unApprovedNumberOfProjects]);
plt.grid();
plt.xticks([1, 2], ['Approved Projects', 'UnApproved Projects']);
plt.ylabel('Number of words in project essay');
plt.show();
```



In [35]:

```
plt.figure(figsize = (10, 6));
sbrn.kdeplot(approvedNumberOfProjects, label = "Approved Projects", bw = 5);
sbrn.kdeplot(unApprovedNumberOfProjects, label = "UnApproved Projects", bw = 5);
plt.legend();
plt.show();
```



Observation:

1. The approved and rejected projects overlap largely when plotted based on number of words in project_essay. So we cannot predict any observation which will be useful for classification.

Univariate Analysis: price

In [36]:

```
projectsData.head(5)
```

Out[36]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Gra
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	2016-10-06 21:16:17	Gra
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	2016-07-11 01:10:09	Gra

In [37]:

```
resourcesData.head(5)
```

Out[37]:

	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
2	p069063	Cory Stories: A Kid's Book About Living With Adhd	1	8.45
3	p069063	Dixon Ticonderoga Wood-Cased #2 HB Pencils, Bo...	2	13.59
4	p069063	EDUCATIONAL INSIGHTS FLUORESCENT LIGHT FILTERS...	3	24.95

In [38]:

```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in-one-step
priceAndQuantityData = resourcesData.groupby('id').agg({'price': 'sum', 'quantity':
'sum'}).reset_index();
priceAndQuantityData.head(5)
```

Out[38]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

	id	price	quantity
2	p000003	298.97	4
3	p000004	1113.69	98
4	p000005	485.99	8

In [39]:

```
projectsData.shape
```

Out[39]:

```
(109248, 20)
```

In [40]:

```
projectsData = pd.merge(projectsData, priceAndQuantityData, on = 'id', how = 'left');
print(projectsData.shape);
projectsData.head(3)
```

```
(109248, 22)
```

Out[40]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Gra

3 rows × 22 columns

In [41]:

```
projectsData[projectsData['id'] == 'p253737']
```

Out[41]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	proje
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grade

1 rows × 22 columns

In [42]:

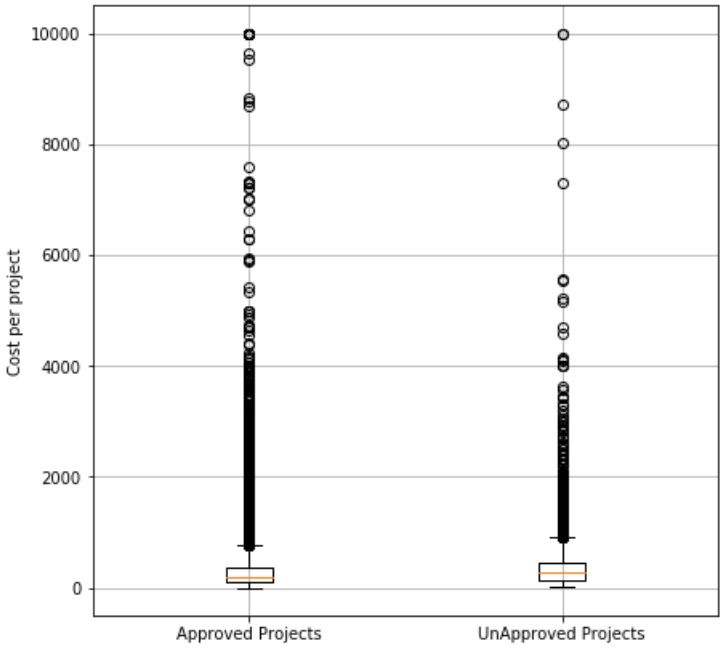
```
priceAndQuantityData[priceAndQuantityData['id'] == 'p253737']
```

Out[42]:

	id	price	quantity
253736	p253737	154.6	23

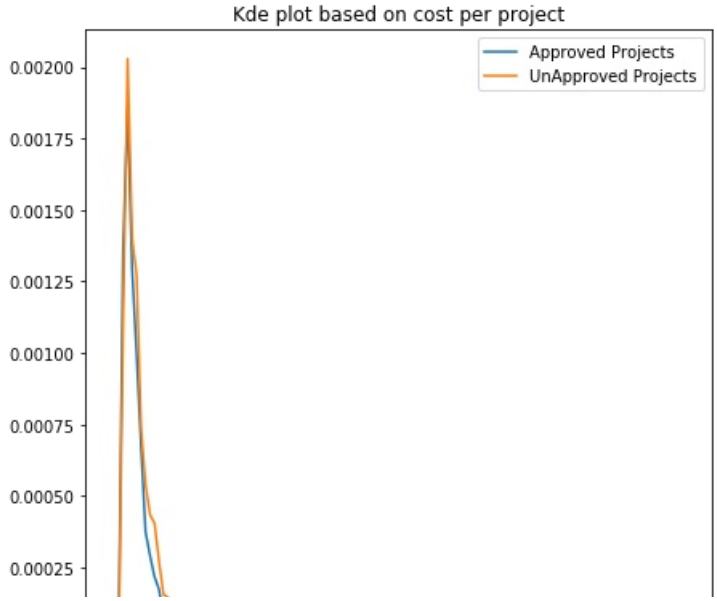
In [43]:

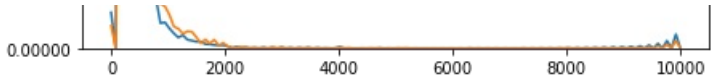
```
approvedProjectsPrice = projectsData[projectsData['project_is_approved'] == 1].price;
unApprovedProjectsPrice = projectsData[projectsData['project_is_approved'] == 0].price;
plt.boxplot([approvedProjectsPrice, unApprovedProjectsPrice]);
plt.grid();
plt.xticks([1, 2], ['Approved Projects', 'UnApproved Projects']);
plt.ylabel('Cost per project');
plt.show();
```



In [44]:

```
plt.title("Kde plot based on cost per project");
sbrn.kdeplot(approvedProjectsPrice, label = "Approved Projects", bw = 0.6);
sbrn.kdeplot(unApprovedProjectsPrice, label = "UnApproved Projects", bw = 0.6);
plt.legend();
plt.show();
```





In [45]:

```
pricePercentilesApproved = [round(np.percentile(approvedProjectsPrice, percentile), 3) for percenti
le in np.arange(0, 100, 5)];
pricePercentilesUnApproved = [round(np.percentile(unApprovedProjectsPrice, percentile), 3) for perc
entile in np.arange(0, 100, 5)];
percentileValuePricesData = pd.DataFrame({'Percentile': np.arange(0, 100, 5), 'Approved projects':
pricePercentilesApproved, 'UnApproved Projects': pricePercentilesUnApproved});
percentileValuePricesData
```

Out[45]:

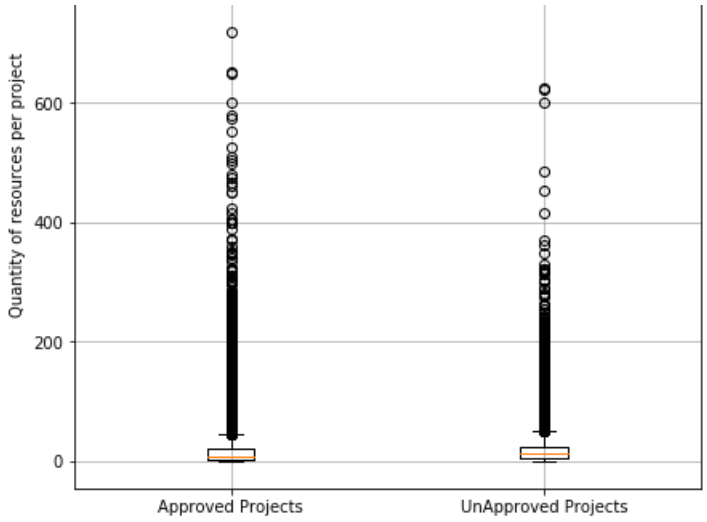
	Percentile	Approved projects	UnApproved Projects
0	0	0.660	1.970
1	5	13.590	41.900
2	10	33.880	73.670
3	15	58.000	99.109
4	20	77.380	118.560
5	25	99.950	140.892
6	30	116.680	162.230
7	35	137.232	184.014
8	40	157.000	208.632
9	45	178.265	235.106
10	50	198.990	263.145
11	55	223.990	292.610
12	60	255.630	325.144
13	65	285.412	362.390
14	70	321.225	399.990
15	75	366.075	449.945
16	80	411.670	519.282
17	85	479.000	618.276
18	90	593.110	739.356
19	95	801.598	992.486

Observation:

1. Most of the projects proposed are of less cost.

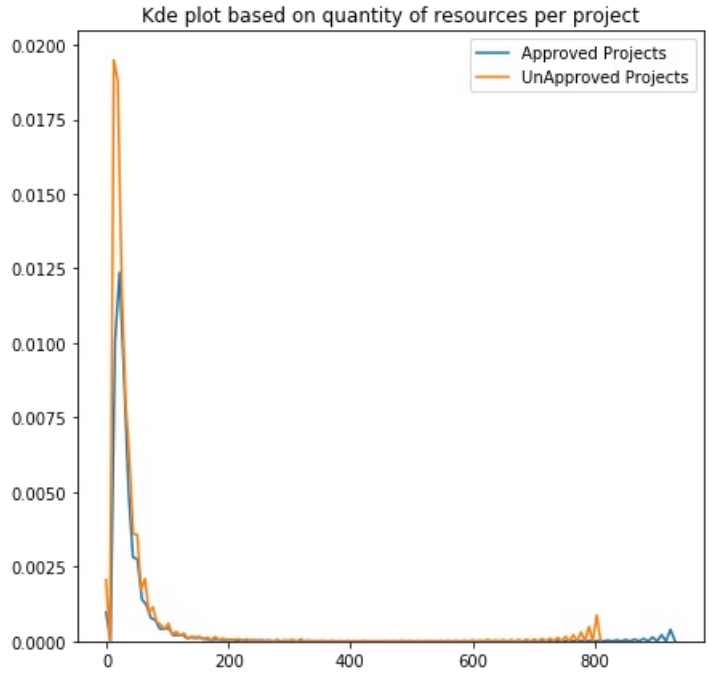
In [46]:

```
approvedProjectsQuantity = projectsData[projectsData['project_is_approved'] == 1].quantity;
unApprovedProjectsQuantity = projectsData[projectsData['project_is_approved'] == 0].quantity;
plt.boxplot([approvedProjectsQuantity, unApprovedProjectsQuantity]);
plt.grid();
plt.xticks([1, 2], ['Approved Projects', 'UnApproved Projects']);
plt.ylabel('Quantity of resources per project');
plt.show();
```



In [47]:

```
plt.title("Kde plot based on quantity of resources per project");
sbrn.kdeplot(approvedProjectsQuantity, label = "Approved Projects", bw = 0.6);
sbrn.kdeplot(unApprovedProjectsQuantity, label = "UnApproved Projects", bw = 0.6);
plt.legend();
plt.show();
```



In [48]:

```
quantityPercentilesApproved = [round(np.percentile(approvedProjectsQuantity, percentile), 3) for pe
rcentile in np.arange(0, 100, 5)];
quantityPercentilesUnApproved = [round(np.percentile(unApprovedProjectsQuantity, percentile), 3) fo
r percentile in np.arange(0, 100, 5)];
percentileValueQuantitiesData = pd.DataFrame({'Percentile': np.arange(0, 100, 5), 'Approved project
s': quantityPercentilesApproved, 'UnApproved Projects': quantityPercentilesUnApproved});
percentileValueQuantitiesData
```

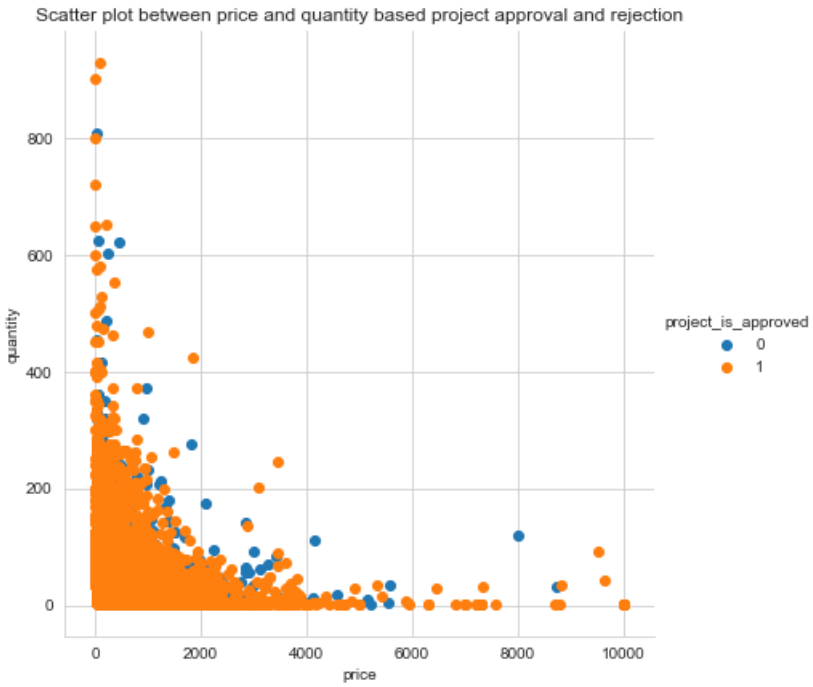
Out[48]:

	Percentile	Approved projects	UnApproved Projects
0	0	1.0	1.0
1	5	1.0	2.0
2	10	1.0	3.0

	Percentile	Approved projects	UnApproved Projects
3	15	2.0	4.0
4	20	3.0	5.0
5	25	3.0	6.0
6	30	4.0	7.0
7	35	5.0	8.0
8	40	6.0	9.0
9	45	7.0	10.0
10	50	8.0	12.0
11	55	10.0	13.0
12	60	11.0	15.0
13	65	14.0	18.0
14	70	16.0	20.0
15	75	20.0	24.0
16	80	25.0	29.0
17	85	30.0	35.0
18	90	38.0	45.0
19	95	56.0	63.0

In [49]:

```
sbrn.set_style('whitegrid');
sbrn.FacetGrid(projectsData, hue = 'project_is_approved', size = 6) \
    .map(plt.scatter, 'price', 'quantity') \
    .add_legend();
plt.title("Scatter plot between price and quantity based project approval and rejection");
plt.show();
```



Observation:

1. When plotted scatter plot between approved and rejected projects based on price and quantity there is huge overlap. So the projects approval is not actually depending on price and quantity resources of the project.

Univariate Analysis: teacher_number_of_previously_posted_projects

In [50]:

```
projectsData.head(5)
```

Out[50]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Gra
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Gra
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs.	KY	2016-10-06 21:16:17	Gra
4	172407	p104768	be1f7507a41f8479dc06f047086a39ec	Mrs.	TX	2016-07-11 01:10:09	Gra

5 rows × 22 columns

In [51]:

```
previouslyPostedApprovedNumberData =
projectsData.groupby('teacher_number_of_previously_posted_projects')['project_is_approved'].agg(la
mbda x: x.eq(1).sum()).reset_index();
previouslyPostedRejectedNumberData =
projectsData.groupby('teacher_number_of_previously_posted_projects')['project_is_approved'].agg(la
mbda x: x.eq(0).sum()).reset_index();
print("Total number of projects approved: ", len(projectsData[projectsData['project_is_approved']
== 1]));
print("Total number of projects rejected: ", len(projectsData[projectsData['project_is_approved']
== 0]));
print("Number of projects approved categorized by previously_posted: ",
previouslyPostedApprovedNumberData['project_is_approved'].sum());
print("Number of projects rejected categorized by previously_posted: ",
previouslyPostedRejectedNumberData['project_is_approved'].sum());
previouslyPostedNumberData = pd.merge(previouslyPostedApprovedNumberData,
previouslyPostedRejectedNumberData, on = 'teacher_number_of_previously_posted_projects', how =
'inner');
previouslyPostedNumberData.head(5)
```

```
Total number of projects approved:  92706
Total number of projects rejected:  16542
Number of projects approved categorized by previously_posted:  92706
Number of projects rejected categorized by previously_posted:  16542
```

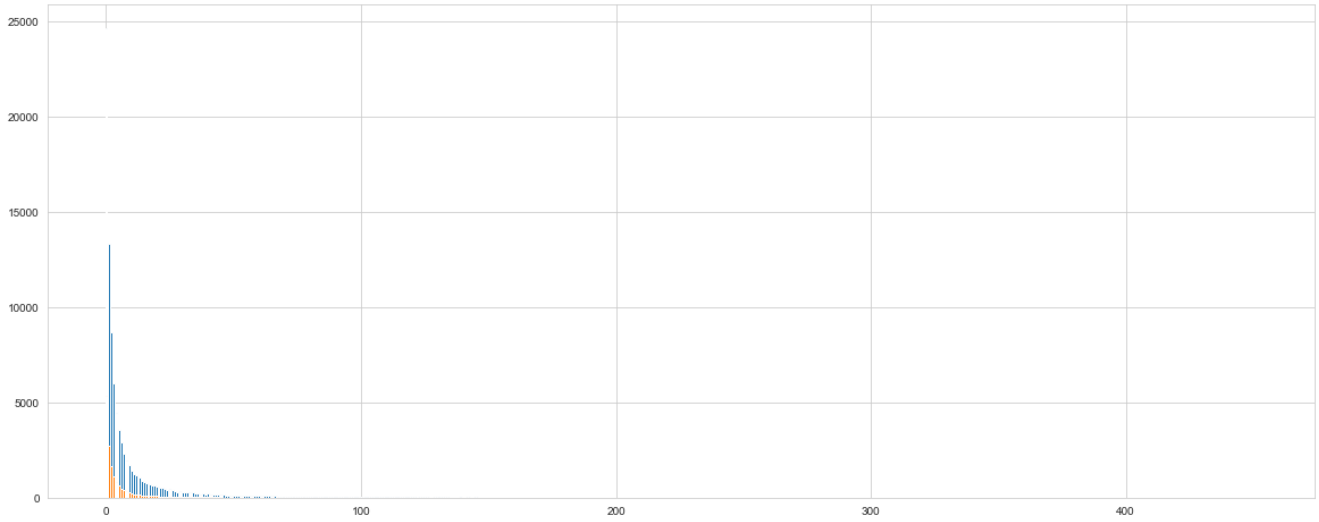
Out[51]:

	teacher_number_of_previously_posted_projects	project_is_approved_x	project_is_approved_y
0	0	24652	5362

	teacher_number_of_previously_posted_projects	project_is_approved_x	project_is_approved_y
1	1	13329	2726
2	2	8705	1645
3	3	5997	1113
4	4	4452	814

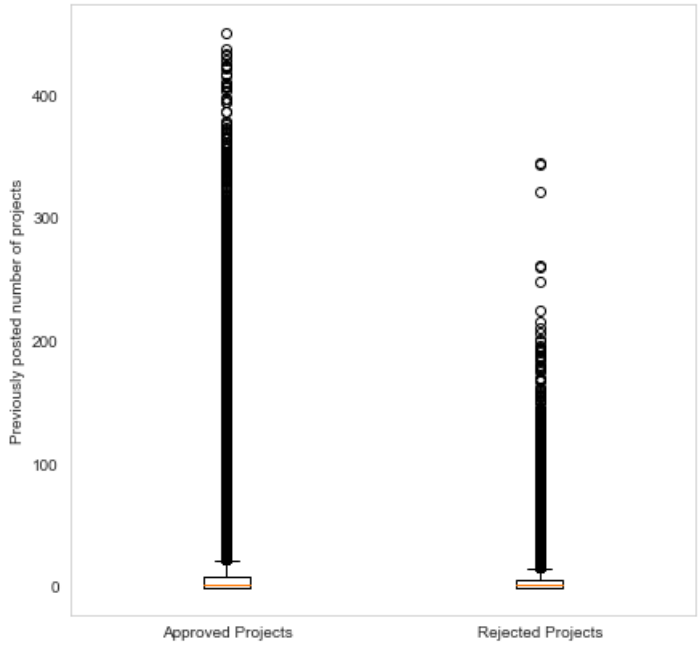
In [52]:

```
plt.figure(figsize = (20, 8));
plt.bar(previouslyPostedNumberData.teacher_number_of_previously_posted_projects,
previouslyPostedNumberData.project_is_approved_x);
plt.bar(previouslyPostedNumberData.teacher_number_of_previously_posted_projects,
previouslyPostedNumberData.project_is_approved_y);
plt.show();
```



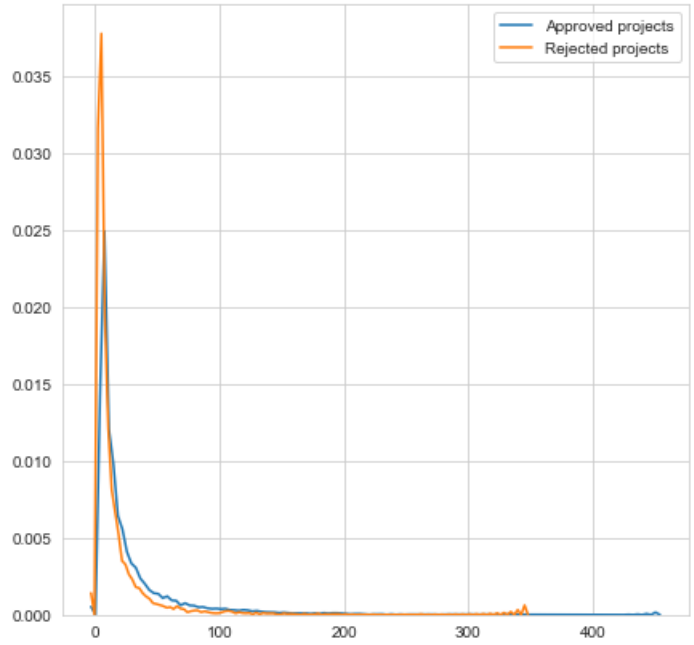
In [53]:

```
previouslyPostedApprovedData = projectsData[projectsData['project_is_approved'] ==
1].teacher_number_of_previously_posted_projects;
previouslyPostedRejectedData = projectsData[projectsData['project_is_approved'] ==
0].teacher_number_of_previously_posted_projects;
plt.boxplot([previouslyPostedApprovedData, previouslyPostedRejectedData]);
plt.grid();
plt.xticks([1, 2], ['Approved Projects', 'Rejected Projects']);
plt.ylabel('Previously posted number of projects');
plt.show();
```



In [54]:

```
sbrn.kdeplot(previouslyPostedApprovedData, label = "Approved projects", bw = 1);
sbrn.kdeplot(previouslyPostedRejectedData, label = "Rejected projects", bw = 1);
plt.show();
```



Observation:

1. Most of the projects approved and rejected are with less number of teacher_number_of_previously_posted_projects. So the approval is not much depending on how many number of projects proposed by teacher previously.

In [55]:

```
def stringContainsNumbers(string):
    return any([character.isdigit() for character in string])
```

In [56]:

```
numericResourceApprovedData =
projectsData[(projectsData['project_resource_summary'].apply(stringContainsNumbers) == True) &
(projectsData['project_is_approved'] == 1)]
textResourceApprovedData =
projectsData[(projectsData['project_resource_summary'].apply(stringContainsNumbers) == False) &
(projectsData['project_is_approved'] == 1)]
numericResourceRejectedData =
projectsData[(projectsData['project_resource_summary'].apply(stringContainsNumbers) == True) &
(projectsData['project_is_approved'] == 0)]
textResourceRejectedData =
projectsData[(projectsData['project_resource_summary'].apply(stringContainsNumbers) == False) &
(projectsData['project_is_approved'] == 0)]
print("Checking whether numbers in resource summary will be useful for project approval?");
equalsBorder(70);
print("Number of approved projects with numbers in resource summary: ",
numericResourceApprovedData.shape[0]);
print("Number of rejected projects with numbers in resource summary: ",
numericResourceRejectedData.shape[0]);
print("Number of approved projects without numbers in resource summary: ",
textResourceApprovedData.shape[0]);
print("Number of rejected projects without numbers in resource summary: ",
textResourceRejectedData.shape[0]);
```

```
Checking whether numbers in resource summary will be useful for project approval?
======================================================================
Number of approved projects with numbers in resource summary:  14090
Number of rejected projects with numbers in resource summary:  1666
```

```
Number of approved projects without numbers in resource summary:  78616
Number of rejected projects without numbers in resource summary:  14876
```

Observation:

1. The rejection rate of project is less when projects resource summary has numbers in it.
2. Even the number of projects approved without numbers in resource summary is high which means that the classification does not actually depends on whether resource summary contains numerical digits or not.

Conclusion of univariate analysis:

1. There is huge overlap of approved and rejected projects when taken for all single featurs. So, this project cannot be classified using single features.
2. project_title is some what better in text type of feature because of less overlap than others.
3. The project approval is not depending on resources cost, but the probability of project rejection is more when resources cost is more.

Preprocessing data

In [57]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# All stopwords that are needed to be removed in the text
stopWords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]);
def preProcessingWithAndWithoutStopWords(texts):
    """
    This function takes list of texts and returns preprocessed list of texts one with
    stop words and one without stopwords.
    """
    # Variable for storing preprocessed text with stop words
    preProcessedTextsWithStopWords = [];
    # Variable for storing preprocessed text without stop words
    preProcessedTextsWithoutStopWords = [];

    # Looping over list of texts for performing pre processing
    for text in tqdm(texts, total = len(texts)):
        # Removing all links in the text
        text = re.sub(r"http\S+", "", text);

        # Removing all html tags in the text
        text = re.sub(r"<\w+/>", "", text);
```

```
        text = re.sub(r"<\w+>", "", text);

        # https://stackoverflow.com/a/47091490/4084039
        # Replacing all below words with adverbs
        text = re.sub(r"won't", "will not", text)
        text = re.sub(r"can\'t", "can not", text)
        text = re.sub(r"n\'t", " not", text)
        text = re.sub(r"\'re", " are", text)
        text = re.sub(r"\'s", " is", text)
        text = re.sub(r"\'d", " would", text)
        text = re.sub(r"\'ll", " will", text)
        text = re.sub(r"\'t", " not", text)
        text = re.sub(r"\'ve", " have", text)
        text = re.sub(r"\'m", " am", text)

        # Removing backslash symbols in text
        text = text.replace('\\r', ' ');
        text = text.replace('\\n', ' ');
        text = text.replace('\\"', ' ');

        # Removing all special characters of text
        text = re.sub(r"[^a-zA-Z0-9]+", " ", text);

        # Converting whole review text into lower case
        text = text.lower();

        # adding this preprocessed text without stopwords to list
        preProcessedTextsWithStopWords.append(text);

        # removing stop words from text
        textWithoutStopWords = ' '.join([word for word in text.split() if word not in stopWords]);
        # adding this preprocessed text without stopwords to list
        preProcessedTextsWithoutStopWords.append(textWithoutStopWords);

    return [preProcessedTextsWithStopWords, preProcessedTextsWithoutStopWords];
```

In [58]:

```
texts = [projectsData['project_essay'].values[0]]
preProcessedTextsWithStopWords, preProcessedTextsWithoutStopWords =
preProcessingWithAndWithoutStopWords(texts);
print("Example project essay without pre-processing: ");
equalsBorder(70);
print(texts);
equalsBorder(70);
print("Example project essay with stop words and pre-processing: ");
equalsBorder(70);
print(preProcessedTextsWithStopWords);
equalsBorder(70);
print("Example project essay without stop words and pre-processing: ");
equalsBorder(70);
print(preProcessedTextsWithoutStopWords);
```

```
Example project essay without pre-processing:
======================================================================
['My students are English learners that are working on English as their second or third languages.
We are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of langu
age to our school. \\r\\n\\r\\n We have over 24 languages represented in our English Learner
program with students at every level of mastery.  We also have over 40 countries represented with
the families within our school.  Each student brings a wealth of knowledge and experiences to us t
hat open our eyes to new cultures, beliefs, and respect.\\"The limits of your language are the lim
its of your world.\\"-Ludwig Wittgenstein  Our English learner\'s have a strong support system at
home that begs for more resources.  Many times our parents are learning to read and speak English
along side of their children.  Sometimes this creates barriers for parents to be able to help
their child learn phonetics, letter recognition, and other reading skills.\\r\\n\\r\\nBy providing
these dvd\'s and players, students are able to continue their mastery of the English language even
if no one at home is able to assist.  All families with students within the Level 1 proficiency st
atus, will be a offered to be a part of this program.  These educational videos will be specially
chosen by the English Learner Teacher and will be sent home regularly to watch.  The videos are to
help the child develop early reading skills.\\r\\n\\r\\nParents that do not have access to a dvd p
layer will have the opportunity to check out a dvd player to use for the year.  The plan is to use
these videos and educational dvd\'s for the years to come for other EL students.\\r\\nnannan']
======================================================================
Example project essay with stop words and pre-processing:
======================================================================
```

```
['my students are english learners that are working on english as their second or third languages
we are a melting pot of refugees immigrants and native born americans bringing the gift of languag
e to our school we have over 24 languages represented in our english learner program with students
at every level of mastery we also have over 40 countries represented with the families within our
school each student brings a wealth of knowledge and experiences to us that open our eyes to new c
ultures beliefs and respect the limits of your language are the limits of your world ludwig wittge
nstein our english learner is have a strong support system at home that begs for more resources ma
ny times our parents are learning to read and speak english along side of their children sometimes
this creates barriers for parents to be able to help their child learn phonetics letter
recognition and other reading skills by providing these dvd is and players students are able to co
ntinue their mastery of the english language even if no one at home is able to assist all families
with students within the level 1 proficiency status will be a offered to be a part of this program
these educational videos will be specially chosen by the english learner teacher and will be sent
home regularly to watch the videos are to help the child develop early reading skills parents that
do not have access to a dvd player will have the opportunity to check out a dvd player to use for
the year the plan is to use these videos and educational dvd is for the years to come for other el
students nannan']
==========================================================================
Example project essay without stop words and pre-processing:
==========================================================================
['students english learners working english second third languages melting pot refugees immigrants
native born americans bringing gift language school 24 languages represented english learner progr
am students every level mastery also 40 countries represented families within school student bring
s wealth knowledge experiences us open eyes new cultures beliefs respect limits language limits wo
rld ludwig wittgenstein english learner strong support system home begs resources many times
parents learning read speak english along side children sometimes creates barriers parents able he
lp child learn phonetics letter recognition reading skills providing dvd players students able con
tinue mastery english language even no one home able assist families students within level 1 profi
ciency status offered part program educational videos specially chosen english learner teacher sen
t home regularly watch videos help child develop early reading skills parents not access dvd
player opportunity check dvd player use year plan use videos educational dvd years come el student
s nannan']
```

In [59]:

```
projectEssays = projectsData['project_essay'];
preProcessedEssaysWithStopWords, preProcessedEssaysWithoutStopWords =
preProcessingWithAndWithoutStopWords(projectEssays);
```

In [60]:

```
preProcessedEssaysWithoutStopWords[0:3]
```

Out[60]:

```
['students english learners working english second third languages melting pot refugees immigrants
native born americans bringing gift language school 24 languages represented english learner progr
am students every level mastery also 40 countries represented families within school student bring
s wealth knowledge experiences us open eyes new cultures beliefs respect limits language limits wo
rld ludwig wittgenstein english learner strong support system home begs resources many times
parents learning read speak english along side children sometimes creates barriers parents able he
lp child learn phonetics letter recognition reading skills providing dvd players students able con
tinue mastery english language even no one home able assist families students within level 1 profi
ciency status offered part program educational videos specially chosen english learner teacher sen
t home regularly watch videos help child develop early reading skills parents not access dvd
player opportunity check dvd player use year plan use videos educational dvd years come el student
s nannan',
 'students arrive school eager learn polite generous strive best know education succeed life help
improve lives school focuses families low incomes tries give student education deserve not much st
udents use materials given best projector need school crucial academic improvement students techno
logy continues grow many resources internet teachers use growth students however school limited re
sources particularly technology without disadvantage one things could really help classrooms
projector projector not crucial instruction also growth students projector show presentations docu
mentaries photos historical land sites math problems much projector make teaching learning easier
also targeting different types learners classrooms auditory visual kinesthetic etc nannan',
 'true champions not always ones win guts mia hamm quote best describes students cholla middle sch
ool approach playing sports especially girls boys soccer teams teams made 7th 8th grade students n
ot opportunity play organized sport due family financial difficulties teach title one middle schoo
l urban neighborhood 74 students qualify free reduced lunch many come activity sport opportunity p
oor homes students love participate sports learn new skills apart team atmosphere school lacks fun
ding meet students needs concerned lack exposure not prepare participating sports teams high schoo
l end school year goal provide students opportunity learn variety soccer skills positive qualities
person actively participates team students campus come school knowing face uphill battle comes par
ticipating organized sports players would thrive field confidence appropriate soccer equipment pla
```

```
y soccer best abilities students experience helpful person part team teaches positive supportive e
ncouraging others students using soccer equipment practice games daily basis learn practice
necessary skills develop strong soccer team experience create opportunity students learn part team
positive contribution teammates students get opportunity learn practice variety soccer skills use
skills game access type experience nearly impossible without soccer equipment students players uti
lize practice games nannan']
```

In [61]:

```
projectTitles = projectsData['project_title'];
preProcessedProjectTitlesWithStopWords, preProcessedProjectTitlesWithoutStopWords =
preProcessingWithAndWithoutStopWords(projectTitles);
preProcessedProjectTitlesWithoutStopWords[0:5]
```

Out[61]:

```
['educational support english learners home',
 'wanted projector hungry learners',
 'soccer equipment awesome middle school students',
 'techie kindergarteners',
 'interactive math tools']
```

Preparing data for classification and modelling

In [62]:

```
pd.DataFrame(projectsData.columns, columns = ['All features in projects data'])
```

Out[62]:

	All features in projects data
0	Unnamed: 0
1	id
2	teacher_id
3	teacher_prefix
4	school_state
5	project_submitted_datetime
6	project_grade_category
7	project_subject_categories
8	project_subject_subcategories
9	project_title
10	project_essay_1
11	project_essay_2
12	project_essay_3
13	project_essay_4
14	project_resource_summary
15	teacher_number_of_previously_posted_projects
16	project_is_approved
17	cleaned_categories
18	cleaned_sub_categories
19	project_essay
20	price
21	quantity

Useful features:

Here we will consider only below features for classification and we can ignore the other features

Categorical data:

1. **school_state** - categorical data
2. **project_grade_category** - categorical data
3. **cleaned_categories** - categorical data
4. **cleaned_sub_categories** - categorical data
5. **teacher_prefix** - categorical data

Text data:

1. **project_resource_summary** - text data
2. **project_title - text data**
3. **project_resource_summary** - text data

Numerical data:

1. **teacher_number_of_previously_posted_projects** - numerical data
2. **price** - numerical data
3. **quantity** - numerical data

Vectorizing categorical data

1. Vectorizing cleaned_categories(project_subject_categories cleaned) - One Hot Encoding

In [63]:

```
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e cleaned_categories
subjectsCategoriesVectorizer = CountVectorizer(vocabulary = list(sortedCategoriesDictionary.keys()
), lowercase = False, binary = True);
# Fitting CountVectorizer with cleaned_categories values
subjectsCategoriesVectorizer.fit(projectsData['cleaned_categories'].values);
# Vectorizing categories using one-hot-encoding
categoriesVectors = subjectsCategoriesVectorizer.transform(projectsData['cleaned_categories'].valu
es);
```

In [64]:

```
print("Features used in vectorizing categories: ");
equalsBorder(70);
print(subjectsCategoriesVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of cleaned_categories matrix after vectorization(one-hot-encoding): ",
categoriesVectors.shape);
equalsBorder(70);
print("Sample vectors of categories: ");
equalsBorder(70);
print(categoriesVectors[0:4])
```

```
Features used in vectorizing categories:
======================================================================
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
======================================================================
Shape of cleaned_categories matrix after vectorization(one-hot-encoding):  (109248, 9)
======================================================================
Sample vectors of categories:
======================================================================
  (0, 8) 1
  (1, 2) 1
  (1, 6) 1
  (2, 6) 1
```

```
  (2, 6) 1
  (3, 7) 1
  (3, 8) 1
```

2. Vectorizing cleaned_sub_categories(project_subject_sub_categories cleaned) - One Hot Encoding

In [65]:

```
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e cleaned_sub_categories
subjectsSubCategoriesVectorizer = CountVectorizer(vocabulary = list(sortedDictionarySubCategories.
keys()), lowercase = False, binary = True);
# Fitting CountVectorizer with cleaned_sub_categories values
subjectsSubCategoriesVectorizer.fit(projectsData['cleaned_sub_categories'].values);
# Vectorizing sub categories using one-hot-encoding
subCategoriesVectors =
subjectsSubCategoriesVectorizer.transform(projectsData['cleaned_sub_categories'].values);
```

In [66]:

```
print("Features used in vectorizing subject sub categories: ");
equalsBorder(70);
print(subjectsSubCategoriesVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of cleaned_categories matrix after vectorization(one-hot-encoding): ",
subCategoriesVectors.shape);
equalsBorder(70);
print("Sample vectors of categories: ");
equalsBorder(70);
print(subCategoriesVectors[0:4])
```

```
Features used in vectorizing subject sub 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',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
======================================================================
Shape of cleaned_categories matrix after vectorization(one-hot-encoding):  (109248, 30)
======================================================================
Sample vectors of categories:
======================================================================
  (0, 20) 1
  (0, 29) 1
  (1, 5) 1
  (1, 13) 1
  (2, 13) 1
  (2, 24) 1
  (3, 28) 1
  (3, 29) 1
```

3. Vectorizing teacher_prefix - One Hot Encoding

In [67]:

```
def giveCounter(data):
    counter = Counter();
    for dataValue in data:
        counter.update(str(dataValue).split());
    return counter
```

In [68]:

```
giveCounter(projectsData['teacher_prefix'].values)
```

Out[68]:

```
Counter({'Mrs.': 57269,
         'Mr.': 10648,
         'Ms.': 38955,
         'Teacher': 2360,
         'nan': 3,
         'Dr.': 13})
```

In [69]:

```
projectsData = projectsData.dropna(subset = ['teacher_prefix']);
projectsData.shape
```

Out[69]:

```
(109245, 22)
```

In [70]:

```
teacherPrefixDictionary = dict(giveCounter(projectsData['teacher_prefix'].values));
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e teacher_prefix
teacherPrefixVectorizer = CountVectorizer(vocabulary = list(teacherPrefixDictionary.keys()),
lowercase = False, binary = True);
# Fitting CountVectorizer with teacher_prefix values
teacherPrefixVectorizer.fit(projectsData['teacher_prefix'].values);
# Vectorizing teacher_prefix using one-hot-encoding
teacherPrefixVectors = teacherPrefixVectorizer.transform(projectsData['teacher_prefix'].values);
```

In [71]:

```
print("Features used in vectorizing teacher_prefix: ");
equalsBorder(70);
print(teacherPrefixVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of teacher_prefix matrix after vectorization(one-hot-encoding): ",
teacherPrefixVectors.shape);
equalsBorder(70);
print("Sample vectors of teacher_prefix: ");
equalsBorder(70);
print(teacherPrefixVectors[0:100]);
```

```
Features used in vectorizing teacher_prefix:
======================================================================
['Mrs.', 'Mr.', 'Ms.', 'Teacher', 'Dr.']
======================================================================
Shape of teacher_prefix matrix after vectorization(one-hot-encoding):  (109245, 5)
======================================================================
Sample vectors of teacher_prefix:
======================================================================
  (27, 3) 1
  (75, 3) 1
  (82, 3) 1
  (88, 3) 1
```

In [72]:

```
teacherPrefixes = [prefix.replace('.', '') for prefix in projectsData['teacher_prefix'].values];
teacherPrefixes[0:5]
```

Out[72]:

```
['Mrs', 'Mr', 'Ms', 'Mrs', 'Mrs']
```

In [73]:

```
projectsData['teacher_prefix'] = teacherPrefixes;
projectsData.head(3)
```

Out[73]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr	FL	2016-10-25 09:22:10	Gra
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms	AZ	2016-08-31 12:03:56	Gra

3 rows × 22 columns

In [74]:

```
teacherPrefixDictionary = dict(giveCounter(projectsData['teacher_prefix'].values));
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e teacher_prefix
teacherPrefixVectorizer = CountVectorizer(vocabulary = list(teacherPrefixDictionary.keys()),
lowercase = False, binary = True);
# Fitting CountVectorizer with teacher_prefix values
teacherPrefixVectorizer.fit(projectsData['teacher_prefix'].values);
# Vectorizing teacher_prefix using one-hot-encoding
teacherPrefixVectors = teacherPrefixVectorizer.transform(projectsData['teacher_prefix'].values);
```

In [75]:

```
print("Features used in vectorizing teacher_prefix: ");
equalsBorder(70);
print(teacherPrefixVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of teacher_prefix matrix after vectorization(one-hot-encoding): ",
teacherPrefixVectors.shape);
equalsBorder(70);
print("Sample vectors of teacher_prefix: ");
equalsBorder(70);
print(teacherPrefixVectors[0:4]);
```

```
Features used in vectorizing teacher_prefix:
======================================================================
['Mrs', 'Mr', 'Ms', 'Teacher', 'Dr']
======================================================================
Shape of teacher_prefix matrix after vectorization(one-hot-encoding):  (109245, 5)
======================================================================
Sample vectors of teacher_prefix:
======================================================================
  (0, 0) 1
  (1, 1) 1
  (2, 2) 1
  (3, 0) 1
```

4. Vectorizing school_state - One Hot Encoding

In [76]:

```
schoolStateDictionary = dict(giveCounter(projectsData['school_state'].values));
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e school states
schoolStateVectorizer = CountVectorizer(vocabulary = list(schoolStateDictionary.keys()), lowercase
= False, binary = True);
```

```
# Fitting CountVectorizer with school_state values
schoolStateVectorizer.fit(projectsData['school_state'].values);
# Vectorizing school_state using one-hot-encoding
schoolStateVectors = schoolStateVectorizer.transform(projectsData['school_state'].values);
```

In [77]:

```
print("Features used in vectorizing school_state: ");
equalsBorder(70);
print(schoolStateVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of school_state matrix after vectorization(one-hot-encoding): ", schoolStateVectors.s
hape);
equalsBorder(70);
print("Sample vectors of school_state: ");
equalsBorder(70);
print(schoolStateVectors[0:4]);
```

```
Features used in vectorizing school_state:
======================================================================
['IN', 'FL', 'AZ', 'KY', 'TX', 'CT', 'GA', 'SC', 'NC', 'CA', 'NY', 'OK', 'MA', 'NV', 'OH', 'PA', 'A
L', 'LA', 'VA', 'AR', 'WA', 'WV', 'ID', 'TN', 'MS', 'CO', 'UT', 'IL', 'MI', 'HI', 'IA', 'RI', 'NJ',
'MO', 'DE', 'MN', 'ME', 'WY', 'ND', 'OR', 'AK', 'MD', 'WI', 'SD', 'NE', 'NM', 'DC', 'KS', 'MT', 'NH
', 'VT']
======================================================================
Shape of school_state matrix after vectorization(one-hot-encoding):  (109245, 51)
======================================================================
Sample vectors of school_state:
======================================================================
  (0, 0) 1
  (1, 1) 1
  (2, 2) 1
  (3, 3) 1
```

5. Vectorizing project_grade_category - One Hot Encoding

In [78]:

```
giveCounter(projectsData['project_grade_category'])
```

Out[78]:

```
Counter({'Grades': 109245,
         'PreK-2': 44225,
         '6-8': 16923,
         '3-5': 37135,
         '9-12': 10962})
```

In [79]:

```
cleanedGrades = []
for grade in projectsData['project_grade_category'].values:
    grade = grade.replace(' ', '');
    grade = grade.replace('-', 'to');
    cleanedGrades.append(grade);
cleanedGrades[0:4]
```

Out[79]:

```
['GradesPreKto2', 'Grades6to8', 'Grades6to8', 'GradesPreKto2']
```

In [80]:

```
projectsData['project_grade_category'] = cleanedGrades
projectsData.head(4)
```

Out[80]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs	IN	2016-12-05 13:43:57	Gra
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr	FL	2016-10-25 09:22:10	Gra
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms	AZ	2016-08-31 12:03:56	Gra
3	45	p246581	f3cb9bffbba169bef1a77b243e620b60	Mrs	KY	2016-10-06 21:16:17	Gra

4 rows × 22 columns

In [81]:

```
projectGradeDictionary = dict(giveCounter(projectsData['project_grade_category'].values));
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e project grade categories
projectGradeVectorizer = CountVectorizer(vocabulary = list(projectGradeDictionary.keys()),
lowercase = False, binary = True);
# Fitting CountVectorizer with project_grade_category values
projectGradeVectorizer.fit(projectsData['project_grade_category'].values);
# Vectorizing project_grade_category using one-hot-encoding
projectGradeVectors =
projectGradeVectorizer.transform(projectsData['project_grade_category'].values);
```

In [82]:

```
print("Features used in vectorizing project_grade_category: ");
equalsBorder(70);
print(projectGradeVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of school_state matrix after vectorization(one-hot-encoding): ", projectGradeVectors.
shape);
equalsBorder(70);
print("Sample vectors of school_state: ");
equalsBorder(70);
print(projectGradeVectors[0:4]);
```

```
Features used in vectorizing project_grade_category:
======================================================================
['GradesPreKto2', 'Grades6to8', 'Grades3to5', 'Grades9to12']
======================================================================
Shape of school_state matrix after vectorization(one-hot-encoding):  (109245, 4)
======================================================================
Sample vectors of school_state:
======================================================================
  (0, 0) 1
  (1, 1) 1
  (2, 1) 1
  (3, 0) 1
```

In [135]:

```
projectsDataSub = projectsData[0:40000];
preProcessedEssaysWithoutStopWordsSub = preProcessedEssaysWithoutStopWords[0:40000];
preProcessedProjectTitlesWithoutStopWordsSub = preProcessedProjectTitlesWithoutStopWords[0:40000];
```

Vectorizing Text Data

Bag of Words

1. Vectorizing project_essay

In [136]:

```
# Initializing countvectorizer for bag of words vectorization of preprocessed project essays
bowEssayVectorizer = CountVectorizer(min_df = 10);
# Transforming the preprocessed essays to bag of words vectors
bowEssayModel = bowEssayVectorizer.fit_transform(preProcessedEssaysWithoutStopWordsSub);
```

In [137]:

```
print("Some of the Features used in vectorizing preprocessed essays: ");
equalsBorder(70);
print(bowEssayVectorizer.get_feature_names()[-40:]);
equalsBorder(70);
print("Shape of preprocessed essay matrix after vectorization: ", bowEssayModel.shape);
equalsBorder(70);
print("Sample bag-of-words vector of preprocessed essay: ");
equalsBorder(70);
print(bowEssayModel[0])
```

```
Some of the Features used in vectorizing preprocessed essays:
======================================================================
['yeats', 'yell', 'yelling', 'yellow', 'yemen', 'yes', 'yesterday', 'yet', 'yield', 'yields', 'yog
a', 'york', 'younannan', 'young', 'younger', 'youngest', 'youngsters', 'youth', 'youthful',
'youths', 'youtube', 'yummy', 'zeal', 'zearn', 'zen', 'zenergy', 'zero', 'zest', 'zip', 'ziploc',
'zippers', 'zipping', 'zone', 'zoned', 'zones', 'zoo', 'zoom', 'zooming', 'zoos', 'zumba']
======================================================================
Shape of preprocessed essay matrix after vectorization:  (40000, 11077)
======================================================================
Sample bag-of-words vector of preprocessed essay:
======================================================================
  (0, 6533) 1
  (0, 3306) 1
  (0, 1981) 1
  (0, 11036) 1
  (0, 7347) 1
  (0, 11029) 1
  (0, 10530) 2
  (0, 1734) 1
  (0, 6855) 1
  (0, 7374) 2
  (0, 232) 1
  (0, 6687) 1
  (0, 3211) 1
  (0, 2805) 1
  (0, 10766) 1
  (0, 8133) 1
  (0, 8803) 1
  (0, 9831) 1
  (0, 1794) 1
  (0, 9237) 1
  (0, 10639) 3
  (0, 3274) 2
  (0, 7068) 1
  (0, 6798) 1
  (0, 9399) 1
  : :
  (0, 6123) 2
  (0, 5785) 2
  (0, 3613) 1
  (0, 7703) 2
  (0, 5732) 3
  (0, 8269) 2
  (0, 67) 1
  (0, 8670) 2
  (0, 5664) 3
  (0, 4383) 1
```

```
  (0, 1339) 1
  (0, 553) 1
  (0, 1248) 1
  (0, 6549) 1
  (0, 5003) 1
  (0, 8116) 1
  (0, 7501) 1
  (0, 6207) 1
  (0, 5665) 2
  (0, 9968) 1
  (0, 8736) 1
  (0, 10964) 1
  (0, 5733) 1
  (0, 3449) 7
  (0, 9553) 5
```

2. Vectorizing project_title

In [138]:

```
# Initializing countvectorizer for bag of words vectorization of preprocessed project titles
bowTitleVectorizer = CountVectorizer(min_df = 10);
# Transforming the preprocessed project titles to bag of words vectors
bowTitleModel = bowTitleVectorizer.fit_transform(preProcessedProjectTitlesWithoutStopWordsSub);
```

In [139]:

```
print("Some of the Features used in vectorizing preprocessed titles: ");
equalsBorder(70);
print(bowTitleVectorizer.get_feature_names()[-40:]);
equalsBorder(70);
print("Shape of preprocessed title matrix after vectorization: ", bowTitleModel.shape);
equalsBorder(70);
print("Sample bag-of-words vector of preprocessed title: ");
equalsBorder(70);
print(bowTitleModel[0])
```

```
Some of the Features used in vectorizing preprocessed titles:
======================================================================
['wireless', 'wise', 'wish', 'within', 'without', 'wizards', 'wo', 'wobble', 'wobbles',
'wobbling', 'wobbly', 'wonder', 'wonderful', 'wonders', 'word', 'words', 'work', 'workers',
'working', 'works', 'workshop', 'world', 'worlds', 'worms', 'worth', 'would', 'wow', 'write', 'wri
ter', 'writers', 'writing', 'ye', 'year', 'yearbook', 'yes', 'yoga', 'young', 'youth', 'zone', 'zo
om']
======================================================================
Shape of preprocessed title matrix after vectorization:  (40000, 1774)
======================================================================
Sample bag-of-words vector of preprocessed title:
======================================================================
  (0, 766) 1
  (0, 906) 1
  (0, 514) 1
  (0, 1553) 1
  (0, 483) 1
```

Tf-Idf Vectorization

1. Vectorizing project_essay

In [140]:

```
# Intializing tfidf vectorizer for tf-idf vectorization of preprocessed project essays
tfIdfEssayVectorizer = TfidfVectorizer(min_df = 10);
# Transforming the preprocessed project essays to tf-idf vectors
tfIdfEssayModel = tfIdfEssayVectorizer.fit_transform(preProcessedEssaysWithoutStopWordsSub);
```

In [141]:

```
```

```
print("Some of the Features used in tf-idf vectorizing preprocessed essays: ");
equalsBorder(70);
print(tfIdfEssayVectorizer.get_feature_names()[-40:]);
equalsBorder(70);
print("Shape of preprocessed title matrix after tf-idf vectorization: ", tfIdfEssayModel.shape);
equalsBorder(70);
print("Sample Tf-Idf vector of preprocessed essay: ");
equalsBorder(70);
print(tfIdfEssayModel[0])
```

```
Some of the Features used in tf-idf vectorizing preprocessed essays:
======================================================================
['yeats', 'yell', 'yelling', 'yellow', 'yemen', 'yes', 'yesterday', 'yet', 'yield', 'yields', 'yog
a', 'york', 'younannan', 'young', 'younger', 'youngest', 'youngsters', 'youth', 'youthful',
'youths', 'youtube', 'yummy', 'zeal', 'zearn', 'zen', 'zenergy', 'zero', 'zest', 'zip', 'ziploc',
'zippers', 'zipping', 'zone', 'zoned', 'zones', 'zoo', 'zoom', 'zooming', 'zoos', 'zumba']
======================================================================
Shape of preprocessed title matrix after tf-idf vectorization:  (40000, 11077)
======================================================================
Sample Tf-Idf vector of preprocessed essay:
======================================================================
  (0, 9553) 0.07732161197654648
  (0, 3449) 0.2978137199079083
  (0, 5733) 0.036113118250709 74
  (0, 10964) 0.03819325396356506
  (0, 8736) 0.04966730436190034
  (0, 9968) 0.05933894161734909
  (0, 5665) 0.13189136979245247
  (0, 6207) 0.09909858268088724
  (0, 7501) 0.09797369103397546
  (0, 8116) 0.09716121418147701
  (0, 5003) 0.09174889764250635
  (0, 6549) 0.07739523816315956
  (0, 1248) 0.09041771504928811
  (0, 553) 0.09502243963232913
  (0, 1339) 0.07922532406820633
  (0, 4383) 0.08387324724715874
  (0, 5664) 0.12052414724469786
  (0, 8670) 0.03565737676523101
  (0, 67) 0.0797508795755641
  (0, 8269) 0.18440093271700464
  (0, 5732) 0.23244852084297085
  (0, 7703) 0.0932371184396508
  (0, 3613) 0.033250154942777416
  (0, 5785) 0.08336998078832462
  (0, 6123) 0.18451571587493337
  : :
  (0, 9399) 0.0680639151319745
  (0, 6798) 0.08632328546640713
  (0, 7068) 0.046135007257522224
  (0, 3274) 0.10489683635458984
  (0, 10639) 0.2063461965343629
  (0, 9237) 0.1100116652395096
  (0, 1794) 0.07900547931629058
  (0, 9831) 0.03792376194008962
  (0, 8803) 0.09740047454864696
  (0, 8133) 0.09001501053091984
  (0, 10766) 0.07024528926492071
  (0, 2805) 0.05089165427462248
  (0, 3211) 0.06222851802675729
  (0, 6687) 0.022226920710368445
  (0, 232) 0.040248356980164615
  (0, 7374) 0.1846309297399045
  (0, 6855) 0.03799907965204156
  (0, 1734) 0.07743897673831124
  (0, 10530) 0.05491069896079749
  (0, 11029) 0.030886589234837624
  (0, 7347) 0.06268239285732621
  (0, 11036) 0.04610937510882687
  (0, 1981) 0.02654012905964554
  (0, 3306) 0.1031894334469226
  (0, 6533) 0.016043824658976313
```

2. Vectorizing project_title

In [142]:

```
# Intializing tfidf vectorizer for tf-idf vectorization of preprocessed project titles
tfIdfTitleVectorizer = TfidfVectorizer(min_df = 10);
# Transforming the preprocessed project titles to tf-idf vectors
tfIdfTitleModel = tfIdfTitleVectorizer.fit_transform(preProcessedProjectTitlesWithoutStopWordsSub)
;
```

In [143]:

```
print("Some of the Features used in tf-idf vectorizing preprocessed titles: ");
equalsBorder(70);
print(tfIdfTitleVectorizer.get_feature_names()[-40:]);
equalsBorder(70);
print("Shape of preprocessed title matrix after tf-idf vectorization: ", tfIdfTitleModel.shape);
equalsBorder(70);
print("Sample Tf-Idf vector of preprocessed title: ");
equalsBorder(70);
print(tfIdfTitleModel[0])
```

```
Some of the Features used in tf-idf vectorizing preprocessed titles:
======================================================================
['wireless', 'wise', 'wish', 'within', 'without', 'wizards', 'wo', 'wobble', 'wobbles',
'wobbling', 'wobbly', 'wonder', 'wonderful', 'wonders', 'word', 'words', 'work', 'workers',
'working', 'works', 'workshop', 'world', 'worlds', 'worms', 'worth', 'would', 'wow', 'write', 'wri
ter', 'writers', 'writing', 'ye', 'year', 'yearbook', 'yes', 'yoga', 'young', 'youth', 'zone', 'zo
om']
======================================================================
Shape of preprocessed title matrix after tf-idf vectorization:  (40000, 1774)
======================================================================
Sample Tf-Idf vector of preprocessed title:
======================================================================
  (0, 483) 0.5356140846908081
  (0, 1553) 0.4441059196924978
  (0, 514) 0.4615835742389133
  (0, 906) 0.3400969810242112
  (0, 766) 0.4326223894644794
```

Average Word2Vector Vectorization

In [144]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/
# We should have glove_vectors file for creating below model
with open('glove_vectors', 'rb') as f:
    gloveModel = pickle.load(f)
    gloveWords =  set(gloveModel.keys())
```

In [149]:

```
print("Glove vector of sample word: ");
equalsBorder(70);
print(gloveModel['technology']);
equalsBorder(70);
print("Shape of glove vector: ", gloveModel['technology'].shape);
```

```
Glove vector of sample word:
======================================================================
[-0.26078   -0.36898   -0.022831   0.21666    0.16672   -0.20268
 -3.1219     0.33057    0.71512    0.28874    0.074368  -0.033203
  0.23783    0.21052    0.076562   0.13007   -0.31706   -0.45888
 -0.45463   -0.13191    0.49761    0.072704   0.16811    0.18846
 -0.16688   -0.21973    0.08575   -0.19577   -0.2101    -0.32436
 -0.56336    0.077996  -0.22758   -0.66569    0.14824    0.038945
  0.50881   -0.1352     0.49966   -0.4401    -0.022335  -0.22744
  0.22086    0.21865    0.36647    0.30495   -0.16565    0.038759
  0.28108   -0.2167     0.12453    0.65401    0.34584   -0.2557
 -0.046363  -0.31111   -0.020936  -0.17122   -0.77114    0.29289
 -0.14625    0.39541   -0.078938   0.051127   0.15076    0.085126
  0.183     -0.06755    0.26312    0.0087276  0.0066415  0.37033
```

```
  0.03496   -0.12627   -0.052626  -0.34897    0.14672    0.14799
 -0.21821   -0.042785   0.2661    -1.1105     0.31789    0.27278
  0.054468  -0.27458    0.42732   -0.44101   -0.19302   -0.32948
  0.61501   -0.22301   -0.36354   -0.34983   -0.16125   -0.17195
 -3.363      0.45146   -0.13753    0.31107    0.2061     0.33063
  0.45879    0.24256    0.042342   0.074837  -0.12869    0.12066
  0.42843   -0.4704    -0.18937    0.32685    0.26079    0.20518
 -0.18432   -0.47658    0.69193    0.18731   -0.12516    0.35447
 -0.1969    -0.58981   -0.88914    0.5176     0.13177   -0.078557
  0.032963  -0.19411    0.15109    0.10547   -0.1113    -0.61533
  0.0948    -0.3393    -0.20071   -0.30197    0.29531    0.28017
  0.16049    0.25294   -0.44266   -0.39412    0.13486    0.25178
 -0.044114   1.1519     0.32234   -0.34323   -0.10713   -0.15616
  0.031206   0.46636   -0.52761   -0.39296   -0.068424  -0.04072
  0.41508   -0.34564    0.71001   -0.364      0.2996     0.032281
  0.34035    0.23452    0.78342    0.48045   -0.1609     0.40102
 -0.071795  -0.16531    0.082153   0.52065    0.24194    0.17113
  0.33552   -0.15725   -0.38984    0.59337   -0.19388   -0.39864
 -0.47901    1.0835     0.24473    0.41309    0.64952    0.46846
  0.024386  -0.72087   -0.095061   0.10095   -0.025229   0.29435
 -0.57696    0.53166   -0.0058338 -0.3304     0.19661   -0.085206
  0.34225    0.56262    0.19924   -0.027111  -0.44567    0.17266
  0.20887   -0.40702    0.63954    0.50708   -0.31862   -0.39602
 -0.1714    -0.040006  -0.45077   -0.32482   -0.0316     0.54908
 -0.1121     0.12951   -0.33577   -0.52768   -0.44592   -0.45388
  0.66145    0.33023   -1.9089     0.5318     0.21626   -0.13152
  0.48258    0.68028   -0.84115   -0.51165    0.40017    0.17233
 -0.033749   0.045275   0.37398   -0.18252    0.19877    0.1511
  0.029803   0.16657   -0.12987   -0.50489    0.55311   -0.22504
  0.13085   -0.78459    0.36481   -0.27472    0.031805   0.53052
 -0.20078    0.46392   -0.63554    0.040289  -0.19142   -0.0097011
  0.068084  -0.10602    0.25567    0.096125  -0.10046    0.15016
 -0.26733   -0.26494    0.057888   0.062678  -0.11596    0.28115
  0.25375   -0.17954    0.20615    0.24189    0.062696   0.27719
 -0.42601   -0.28619   -0.44697   -0.082253  -0.73415   -0.20675
 -0.60289   -0.06728    0.15666   -0.042614   0.41368   -0.17367
 -0.54012    0.23883    0.23075    0.13608   -0.058634  -0.089705
  0.18469    0.023634   0.16178    0.23384    0.24267    0.091846 ]
==========================================================================
Shape of glove vector:  (300,)
```

In [146]:

```
def getWord2VecVectors(texts):
    word2VecTextsVectors = [];
    for preProcessedText in tqdm(texts):
        word2VecTextVector = np.zeros(300);
        numberOfWordsInText = 0;
        for word in preProcessedText.split():
            if word in gloveWords:
                word2VecTextVector += gloveModel[word];
                numberOfWordsInText += 1;
        if numberOfWordsInText != 0:
            word2VecTextVector = word2VecTextVector / numberOfWordsInText;
        word2VecTextsVectors.append(word2VecTextVector);
    return word2VecTextsVectors;
```

1. Vectorizing project_essay

In [147]:

```
word2VecEssaysVectors = getWord2VecVectors(preProcessedEssaysWithoutStopWords);
```

In [176]:

```
print("Shape of Word2Vec vectorization matrix of essays: {},{}".format(len(word2VecEssaysVectors),
len(word2VecEssaysVectors[0])));
equalsBorder(70);
print("Sample essay: ");
equalsBorder(70);
print(preProcessedEssaysWithoutStopWords[0]);
equalsBorder(70);
print("Word2Vec vector of sample essay: ");
```

```
equalsBorder(70);
print(word2VecEssaysVectors[0]);
```

```
Shape of Word2Vec vectorization matrix of essays: 109248,300
======================================================================
Sample essay:
======================================================================
students english learners working english second third languages melting pot refugees immigrants n
ative born americans bringing gift language school 24 languages represented english learner progra
m students every level mastery also 40 countries represented families within school student brings
wealth knowledge experiences us open eyes new cultures beliefs respect limits language limits worl
d ludwig wittgenstein english learner strong support system home begs resources many times parents
learning read speak english along side children sometimes creates barriers parents able help child
learn phonetics letter recognition reading skills providing dvd players students able continue mas
tery english language even no one home able assist families students within level 1 proficiency st
atus offered part program educational videos specially chosen english learner teacher sent home re
gularly watch videos help child develop early reading skills parents not access dvd player
opportunity check dvd player use year plan use videos educational dvd years come el students nanna
n
======================================================================
Word2Vec vector of sample essay:
======================================================================
[-1.40030644e-02  8.78995685e-02  3.50108161e-02 -5.90358980e-03
 -5.93166809e-02 -6.21039893e-02 -2.96711248e+00  9.45840302e-02
 -8.18737785e-03  4.46964161e-02 -7.64722101e-02  6.97099444e-02
  8.44441262e-02 -1.22974138e-01 -3.55310208e-02 -8.90947154e-02
  1.20959579e-01 -1.21977699e-01  4.61334597e-02 -3.33640832e-02
  1.24900557e-01  7.18837631e-02 -6.14885114e-02 -2.67269047e-02
  6.82086621e-02 -3.60263034e-02  1.17172255e-01 -1.17868631e-01
 -1.13467710e-01 -9.25920168e-02 -2.42461725e-01 -7.92963658e-02
  3.52513154e-03  1.79752468e-01 -4.69217812e-02 -3.56593007e-02
 -7.95331477e-03 -6.71107383e-04 -1.80828067e-02 -1.16224805e-02
 -3.69645852e-02  1.61287176e-01 -1.75201329e-01 -6.02256376e-02
  1.48811886e-02 -9.00106181e-02  7.72160490e-02  7.42989819e-02
 -1.02682389e-02 -1.33311658e-01 -2.82030537e-02 -7.71051879e-03
  7.33988450e-02  3.54095087e-02 -5.80719597e-03 -8.70242758e-02
 -3.57117638e-02  2.78475651e-02 -1.54957291e-01 -3.24157495e-02
 -5.93266570e-02 -8.80254174e-02  2.18914318e-01 -1.22730395e-02
 -1.05831485e-01  1.53985730e-01  7.15618933e-02 -3.97147470e-02
  1.47169116e-01 -4.50476644e-03 -1.49678829e-01  5.52201396e-02
  3.04915879e-02 -6.24086617e-02 -7.68483134e-02 -7.50149195e-02
 -1.07105068e-01 -2.69954530e-02  1.28067340e-01 -3.42946330e-02
  4.24139667e-02 -4.49685043e-01  1.52793905e-01 -9.06178181e-02
 -6.67951510e-02 -2.72063766e-02  7.37261792e-02 -8.64977130e-02
  1.64616877e-01  4.86745523e-02 -4.44542828e-02 -3.04823530e-02
  2.63897436e-02 -6.59345034e-02 -5.21813664e-02 -7.45015886e-02
 -2.21975948e+00  8.57858456e-02  7.73778584e-02  1.14644799e-01
 -1.50536483e-01 -5.17326940e-02  3.23826117e-02 -1.15700542e-01
  7.15651973e-02  9.15412617e-02  5.41334631e-02 -1.25451318e-01
  2.80941483e-02 -3.95890262e-02 -1.67010497e-02  1.74708879e-02
  4.58374505e-02  2.56664910e-01  3.74891134e-02  3.00990497e-02
 -2.18904765e-01  9.37672966e-02  9.99403436e-02  5.26255996e-02
 -6.67958718e-02  5.97650946e-02  4.14311192e-02 -6.85917603e-02
  1.72453235e-02  1.02485026e-01  3.02940430e-02  9.59998859e-03
  1.96364913e-02  1.22438477e-01  7.98410557e-02  1.92611322e-02
  6.44085906e-03  4.94252148e-03 -5.36137718e-03 -1.17976934e-01
  1.77991634e-01 -2.51954819e-02  8.02478188e-02  2.29125079e-01
  3.79080403e-02  1.22892819e-02  7.19621470e-02 -9.25031570e-02
 -8.86571674e-02 -4.74898563e-02  1.68688409e-02 -1.15134901e-01
  1.76528904e-01 -6.30485141e-02 -4.99678329e-02 -1.00350507e-01
  1.25089302e-02 -4.08706114e-02  4.50565289e-02  2.49286074e-02
 -1.29713758e-03 -3.21404376e-02 -2.52972249e-02 -9.63531510e-02
  8.42448993e-04 -7.29482953e-03 -3.77497893e-02 -9.35034987e-02
 -3.45719793e-02  7.15921796e-02 -1.29330935e-01  1.28508101e-02
  4.24846988e-02 -8.43078228e-02  4.79772134e-02 -3.05753799e-02
 -3.03772013e-02 -2.10572558e-01 -1.05464289e-03  5.18230436e-02
 -4.39921874e-02  5.29591584e-02 -1.08551689e-01  2.88053128e-02
 -4.88957058e-02  2.31962381e-01 -2.90986193e-02 -2.83725755e-02
 -6.80350899e-02 -6.99966387e-02 -6.80414679e-02 -7.63552362e-02
 -1.59287859e-02 -2.59947651e-03 -7.81848121e-03 -1.14299579e-01
 -2.02054698e-02  1.21184430e-03  2.59984919e-02 -7.64172013e-02
  9.47882617e-03 -5.71751181e-02  1.25667972e-01 -4.60388139e-02
  5.51296403e-02 -6.73280980e-02 -2.06862389e-02  1.12049165e-01
 -7.63451436e-02  4.71124027e-02  6.32404235e-02 -2.13828034e-02
  1.24239236e-01  5.08985235e-02  2.05136711e-03  1.45916498e-02
  4.25123886e-02 -9.41766832e-02 -3.08569389e-02 -2.57995470e-02
```

```
 -3.53808765e-02 -7.16000389e-02  1.35426121e-02  4.57596799e-02
 -1.85721693e-01 -6.62042523e-02 -1.45448285e-01  5.50366758e-02
 -2.09367026e+00  1.23479489e-01 -1.46630889e-01 -8.86940765e-02
 -7.32806463e-02 -1.48629733e-01  3.23867248e-03  7.08553181e-02
  1.10315906e-02 -2.35431879e-02 -7.69633283e-02 -1.13640894e-01
  9.96301846e-02 -5.70585054e-02 -5.45997987e-04  9.42995174e-02
 -1.40422433e-01 -5.03571812e-04 -2.50305216e-01  3.79384141e-02
 -6.44086637e-02 -1.53146188e-02 -2.55858274e-02 -1.10195376e-01
  1.62183899e-02 -1.61929591e-02  2.03421993e-02  1.21424534e-01
  5.02740463e-02  2.37900799e-02  9.07398322e-02  1.57962685e-02
  3.73036075e-02 -8.14876248e-02  1.37349395e-01 -8.17880913e-02
  9.27907812e-02  6.76093826e-03 -5.22928389e-02  6.02994188e-02
  8.28096711e-03 -1.05344042e-01 -1.02705751e-01  2.45275938e-02
 -1.18970611e-02  9.86759282e-02 -1.92870134e-02  9.71936577e-03
 -1.40249490e-01  1.61314103e-01 -4.55344879e-02  2.21929812e-02
  9.54108215e-02 -1.25028370e-02  2.89625007e-02  1.65818081e-02
 -2.34467852e-02 -7.88610081e-02  3.34242148e-03  4.43269879e-02
 -4.08419376e-02  6.06990416e-02  2.33916564e-02 -1.02773899e-02
  9.21596550e-02  9.90483805e-02  7.50525638e-03 -4.07725570e-03
 -6.93980047e-02 -3.50341946e-02 -8.79849597e-02 -4.10474223e-02
  4.55004698e-03  2.27073689e-01  1.37340472e-01  4.43856114e-02]
```

2. Vectorizing project_title

In [150]:

```
word2VecTitlesVectors = getWord2VecVectors(preProcessedProjectTitlesWithoutStopWords);
```

In [175]:

```
print("Shape of Word2Vec vectorization matrix of project titles: {},
{}".format(len(word2VecTitlesVectors), len(word2VecTitlesVectors[0])));
equalsBorder(70);
print("Sample title: ");
equalsBorder(70);
print(preProcessedProjectTitlesWithoutStopWords[0]);
equalsBorder(70);
print("Word2Vec vector of sample title: ");
equalsBorder(70);
print(word2VecTitlesVectors[0]);
```

```
Shape of Word2Vec vectorization matrix of project titles: 109248, 300
======================================================================
Sample title:
======================================================================
educational support english learners home
======================================================================
Word2Vec vector of sample title:
======================================================================
[-4.1285000e-02  4.4970000e-02  1.4283080e-01  1.9901860e-02
 -8.4519200e-02 -4.3207400e-01 -2.8496800e+00 -2.2953320e-01
  2.1736960e-01  3.4239600e-01 -7.5568200e-02  1.8077600e-01
  1.3998316e-01 -1.6401800e-01 -2.9812820e-01 -2.5030200e-01
  2.0420960e-01 -1.6882720e-01  6.5439800e-02 -1.6061000e-01
  2.2179020e-01  2.9944900e-01  2.7358000e-02 -8.8528800e-02
  1.5856400e-01  6.2905000e-02  2.0427440e-01 -1.9312560e-01
 -9.2904600e-02 -2.2050020e-01 -5.7761060e-01 -1.2101294e-01
  1.6846980e-01  2.8212460e-01 -1.8210120e-01  1.7754000e-02
  1.4805200e-01  4.1059000e-02  3.1145000e-02 -9.5658000e-02
 -9.6840000e-03  2.4896520e-01 -2.5047440e-01  7.7859000e-02
 -3.7512000e-03 -2.7071920e-01  2.5586200e-02  2.3205600e-01
  1.0154800e-01 -5.2259200e-01 -1.3211440e-01  1.1908300e-01
  2.7147196e-01  5.6135400e-02 -5.3140200e-02 -1.4937160e-01
 -1.0488160e-01  1.2059600e-01 -1.2639620e-01 -1.4316640e-01
 -2.2147600e-01 -1.9137800e-01  1.6595340e-01 -5.6078000e-02
  3.9884400e-02  1.0854760e-01  1.5552920e-01  7.8204600e-02
  9.5928000e-02 -6.2156000e-03 -1.1407312e-01  3.6862800e-02
 -8.7530020e-02 -4.7668000e-02 -2.3264200e-01 -6.1687200e-02
 -3.1690916e-01 -1.1851380e-01  1.4931240e-01 -7.7857200e-02
  1.8634840e-01 -4.6202100e-01  2.7096800e-01 -3.0512800e-02
 -2.1226400e-01 -1.5356200e-02  1.0844260e-01 -8.2669200e-02
  2.8918600e-01  1.3372960e-01 -8.3522800e-02  4.6474200e-02
```

```
 2.0703580e-01 -2.1937640e-01 -1.0252400e-01 -2.5177000e-01
-2.8408000e+00  1.6622880e-01  1.1216234e-01  2.0837920e-01
-1.5711600e-01 -1.9159400e-01 -1.4992160e-01 -2.7392820e-01
 3.4989140e-01  1.3991600e-01  1.6275200e-01  1.3887200e-01
 1.8212760e-01 -3.2218600e-02  4.3172000e-02  1.8323640e-01
 1.2295780e-01  4.4706600e-01  2.1688400e-02 -3.8988200e-02
-3.2467400e-01  3.8389160e-01 -1.4416560e-01  1.1117380e-01
-1.6218300e-01  1.3871928e-01  1.4305240e-01 -7.6173200e-02
 8.9476800e-02  2.6043820e-01  5.1114000e-02  1.0619800e-01
 1.5968840e-01  1.0530680e-01  8.6300000e-02  1.4667260e-01
 1.2320460e-02 -6.6124620e-02 -1.1017760e-01 -1.5091940e-01
 2.1297280e-01 -3.2808520e-01  1.4493194e-01  2.1848680e-01
-4.1809800e-03  8.5340000e-02 -1.2410789e-01 -2.2308140e-01
 8.8026000e-02  1.9555000e-01 -3.7981400e-02 -1.7720080e-01
 3.4328600e-01 -3.7459600e-01 -1.7268200e-01 -2.1554400e-01
-1.1533400e-01  9.9680000e-02 -1.9032980e-01  8.6249800e-02
 7.6682200e-02 -9.1090380e-02 -9.3714000e-02 -1.7333260e-01
 8.6429960e-02 -6.7933600e-02 -8.6470600e-02 -2.2431600e-01
-2.8319800e-01  1.0138200e-01 -2.8114320e-01 -1.1168240e-01
 2.1770560e-02 -1.3971160e-01  2.1795080e-01 -1.1995600e-01
-1.3166600e-02 -3.4848260e-01 -3.0102000e-02  2.3396200e-02
 2.8840000e-02  2.8763000e-01 -2.3679600e-02  1.1806440e-01
-3.2261460e-01  2.2622920e-01  1.9506400e-02  1.4363200e-01
-1.3668380e-01 -1.0521880e-01 -3.9385400e-03 -4.6388000e-02
-7.7493780e-02 -2.4700800e-02 -5.2006200e-02 -2.6299360e-01
-2.5607520e-01  2.1704520e-01  5.6336000e-02 -6.3474400e-02
-1.0400400e-01 -1.7901000e-01  2.0326180e-01 -2.8708740e-01
 1.0132000e-01 -1.6278080e-01  1.2441440e-01  3.2699820e-01
-4.8321600e-02 -3.6052800e-02  2.2539620e-01 -8.2764000e-03
 3.1087258e-01  2.4090500e-01 -9.9590000e-02  1.2362460e-01
 1.7440000e-03 -1.6117280e-01  7.4570000e-02  3.1281120e-02
-1.1758000e-02 -1.8464800e-02 -2.0872020e-01 -3.9510000e-03
-5.7714400e-01 -1.8090080e-01 -2.8288200e-01 -2.4662120e-01
-1.8806540e+00  4.4765400e-01 -2.9412700e-01 -1.7280000e-02
-3.1931600e-01 -1.9190500e-01 -1.1642000e-02  1.7475600e-01
 1.3068840e-01  1.1943000e-01 -1.7219524e-01  1.9224000e-02
 2.2620000e-01 -1.0821980e-01  1.3789060e-01  2.6989320e-01
-2.4364960e-01 -1.3650800e-01 -3.0984180e-01 -3.9546200e-02
-1.1410800e-01 -6.6744640e-02  1.6330620e-01 -4.0601000e-01
 9.3793000e-02 -8.3026800e-02  9.0567600e-02  3.1595600e-01
 1.6786620e-01  1.0099860e-01  3.5043600e-02  6.6221200e-02
-3.5907800e-02 -2.4589760e-01  2.6006800e-01 -8.0637000e-02
 1.5359624e-01 -1.1078680e-01 -5.6956400e-02  2.2253080e-01
 3.5808000e-02 -1.8873860e-01 -2.5032660e-01  3.6167400e-02
-2.2424700e-01  2.7863640e-01  2.2622600e-02  1.3753300e-01
-2.3369620e-01  2.8058040e-01  5.0818000e-02 -3.4805800e-02
 1.7916600e-01 -7.5374000e-02  7.1228900e-02  1.7556000e-01
-5.8004120e-01 -2.0522500e-01 -1.3367960e-01  1.3656000e-02
-2.9052200e-02  1.3698600e-02  1.1746340e-01 -2.3288400e-02
 2.7706200e-01  1.6106000e-01 -2.0183340e-01  5.7781800e-02
-2.0954400e-01 -1.4111260e-02 -3.1186860e-01 -2.9536360e-02
-1.7226500e-01  3.5709400e-01  2.9448200e-01  8.5600000e-05]
```

Tf-Idf Weighted Word2Vec Vectorization

1. Vectorizing project_essay

In [154]:

```
# Initializing tfidf vectorizer
tfIdfEssayTempVectorizer = TfidfVectorizer();
# Vectorizing preprocessed essays using tfidf vectorizer initialized above
tfIdfEssayTempVectorizer.fit(preProcessedEssaysWithoutStopWords);
# Saving dictionary in which each word is key and it's idf is value
tfIdfEssayDictionary = dict(zip(tfIdfEssayTempVectorizer.get_feature_names(),
list(tfIdfEssayTempVectorizer.idf_)));
# Creating set of all unique words used by tfidf vectorizer
tfIdfEssayWords = set(tfIdfEssayTempVectorizer.get_feature_names());
```

In [159]:

```
# Creating list to save tf-idf weighted vectors of essays
tfIdfWeightedWord2VecEssaysVectors = [];
```

```
# Iterating over each essay
for essay in tqdm(preProcessedEssaysWithoutStopWords):
    # Sum of tf-idf values of all words in a particular essay
    cumulativeSumTfIdfWeightOfEssay = 0;
    # Tf-Idf weighted word2vec vector of a particular essay
    tfIdfWeightedWord2VecEssayVector = np.zeros(300);
    # Splitting essay into list of words
    splittedEssay = essay.split();
    # Iterating over each word
    for word in splittedEssay:
        # Checking if word is in glove words and set of words used by tfIdf essay vectorizer
        if (word in gloveWords) and (word in tfIdfEssayWords):
            # Tf-Idf value of particular word in essay
            tfIdfValueWord = tfIdfEssayDictionary[word] * (essay.count(word) / len(splittedEssay));
            # Making tf-idf weighted word2vec
            tfIdfWeightedWord2VecEssayVector += tfIdfValueWord * gloveModel[word];
            # Summing tf-idf weight of word to cumulative sum
            cumulativeSumTfIdfWeightOfEssay += tfIdfValueWord;
    if cumulativeSumTfIdfWeightOfEssay != 0:
        # Taking average of sum of vectors with tf-idf cumulative sum
        tfIdfWeightedWord2VecEssayVector = tfIdfWeightedWord2VecEssayVector /
cumulativeSumTfIdfWeightOfEssay;
    # Appending the above calculated tf-idf weighted vector of particular essay to list of vectors
of essays
    tfIdfWeightedWord2VecEssaysVectors.append(tfIdfWeightedWord2VecEssayVector);
```

In [174]:

```
print("Shape of Tf-Idf weighted Word2Vec vectorization matrix of project essays: {}, {}".format(le
n(tfIdfWeightedWord2VecEssaysVectors), len(tfIdfWeightedWord2VecEssaysVectors[0])));
equalsBorder(70);
print("Sample Essay: ");
equalsBorder(70);
print(preProcessedEssaysWithoutStopWords[0]);
equalsBorder(70);
print("Tf-Idf Weighted Word2Vec vector of sample essay: ");
equalsBorder(70);
print(tfIdfWeightedWord2VecEssaysVectors[0]);
```

```
Shape of Tf-Idf weighted Word2Vec vectorization matrix of project essays: 109248, 300
======================================================================
Sample Essay:
======================================================================
students english learners working english second third languages melting pot refugees immigrants n
ative born americans bringing gift language school 24 languages represented english learner progra
m students every level mastery also 40 countries represented families within school student brings
wealth knowledge experiences us open eyes new cultures beliefs respect limits language limits worl
d ludwig wittgenstein english learner strong support system home begs resources many times parents
learning read speak english along side children sometimes creates barriers parents able help child
learn phonetics letter recognition reading skills providing dvd players students able continue mas
tery english language even no one home able assist families students within level 1 proficiency st
atus offered part program educational videos specially chosen english learner teacher sent home re
gularly watch videos help child develop early reading skills parents not access dvd player
opportunity check dvd player use year plan use videos educational dvd years come el students nanna
n
======================================================================
Tf-Idf Weighted Word2Vec vector of sample essay:
======================================================================
[-5.37582850e-02  7.68689598e-02  7.85741822e-02  4.38958976e-02
 -8.56874440e-02 -1.20832331e-01 -2.68120986e+00  7.17018732e-02
  1.03799206e-04 -5.17255299e-03 -2.67529751e-02  7.40185988e-02
  1.36881934e-01 -8.62706493e-02 -6.35020145e-02 -8.44084597e-02
  1.27523921e-01 -1.77105602e-01  3.68451284e-02 -5.74471880e-02
  1.86477259e-01  9.28786009e-02 -9.73137896e-02 -1.15230456e-02
  4.41962185e-02 -9.32894883e-02  1.11912943e-01 -1.17540961e-01
 -1.22150893e-01 -9.14028838e-02 -1.73918944e-01 -4.54143189e-02
 -7.82036060e-02  3.05617633e-01 -8.71850266e-02  6.31466708e-03
  1.15683161e-01  1.71477594e-02 -5.52983597e-02  9.08989585e-02
 -3.89808292e-04  1.97696142e-01 -4.08078376e-01 -5.39990199e-02
 -1.20129600e-02 -1.12456389e-01  2.92046345e-02  1.37924729e-01
  2.83465620e-02 -2.26817169e-01 -2.29639267e-02  6.94257143e-03
  5.80535394e-02  2.86454339e-02 -7.51508216e-02 -6.21569354e-02
 -1.41805544e-01  2.78707358e-02 -1.63165999e-01 -1.29716251e-01
 -5.67625355e-02 -8.59507500e-02  3.54019902e-01 -4.96274469e-02
```

```
-6.88414062e-02  1.58623510e-01  1.24798600e-01  4.29711440e-02
 7.82814323e-02 -1.73260116e-02 -1.23679491e-01  1.47617250e-01
 4.27083617e-02 -1.16531047e-01 -1.27122530e-01 -5.93638332e-03
-1.99224414e-01 -8.66160391e-02  2.47701354e-01  1.61218205e-02
 3.56880345e-02 -3.71320273e-01  2.65501745e-01 -4.56454865e-02
-7.85433814e-02 -5.99177835e-02  4.42212779e-02 -8.20739267e-02
 2.14031939e-01  2.42131497e-02 -1.34069697e-01  7.15871686e-03
 4.00667270e-02 -6.75881497e-02 -7.07967357e-02 -2.15984749e-02
-2.09734597e+00  1.02300477e-01  6.61169899e-02  5.70146517e-02
-1.91302495e-01 -1.38114014e-01 -1.10709961e-01 -1.66994098e-01
 9.17800823e-02  1.35327093e-01  2.20333244e-02 -3.83844831e-02
 2.57206511e-02 -5.54503565e-02 -3.41973653e-03  1.99777588e-02
 4.85050396e-02  2.13190534e-01  4.64281665e-02  6.51171751e-02
-5.80015838e-02  1.19900386e-01  1.18803830e-01  7.05550873e-02
-1.87330886e-01  1.41219129e-01  1.33569574e-01  1.00530000e-01
 4.14498415e-02  1.39860952e-01 -7.95709830e-02  9.70242332e-02
 1.07442882e-01  9.00794808e-02  7.47745032e-02  4.18772282e-02
-7.10347826e-03 -7.62379756e-03 -7.31715828e-02 -1.16370646e-01
 2.82271708e-01 -5.30885621e-02  4.51472249e-02  2.61376253e-01
 1.29080066e-02  3.96843846e-02  1.04430681e-01 -1.30495811e-01
-1.17999239e-01 -1.02810089e-01 -6.52713784e-02 -1.81350799e-01
 1.55415740e-01 -4.43517889e-02 -8.34350788e-02 -1.31445407e-01
-8.87524029e-02 -1.15321245e-02  8.67587067e-03  3.55646447e-02
-4.32365925e-02  2.44285859e-03  2.73165854e-02 -1.91651165e-01
 6.70942750e-03  1.45533103e-02 -5.95191056e-02 -9.78336553e-02
-4.61200683e-02  1.04017495e-02 -1.68129330e-01 -5.53455289e-02
-1.95353920e-02 -3.24088827e-03  9.94121739e-02 -2.20584067e-02
 1.36190091e-02 -3.13014669e-01  4.46748268e-02  6.11251996e-02
-5.59088914e-02  8.07071841e-02 -7.80920682e-02  1.05535003e-02
-8.49705076e-02  1.87800458e-01 -5.53305425e-02 -4.05296946e-02
-1.68105655e-02 -9.64697267e-02 -1.00114054e-01 -1.25303984e-01
-6.77861115e-02  1.38106300e-02  4.97948787e-02 -1.04414463e-01
 3.12147536e-03 -2.46650333e-02  1.56250756e-02 -3.41987984e-02
 2.90197738e-02 -1.30795750e-01  1.71425098e-01 -1.33199913e-01
-4.35452619e-02 -1.52841321e-01  3.37717104e-02  2.11400042e-01
-1.08493100e-01  6.64905827e-02  4.45687503e-02 -3.38898797e-03
 1.47302984e-01  3.10931848e-02  6.94873935e-03 -3.79090162e-02
 3.97055902e-02 -3.12563998e-02  2.99815273e-02 -9.30892230e-03
-3.37192802e-02 -7.79667288e-02  4.20509297e-02  4.33535394e-02
-2.38238094e-01 -4.11188300e-02 -1.93930088e-01  1.15012485e-01
-2.14605373e+00  1.36975648e-01 -1.79026305e-01 -1.42630498e-01
-1.37558424e-01 -1.55433436e-01 -6.96701214e-02  1.05328488e-01
 3.43486342e-02 -2.37676310e-03 -6.80980842e-02 -1.92470331e-01
 1.54727348e-01 -7.47455695e-02 -1.58054203e-02  3.33369549e-02
-1.70510752e-01 -5.74331307e-02 -2.38994456e-01  5.64188931e-02
-8.55051184e-02 -5.52984572e-02 -5.00408589e-02 -6.81572658e-02
 5.15848477e-03 -3.58487773e-02  7.00056842e-02  1.33127170e-01
 5.57938159e-02  1.03106840e-01  4.18598320e-02 -2.78162076e-03
 8.83131944e-02 -1.31482831e-01  1.34875022e-01 -8.31772344e-02
 1.62319378e-01  9.25839856e-02 -7.07548194e-02  1.74355644e-01
 1.53106818e-02 -1.74504449e-01 -5.39158255e-02 -1.16968555e-02
-1.37824311e-01  1.07713713e-01  4.48548015e-02  1.07272158e-01
-1.59084558e-01  1.94342786e-01 -4.73514319e-02 -4.87250503e-02
 2.82023483e-02 -4.18474756e-02  8.04397595e-02 -3.34005484e-02
-1.00808502e-01 -1.15380334e-01  7.05894205e-02  2.92052920e-02
-5.72604859e-02 -7.39274088e-03  1.44106517e-02 -2.64282237e-02
 2.31512689e-01  1.50161666e-01 -5.21462274e-02 -1.00796916e-02
-4.47392305e-02  4.83958092e-02 -2.21927272e-01 -9.69846899e-02
-5.91211767e-03  2.52508756e-01  1.08677704e-01  5.05047869e-02]
```

2. Vectorizing project_title

In [342]:

```
# Initializing tfidf vectorizer
tfIdfTitleTempVectorizer = TfidfVectorizer();
# Vectorizing preprocessed titles using tfidf vectorizer initialized above
tfIdfTitleTempVectorizer.fit(preProcessedProjectTitlesWithoutStopWords);
# Saving dictionary in which each word is key and it's idf is value
tfIdfTitleDictionary = dict(zip(tfIdfTitleTempVectorizer.get_feature_names(),
list(tfIdfTitleTempVectorizer.idf_)));
# Creating set of all unique words used by tfidf vectorizer
tfIdfTitleWords = set(tfIdfTitleTempVectorizer.get_feature_names());
```

In [343]:

```
# Creating list to save tf-idf weighted vectors of project titles
tfIdfWeightedWord2VecTitlesVectors = [];
# Iterating over each title
for title in tqdm(preProcessedProjectTitlesWithoutStopWords):
    # Sum of tf-idf values of all words in a particular project title
    cumulativeSumTfIdfWeightOfTitle = 0;
    # Tf-Idf weighted word2vec vector of a particular project title
    tfIdfWeightedWord2VecTitleVector = np.zeros(300);
    # Splitting title into list of words
    splittedTitle = title.split();
    # Iterating over each word
    for word in splittedTitle:
        # Checking if word is in glove words and set of words used by tfIdf title vectorizer
        if (word in gloveWords) and (word in tfIdfTitleWords):
            # Tf-Idf value of particular word in title
            tfIdfValueWord = tfIdfTitleDictionary[word] * (title.count(word) / len(splittedTitle));
            # Making tf-idf weighted word2vec
            tfIdfWeightedWord2VecTitleVector += tfIdfValueWord * gloveModel[word];
            # Summing tf-idf weight of word to cumulative sum
            cumulativeSumTfIdfWeightOfTitle += tfIdfValueWord;
    if cumulativeSumTfIdfWeightOfTitle != 0:
        # Taking average of sum of vectors with tf-idf cumulative sum
        tfIdfWeightedWord2VecTitleVector = tfIdfWeightedWord2VecTitleVector /
cumulativeSumTfIdfWeightOfTitle;
    # Appending the above calculated tf-idf weighted vector of particular title to list of vectors
of project titles
    tfIdfWeightedWord2VecTitlesVectors.append(tfIdfWeightedWord2VecTitleVector);
```

In [344]:

```
print("Shape of Tf-Idf weighted Word2Vec vectorization matrix of project titles: {}, {}".format(le
n(tfIdfWeightedWord2VecTitlesVectors), len(tfIdfWeightedWord2VecTitlesVectors[0])));
equalsBorder(70);
print("Sample Title: ");
equalsBorder(70);
print(preProcessedProjectTitlesWithoutStopWords[0]);
equalsBorder(70);
print("Tf-Idf Weighted Word2Vec vector of sample title: ");
equalsBorder(70);
print(tfIdfWeightedWord2VecTitlesVectors[0]);
```

```
Shape of Tf-Idf weighted Word2Vec vectorization matrix of project titles: 109248, 300
======================================================================
Sample Title:
======================================================================
educational support english learners home
======================================================================
Tf-Idf Weighted Word2Vec vector of sample title:
======================================================================
[-3.23904891e-02  5.58064810e-02  1.32666911e-01  3.84227573e-02
 -6.71984492e-02 -4.30940397e-01 -2.84607947e+00 -2.45905055e-01
  1.96794858e-01  3.19604663e-01 -6.12568872e-02  1.59218099e-01
  1.25129027e-01 -1.67580327e-01 -2.82644062e-01 -2.47555536e-01
  2.18304104e-01 -1.57431101e-01  7.66481545e-02 -1.61436633e-01
  2.38451267e-01  2.86712258e-01  2.70730890e-02 -9.74962294e-02
  1.67511144e-01  7.18131102e-02  1.82846112e-01 -1.96778087e-01
 -8.19948978e-02 -2.25877630e-01 -5.54573752e-01 -1.28462870e-01
  1.61012606e-01  2.94412658e-01 -1.63196910e-01 -1.23217523e-02
  1.37466355e-01  4.45437696e-02  4.65691769e-02 -1.17867965e-01
 -2.41502151e-03  2.24350668e-01 -2.51274676e-01  8.29431360e-02
 -1.65996673e-02 -2.47747576e-01  1.45110611e-03  2.37117949e-01
  9.71345150e-02 -5.13516477e-01 -1.40296688e-01  1.42775548e-01
  2.89949805e-01  6.49771690e-02 -3.41581088e-02 -1.58076306e-01
 -1.07731741e-01  7.59015357e-02 -1.21511682e-01 -1.16519972e-01
 -2.27321940e-01 -1.63525257e-01  1.80860125e-01 -4.17314689e-02
  4.60171896e-02  1.00024674e-01  1.54588362e-01  8.25394911e-02
  7.45768118e-02 -1.80240543e-02 -1.22956246e-01 -4.97450371e-03
 -8.06577406e-02 -5.00614538e-02 -2.15836210e-01 -5.89271531e-02
 -3.26363335e-01 -1.32706775e-01  1.61236199e-01 -1.25038790e-01
  1.96493846e-01 -4.95095193e-01  2.34765396e-01 -4.44646606e-02
 -2.04266125e-01 -3.21415735e-02  8.48111983e-02 -7.27603472e-02
  2.79183660e-01  1.18968262e-01 -7.43300594e-02  6.34587771e-02
  1.99863053e-01 -2.13382053e-01 -1.01221319e-01 -2.49884070e-01
```

```
 -2.92249478e+00  1.60273141e-01  7.74579728e-02  1.85323805e-01
 -1.33255909e-01 -2.00013519e-01 -1.31974722e-01 -2.62288530e-01
  3.54852941e-01  1.18537924e-01  1.62207829e-01  1.24436802e-01
  1.98867481e-01 -4.87526944e-03  3.00886908e-02  2.09330567e-01
  1.17189984e-01  3.94887340e-01  2.52941492e-02 -5.13348554e-02
 -2.91140828e-01  4.06939567e-01 -1.70319175e-01  1.17651155e-01
 -1.66813086e-01  1.53049826e-01  1.41255472e-01 -8.10785736e-02
  9.57549943e-02  2.73610111e-01  5.85622995e-02  7.91410001e-02
  1.47619459e-01  9.75521835e-02  6.74487028e-02  1.53125504e-01
  2.02791106e-02 -5.59403852e-02 -1.02109913e-01 -1.22913427e-01
  1.99873969e-01 -3.21872719e-01  1.38343165e-01  2.17196179e-01
  4.95201760e-03  8.52128333e-02 -1.45880901e-01 -2.10862397e-01
  1.20343357e-01  2.15598061e-01 -1.14038072e-02 -1.72172799e-01
  3.24157324e-01 -3.82818101e-01 -1.87580283e-01 -2.00827204e-01
 -1.41863370e-01  9.63016678e-02 -2.01659119e-01  6.74342164e-02
  7.12185747e-02 -1.04314039e-01 -9.08169483e-02 -1.63495605e-01
  9.68230169e-02 -5.01176209e-02 -8.34015616e-02 -1.88998660e-01
 -2.84065057e-01  1.16975197e-01 -2.80836800e-01 -9.33191327e-02
  3.79583269e-02 -1.22755412e-01  2.30408258e-01 -1.31968890e-01
  9.72824714e-03 -3.44272546e-01 -2.09522211e-03  2.45944018e-02
  2.94077607e-02  2.67568157e-01 -2.69460269e-02  1.25412311e-01
 -3.47031083e-01  2.09328241e-01  1.25385338e-02  1.55654760e-01
 -1.41368915e-01 -1.01749781e-01 -4.77312036e-04 -4.82325465e-02
 -7.15727478e-02 -3.63658602e-02 -4.33504397e-02 -2.71410315e-01
 -2.40079853e-01  2.01171435e-01  6.39005674e-02 -4.86787485e-02
 -1.48623863e-01 -1.72130906e-01  1.97761227e-01 -3.13043504e-01
  1.07772898e-01 -1.54518908e-01  1.31855435e-01  3.39703669e-01
 -4.51652340e-02 -4.05998340e-02  2.03610454e-01  8.84982054e-03
  3.05974297e-01  2.54736700e-01 -1.06925907e-01  1.27066655e-01
 -1.88835779e-02 -1.56632041e-01  8.45142200e-02  5.70681135e-02
  1.01119358e-02 -6.62387316e-03 -2.18552410e-01  1.20985419e-02
 -5.54006219e-01 -1.72367117e-01 -2.90325016e-01 -2.34816399e-01
 -1.94243114e+00  4.36715446e-01 -2.80713863e-01 -6.33991309e-03
 -2.90035778e-01 -1.98732349e-01  2.96737137e-02  1.50873684e-01
  1.16943997e-01  1.39741722e-01 -1.82238609e-01  4.09714520e-02
  2.37176600e-01 -1.24515116e-01  1.41648743e-01  2.64206287e-01
 -2.40551078e-01 -1.40415333e-01 -2.92432371e-01 -3.03761027e-02
 -9.90320454e-02 -8.43648662e-02  1.81116706e-01 -4.05719699e-01
  1.22898740e-01 -8.80109292e-02  1.09543672e-01  2.96110858e-01
  1.85027885e-01  9.14976115e-02  9.63416424e-03  5.50340717e-02
 -2.59328007e-02 -2.43942768e-01  2.54260096e-01 -1.03280950e-01
  1.56799018e-01 -9.58635926e-02 -4.31948365e-02  2.01228907e-01
  5.20033765e-02 -2.08030399e-01 -2.49149283e-01  3.11752465e-02
 -2.39410711e-01  2.54421815e-01  3.50420005e-02  1.31625993e-01
 -2.19027956e-01  2.75093693e-01  4.31276229e-02 -6.89266192e-02
  1.80694153e-01 -9.77254221e-02  6.52789959e-02  1.81468103e-01
 -5.79288980e-01 -1.91501478e-01 -1.43298895e-01  1.56769073e-02
 -2.28584041e-02 -7.96762354e-03  1.38764109e-01 -2.67804890e-02
  3.02808634e-01  1.63688874e-01 -1.98263925e-01  8.94007093e-02
 -2.01132765e-01  8.29230669e-03 -3.17426319e-01 -4.07929287e-02
 -1.63872993e-01  3.69860278e-01  2.90009047e-01  4.56005599e-02]
```

Vectorizing numerical features

1. Vectorizing price

In [183]:

```
# Standardizing the price data using StandardScaler(Uses mean and std for standardization)
priceScaler = StandardScaler();
priceScaler.fit(projectsData['price'].values.reshape(-1, 1));
priceStandardized = priceScaler.transform(projectsData['price'].values.reshape(-1, 1));
```

In [193]:

```
print("Shape of standardized matrix of prices: ", priceStandardized.shape);
equalsBorder(70);
print("Sample original prices: ");
equalsBorder(70);
print(projectsData['price'].values[0:5]);
print("Sample standardized prices: ");
equalsBorder(70);
```

```
print(priceStandardized[0:5]);
```

```
Shape of standardized matrix of prices:  (109245, 1)
==========================================================================
Sample original prices:
==========================================================================
[154.6  299.   516.85 232.9   67.98]
Sample standardized prices:
==========================================================================
[[-0.39052147]
 [ 0.00240752]
 [ 0.5952024 ]
 [-0.17745817]
 [-0.62622444]]
```

2. Vectorizing quantity

In [185]:

```
# Standardizing the quantity data using StandardScaler(Uses mean and std for standardization)
quantityScaler = StandardScaler();
quantityScaler.fit(projectsData['quantity'].values.reshape(-1, 1));
quantityStandardized = quantityScaler.transform(projectsData['quantity'].values.reshape(-1, 1));
```

In [194]:

```
print("Shape of standardized matrix of quantities: ", quantityStandardized.shape);
equalsBorder(70);
print("Sample original quantities: ");
equalsBorder(70);
print(projectsData['quantity'].values[0:5]);
print("Sample standardized quantities: ");
equalsBorder(70);
print(quantityStandardized[0:5]);
```

```
Shape of standardized matrix of quantities:  (109245, 1)
==========================================================================
Sample original quantities:
==========================================================================
[23  1 22  4  4]
Sample standardized quantities:
==========================================================================
[[ 0.23045805]
 [-0.6097785 ]
 [ 0.19226548]
 [-0.49520079]
 [-0.49520079]]
```

3. Vectorizing teacher_number_of_previously_posted_projects

In [196]:

```
# Standardizing the teacher_number_of_previously_posted_projects data using StandardScaler(Uses me
an and std for standardization)
previouslyPostedScaler = StandardScaler();
previouslyPostedScaler.fit(projectsData['teacher_number_of_previously_posted_projects'].values.res
hape(-1, 1));
previouslyPostedStandardized =
previouslyPostedScaler.transform(projectsData['teacher_number_of_previously_posted_projects'].valu
es.reshape(-1, 1));
```

In [197]:

```
print("Shape of standardized matrix of teacher_number_of_previously_posted_projects: ",
previouslyPostedStandardized.shape);
equalsBorder(70);
print("Sample original quantities: ");
equalsBorder(70);
print(projectsData['teacher_number_of_previously_posted_projects'].values[0:5]);
print("Sample standardized teacher_number_of_previously_posted_projects: ");
```

```
print("Sample standardized teacher_number_of_previously_posted_projects: ");
equalsBorder(70);
print(previouslyPostedStandardized[0:5]);
```

```
Shape of standardized matrix of teacher_number_of_previously_posted_projects:  (109245, 1)
======================================================================
Sample original quantities:
======================================================================
[0 7 1 4 1]
Sample standardized teacher_number_of_previously_posted_projects:
======================================================================
[[-0.40153083]
 [-0.14952695]
 [-0.36553028]
 [-0.25752861]
 [-0.36553028]]
```

Taking 6k points(to avoid memory errors)

In [345]:

```
numberOfPoints = 6000;
# Categorical data
categoriesVectorsSub = categoriesVectors[0:numberOfPoints];
subCategoriesVectorsSub = subCategoriesVectors[0:numberOfPoints];
teacherPrefixVectorsSub = teacherPrefixVectors[0:numberOfPoints];
schoolStateVectorsSub = schoolStateVectors[0:numberOfPoints];
projectGradeVectorsSub = projectGradeVectors[0:numberOfPoints];

# Text data
bowEssayModelSub = bowEssayModel[0:numberOfPoints];
bowTitleModelSub = bowTitleModel[0:numberOfPoints];
tfIdfEssayModelSub = tfIdfEssayModel[0:numberOfPoints];
tfIdfTitleModelSub = tfIdfTitleModel[0:numberOfPoints];
word2VecEssaysVectorsSub = word2VecEssaysVectors[0:numberOfPoints];
word2VecTitlesVectorsSub = word2VecTitlesVectors[0:numberOfPoints];
tfIdfWeightedWord2VecEssaysVectorsSub = tfIdfWeightedWord2VecEssaysVectors[0:numberOfPoints];
tfIdfWeightedWord2VecTitlesVectorsSub = tfIdfWeightedWord2VecTitlesVectors[0:numberOfPoints];

# Numerical data
priceStandardizedSub = priceStandardized[0:numberOfPoints];
quantityStandardizedSub = quantityStandardized[0:numberOfPoints];
previouslyPostedStandardizedSub = previouslyPostedStandardized[0:numberOfPoints];
```

In [241]:

```
classesDataSub = projectsData['project_is_approved'][0:numberOfPoints].values
```

In [242]:

```
classesDataSub.shape
```

Out[242]:

```
(6000,)
```

Data Visualization using T-SNE

Classification using data merged with bag of words vectorized title and all considered categorical, numerical features

In [243]:

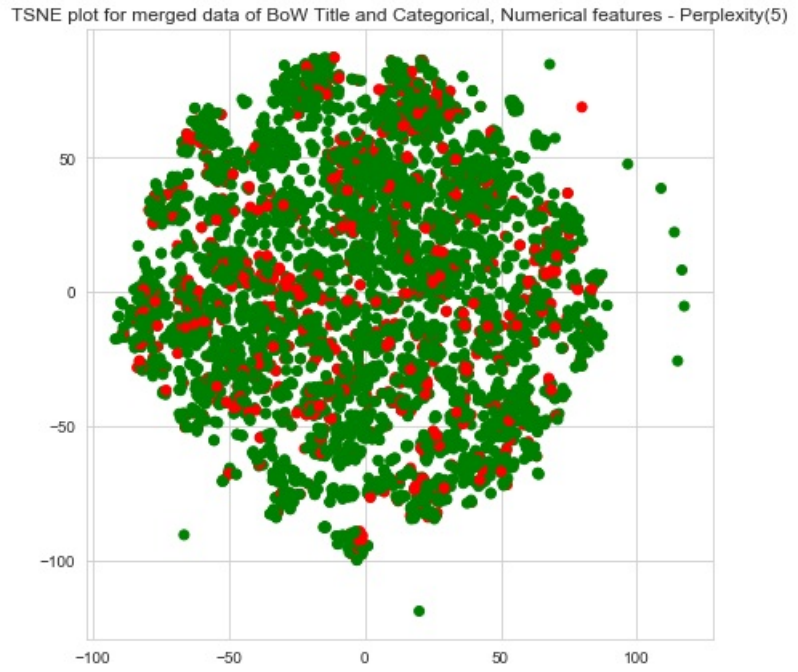
```
bowTitleAndOthers = hstack((bowTitleModelSub, categoriesVectorsSub, subCategoriesVectorsSub,
teacherPrefixVectorsSub, schoolStateVectorsSub, projectGradeVectorsSub, priceStandardizedSub,
previouslyPostedStandardizedSub));
bowTitleAndOthers.shape
```

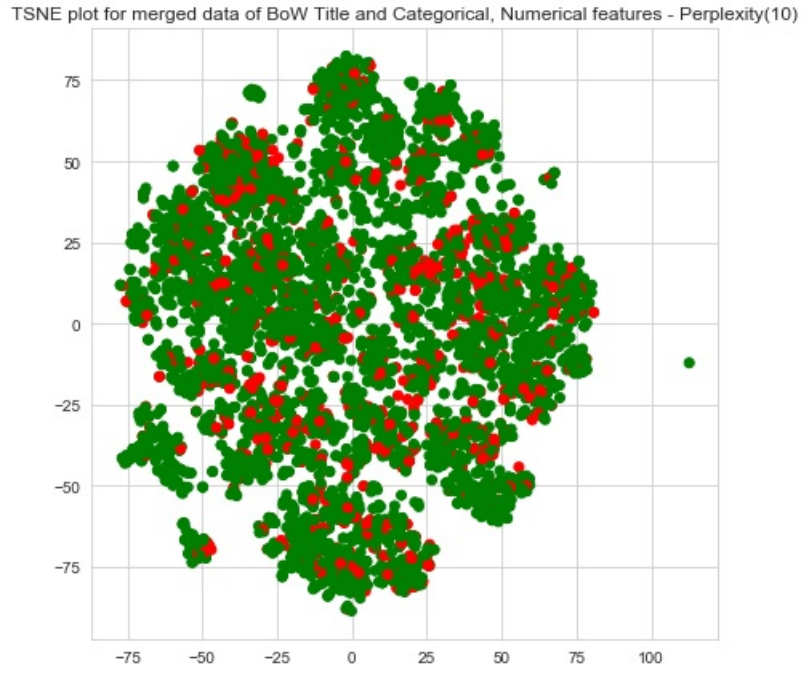
Out[243]:

(6000, 1875)

In [244]:

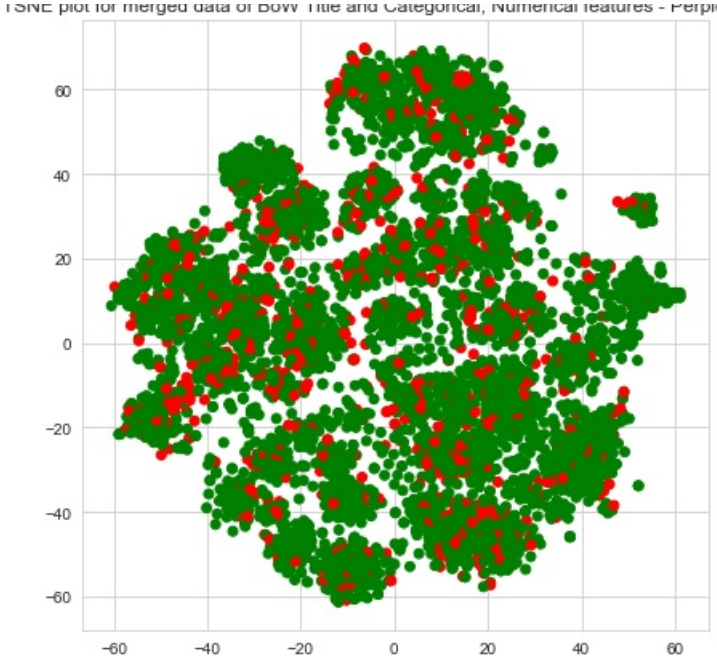
```
perplexityValues = [5, 10, 30, 50, 80, 100]
for perplexityValue in perplexityValues:
    tsne = TSNE(n_components = 2, perplexity = perplexityValue, learning_rate = 200);
    bowTitleAndOthersEmbedded = tsne.fit_transform(bowTitleAndOthers.toarray());
    bowTitleAndOthersTsneData = np.hstack((bowTitleAndOthersEmbedded, classesDataSub.reshape(-1, 1)
));
    bowTitleAndOthersTsneDataFrame = pd.DataFrame(bowTitleAndOthersTsneData, columns = ['Dimension1
', 'Dimension2', 'Class']);
    colors = {0.0:'red', 1.0:'green'}
    plt.title("TSNE plot for merged data of BoW Title and Categorical, Numerical features - Perple
xity({})".format(perplexityValue));
    plt.scatter(bowTitleAndOthersTsneDataFrame['Dimension1'],
bowTitleAndOthersTsneDataFrame['Dimension2'], c = bowTitleAndOthersTsneDataFrame['Class'].apply(la
mbda x: colors[x]));
    plt.show();
```



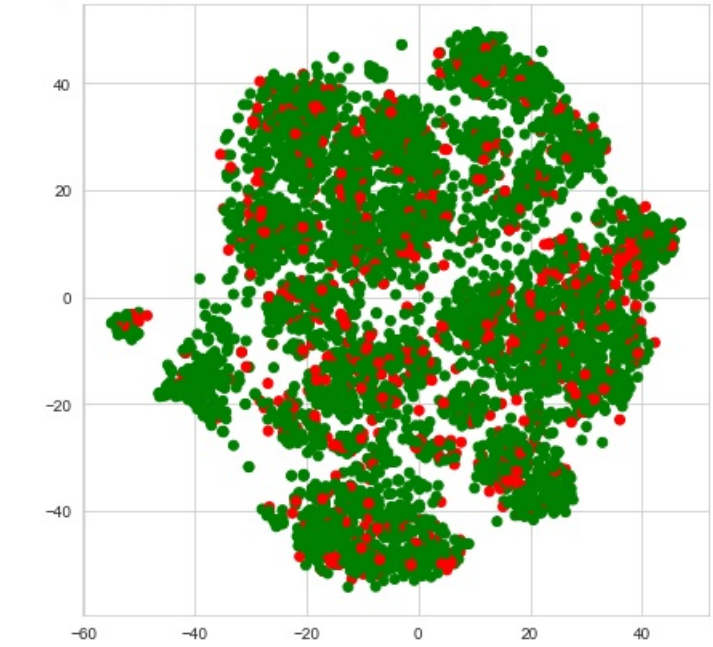


TSNE plot for merged data of BoW Title and Categorical, Numerical features - Perplexity(30)

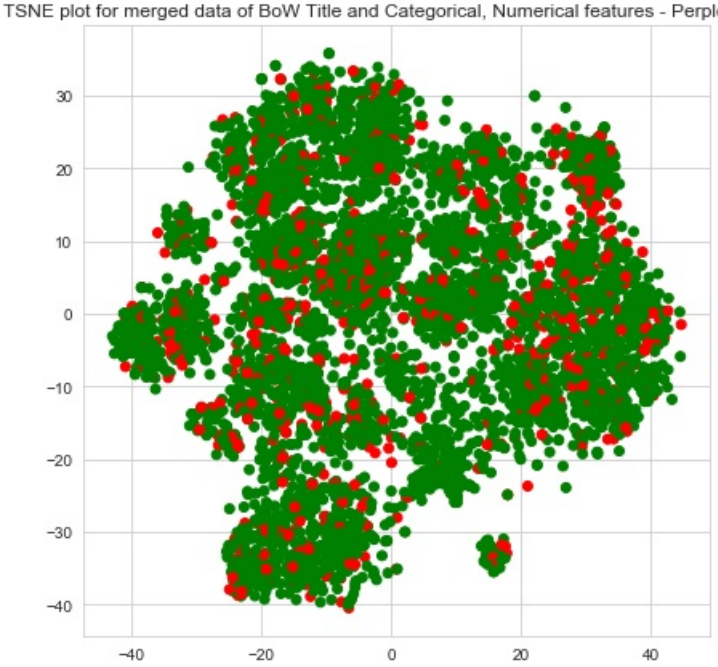
TSNE plot for merged data of BoW Title and Categorical, Numerical features - Perplexity(30)

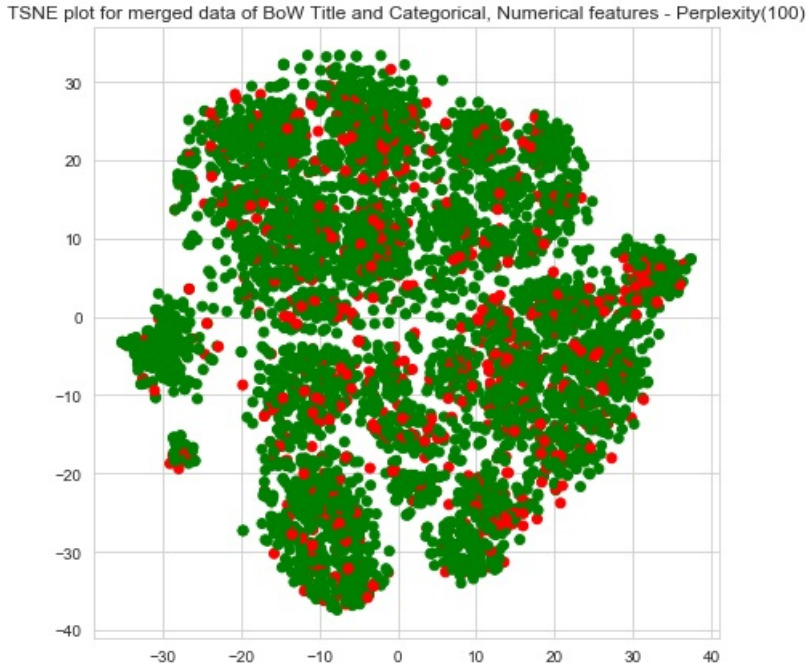


TSNE plot for merged data of BoW Title and Categorical, Numerical features - Perplexity(50)



TSNE plot for merged data of BoW Title and Categorical, Numerical features - Perplexity(80)





Classification using data merged with Tf-Idf vectorized title and all considered categorical, numerical features

In [245]:

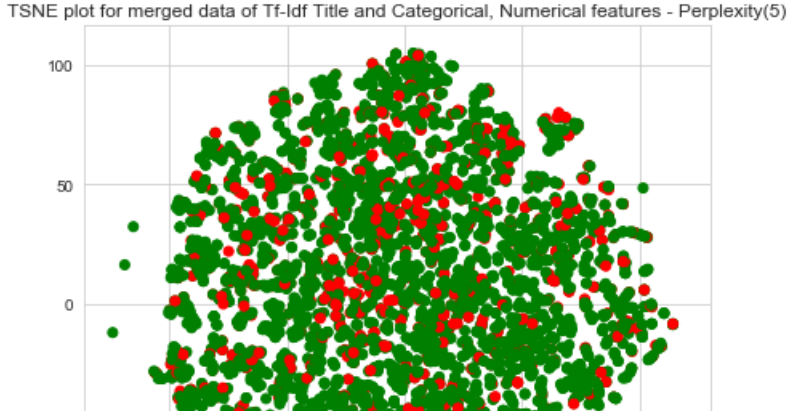
```
tfIdfTitleAndOthers = hstack((tfIdfTitleModelSub, categoriesVectorsSub, subCategoriesVectorsSub,
teacherPrefixVectorsSub, schoolStateVectorsSub, projectGradeVectorsSub, priceStandardizedSub,
previouslyPostedStandardizedSub));
tfIdfTitleAndOthers.shape
```

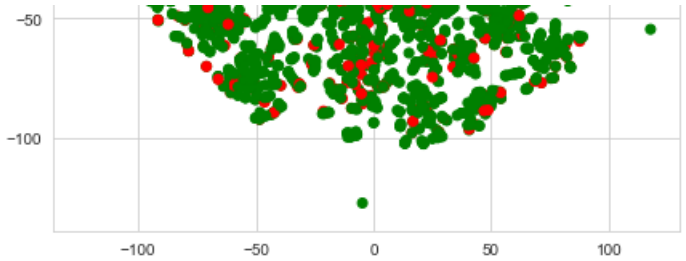
Out[245]:

```
(6000, 1875)
```

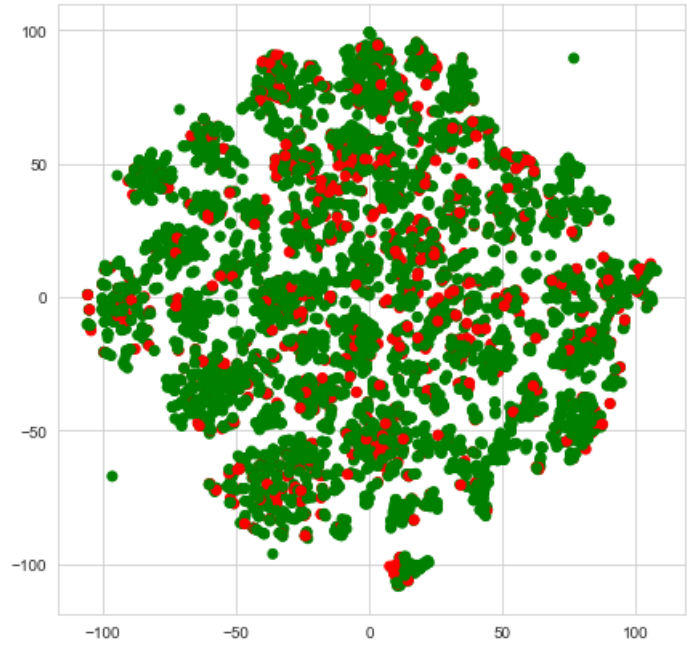
In [246]:

```
perplexityValues = [5, 10, 30, 50, 80, 100]
for perplexityValue in perplexityValues:
    tsne = TSNE(n_components = 2, perplexity = perplexityValue, learning_rate = 200);
    tfIdfTitleAndOthersEmbedded = tsne.fit_transform(tfIdfTitleAndOthers.toarray());
    tfIdfTitleAndOthersTsneData = np.hstack((tfIdfTitleAndOthersEmbedded, classesDataSub.reshape(-1
, 1)));
    tfIdfTitleAndOthersTsneDataFrame = pd.DataFrame(tfIdfTitleAndOthersTsneData, columns =
['Dimension1', 'Dimension2', 'Class']);
    colors = {0.0:'red', 1.0:'green'}
    plt.title("TSNE plot for merged data of Tf-Idf Title and Categorical, Numerical features - Per
plexity({})".format(perplexityValue));
    plt.scatter(tfIdfTitleAndOthersTsneDataFrame['Dimension1'], tfIdfTitleAndOthersTsneDataFrame['
Dimension2'], c = tfIdfTitleAndOthersTsneDataFrame['Class'].apply(lambda x: colors[x]));
    plt.show();
```

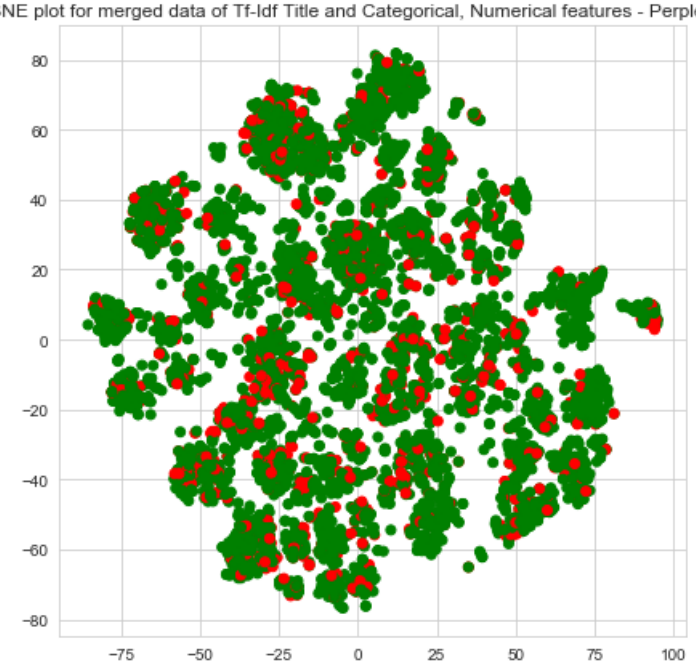




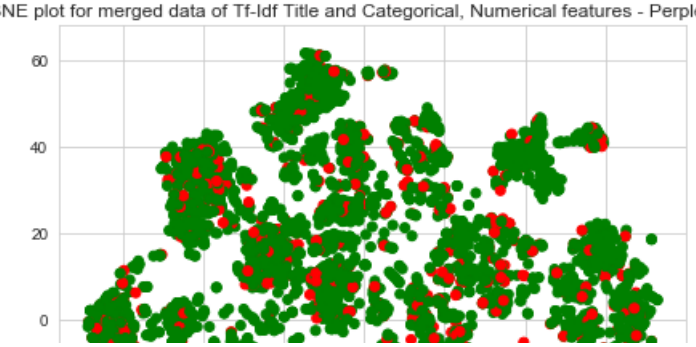
TSNE plot for merged data of Tf-Idf Title and Categorical, Numerical features - Perplexity(10)

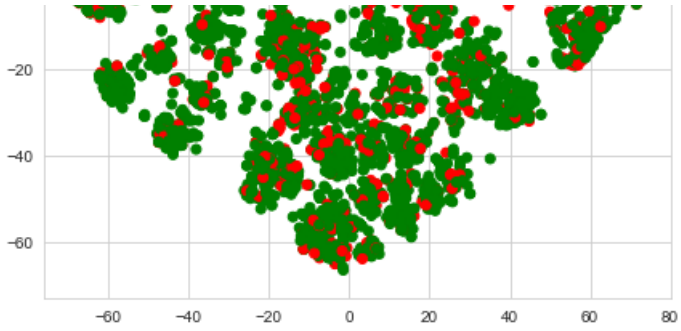


TSNE plot for merged data of Tf-Idf Title and Categorical, Numerical features - Perplexity(30)

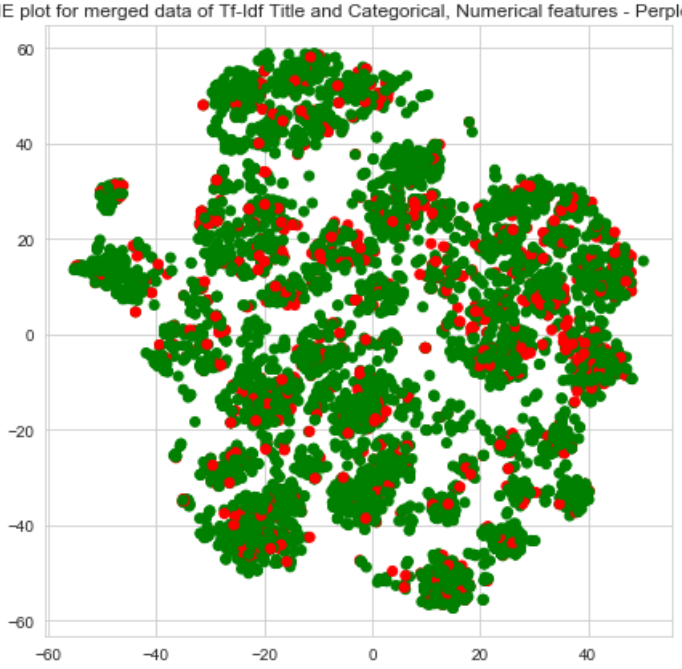


TSNE plot for merged data of Tf-Idf Title and Categorical, Numerical features - Perplexity(50)

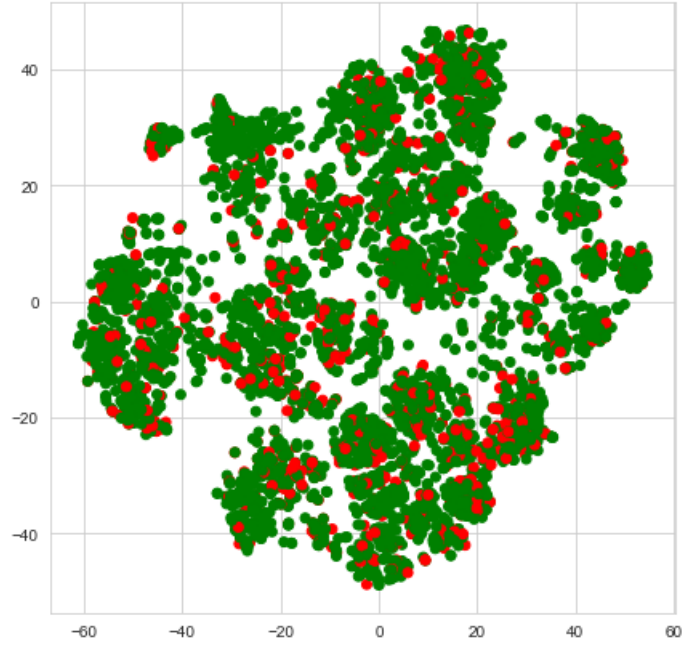




TSNE plot for merged data of Tf-Idf Title and Categorical, Numerical features - Perplexity(80)



TSNE plot for merged data of Tf-Idf Title and Categorical, Numerical features - Perplexity(100)



Classification using data merged with Average Word2Vec vectorized title and all considered categorical, numerical features

In [247]:

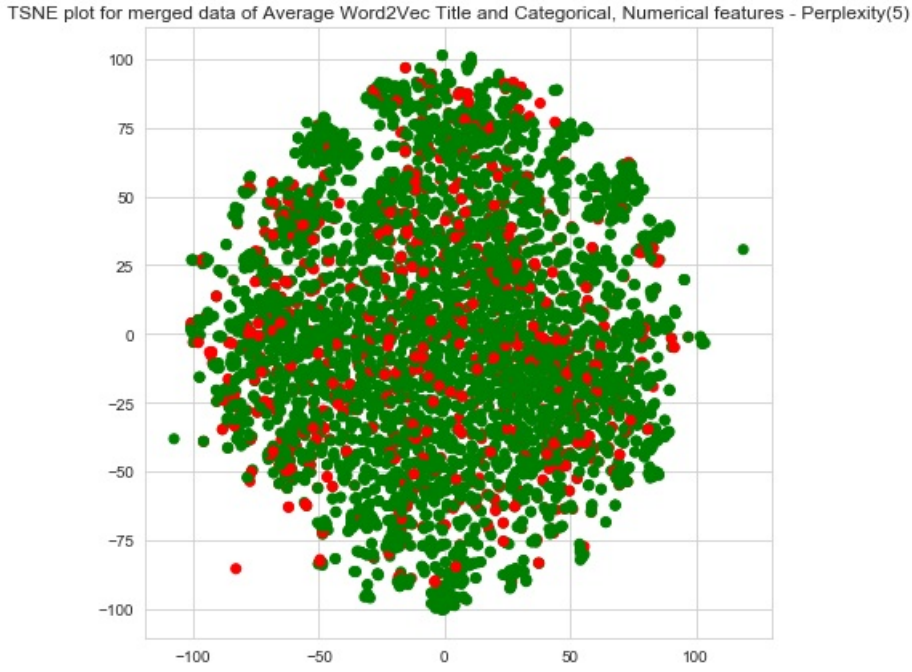
```
word2VecTitleAndOthers = hstack((word2VecTitlesVectorsSub, categoriesVectorsSub,
subCategoriesVectorsSub, teacherPrefixVectorsSub, schoolStateVectorsSub, projectGradeVectorsSub, p
riceStandardizedSub, previouslyPostedStandardizedSub));
word2VecTitleAndOthers.shape
```

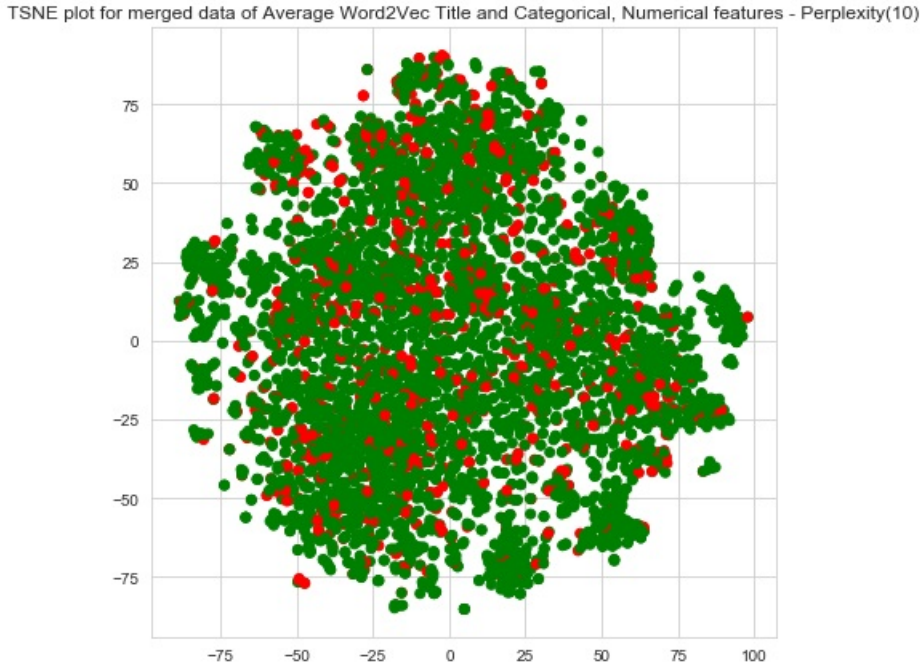
Out[247]:

```
(6000, 401)
```

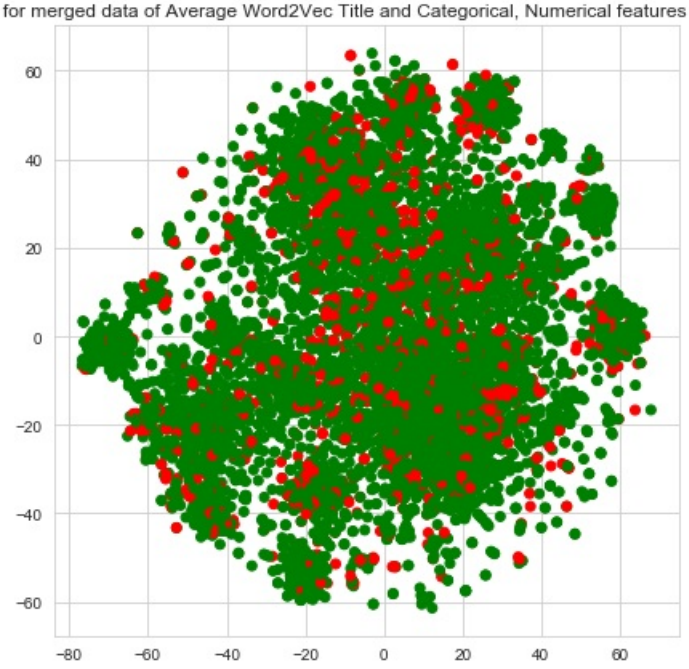
In [248]:

```
perplexityValues = [5, 10, 30, 50, 80, 100]
for perplexityValue in perplexityValues:
    tsne = TSNE(n_components = 2, perplexity = perplexityValue, learning_rate = 200);
    word2VecTitleAndOthersEmbedded = tsne.fit_transform(word2VecTitleAndOthers.toarray());
    word2VecTitleAndOthersTsneData = np.hstack((word2VecTitleAndOthersEmbedded,
classesDataSub.reshape(-1, 1)));
    word2VecTitleAndOthersTsneDataFrame = pd.DataFrame(word2VecTitleAndOthersTsneData, columns = ['
Dimension1', 'Dimension2', 'Class']);
    colors = {0.0:'red', 1.0:'green'}
    plt.title("TSNE plot for merged data of Average Word2Vec Title and Categorical, Numerical feat
ures - Perplexity({})".format(perplexityValue));
    plt.scatter(word2VecTitleAndOthersTsneDataFrame['Dimension1'],
word2VecTitleAndOthersTsneDataFrame['Dimension2'], c = word2VecTitleAndOthersTsneDataFrame['Class'
].apply(lambda x: colors[x]));
    plt.show();
```

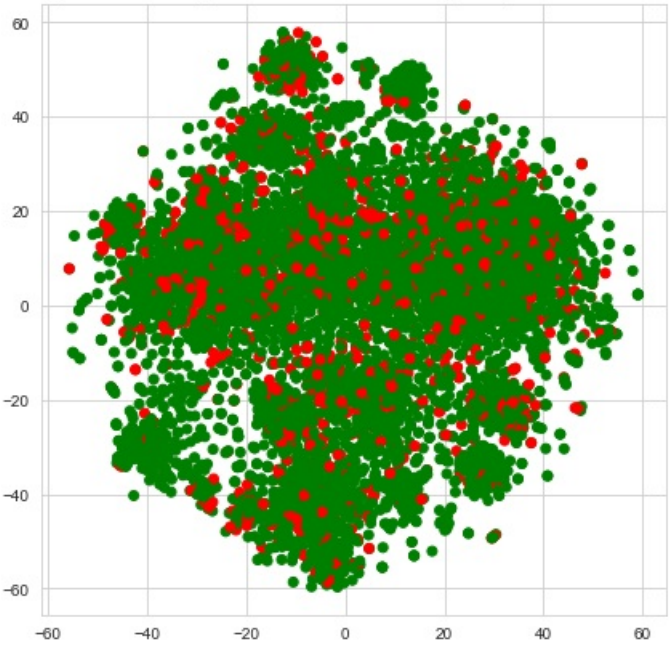




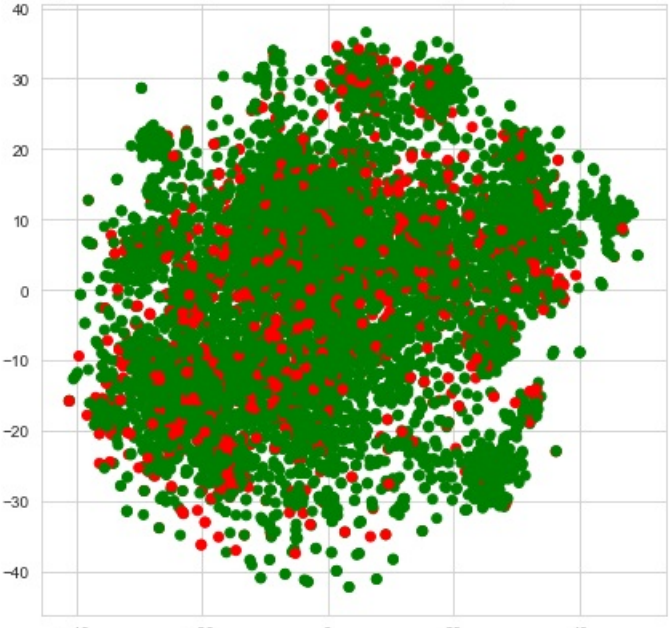
TSNE plot for merged data of Average Word2Vec Title and Categorical, Numerical features - Perplexity(30)

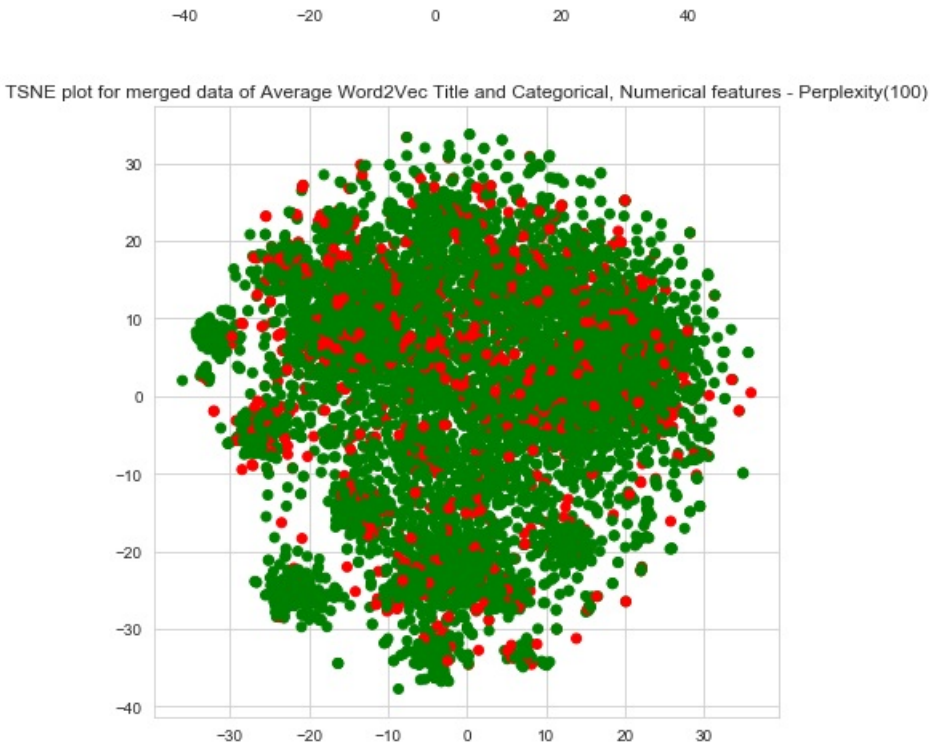


TSNE plot for merged data of Average Word2Vec Title and Categorical, Numerical features - Perplexity(50)



TSNE plot for merged data of Average Word2Vec Title and Categorical, Numerical features - Perplexity(80)





Classification using data merged with Tf-idf Weighted Word2Vec vectorized title and all considered categorical, numerical features

In [346]:

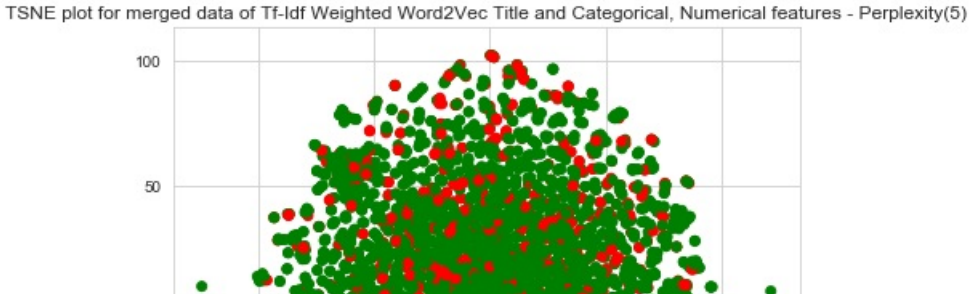
```
tfIdfWeightedWord2VecTitleAndOthers = hstack((tfIdfWeightedWord2VecTitlesVectorsSub,
categoriesVectorsSub, subCategoriesVectorsSub, teacherPrefixVectorsSub, schoolStateVectorsSub,
projectGradeVectorsSub, priceStandardizedSub, previouslyPostedStandardizedSub));
tfIdfWeightedWord2VecTitleAndOthers.shape
```

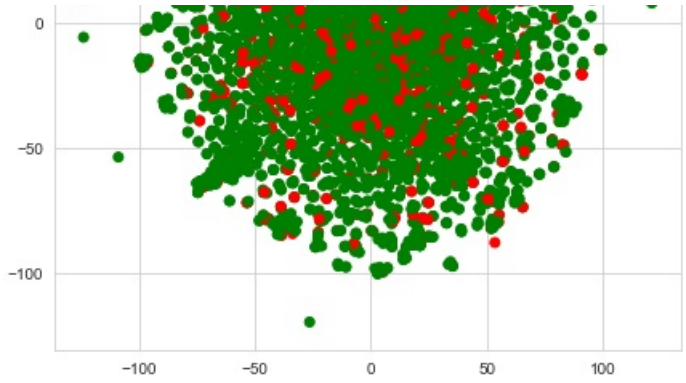
Out[346]:

```
(6000, 401)
```

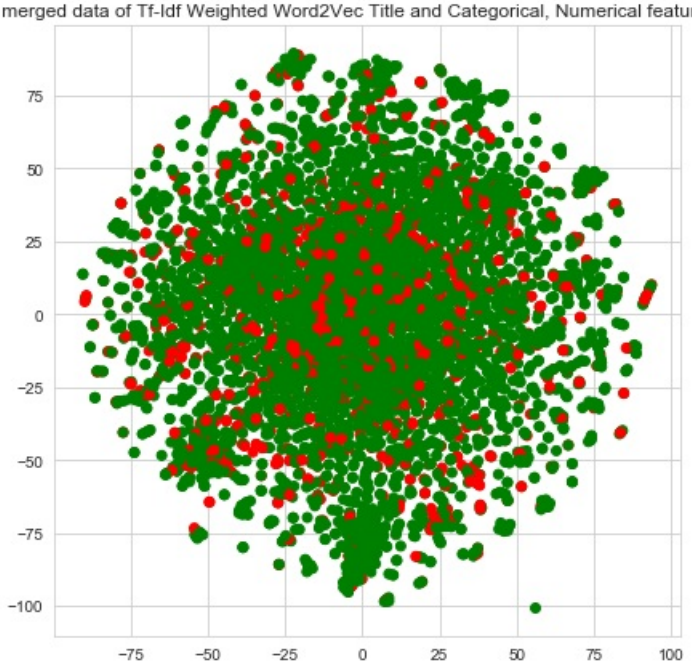
In [347]:

```
perplexityValues = [5, 10, 30, 50, 80, 100]
for perplexityValue in perplexityValues:
    tsne = TSNE(n_components = 2, perplexity = perplexityValue, learning_rate = 200);
    tfIdfWeightedWord2VecTitleAndOthersEmbedded =
tsne.fit_transform(tfIdfWeightedWord2VecTitleAndOthers.toarray());
    tfIdfWeightedWord2VecTitleAndOthersTsneData =
np.hstack((tfIdfWeightedWord2VecTitleAndOthersEmbedded, classesDataSub.reshape(-1, 1)));
    tfIdfWeightedWord2VecTitleAndOthersTsneDataFrame =
pd.DataFrame(tfIdfWeightedWord2VecTitleAndOthersTsneData, columns = ['Dimension1', 'Dimension2', '
Class']);
    colors = {0.0:'red', 1.0:'green'}
    plt.title("TSNE plot for merged data of Tf-Idf Weighted Word2Vec Title and Categorical, Numeri
cal features - Perplexity({})".format(perplexityValue));
    plt.scatter(tfIdfWeightedWord2VecTitleAndOthersTsneDataFrame['Dimension1'],
tfIdfWeightedWord2VecTitleAndOthersTsneDataFrame['Dimension2'], c =
tfIdfWeightedWord2VecTitleAndOthersTsneDataFrame['Class'].apply(lambda x: colors[x]));
    plt.show();
```

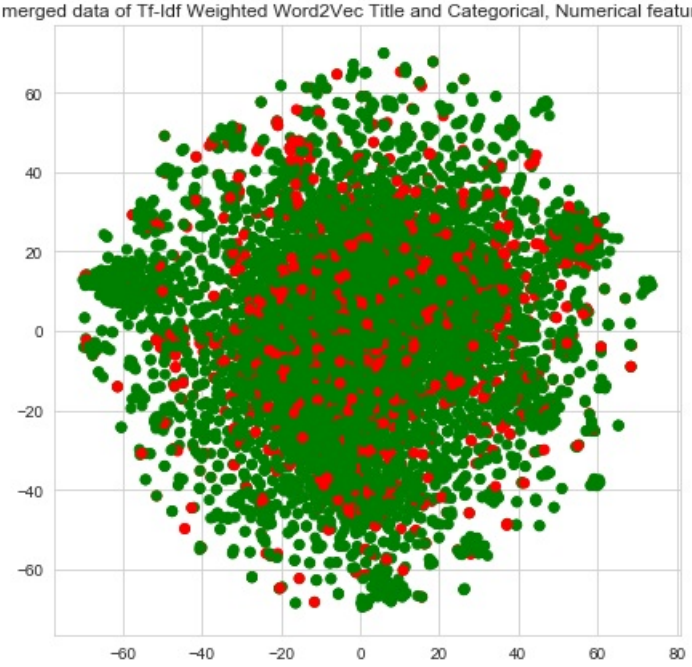




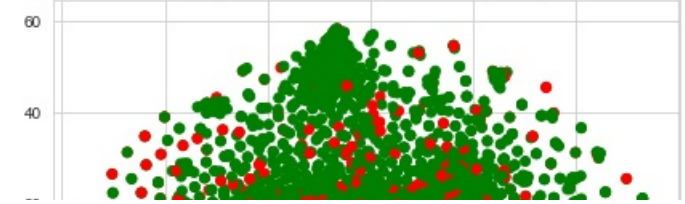
TSNE plot for merged data of Tf-Idf Weighted Word2Vec Title and Categorical, Numerical features - Perplexity(10)

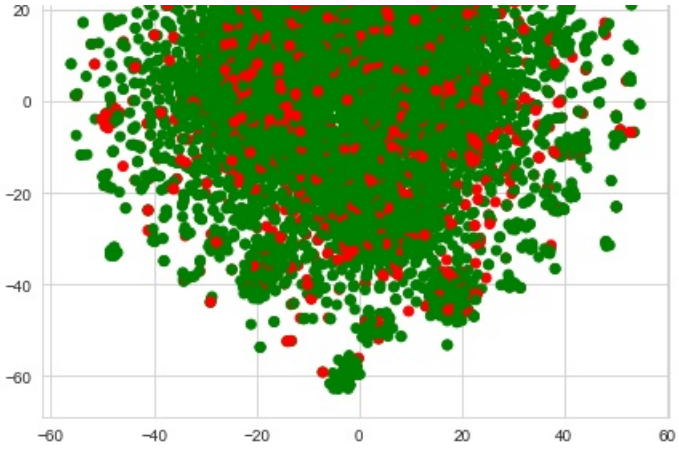


TSNE plot for merged data of Tf-Idf Weighted Word2Vec Title and Categorical, Numerical features - Perplexity(30)

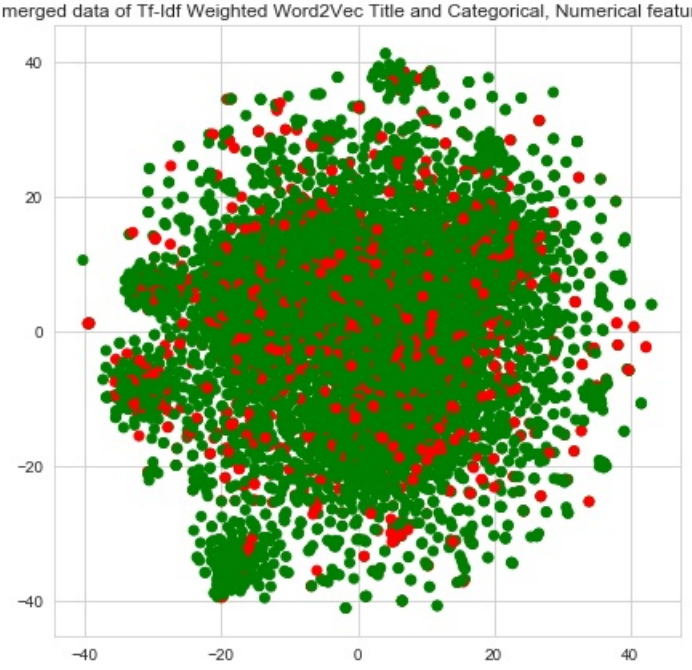


TSNE plot for merged data of Tf-Idf Weighted Word2Vec Title and Categorical, Numerical features - Perplexity(50)

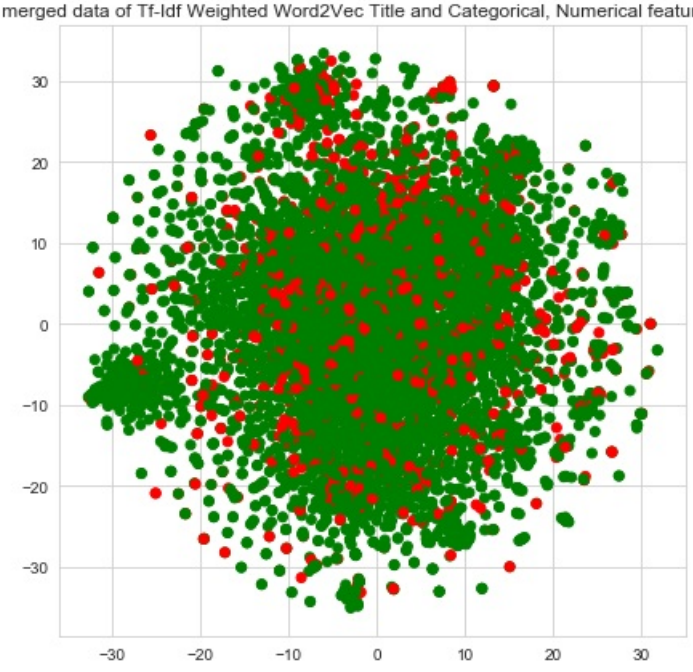




TSNE plot for merged data of Tf-Idf Weighted Word2Vec Title and Categorical, Numerical features - Perplexity(80)



TSNE plot for merged data of Tf-Idf Weighted Word2Vec Title and Categorical, Numerical features - Perplexity(100)



Classification using data merged with all vectorizations of project_title and with all considered categorical, numerical features

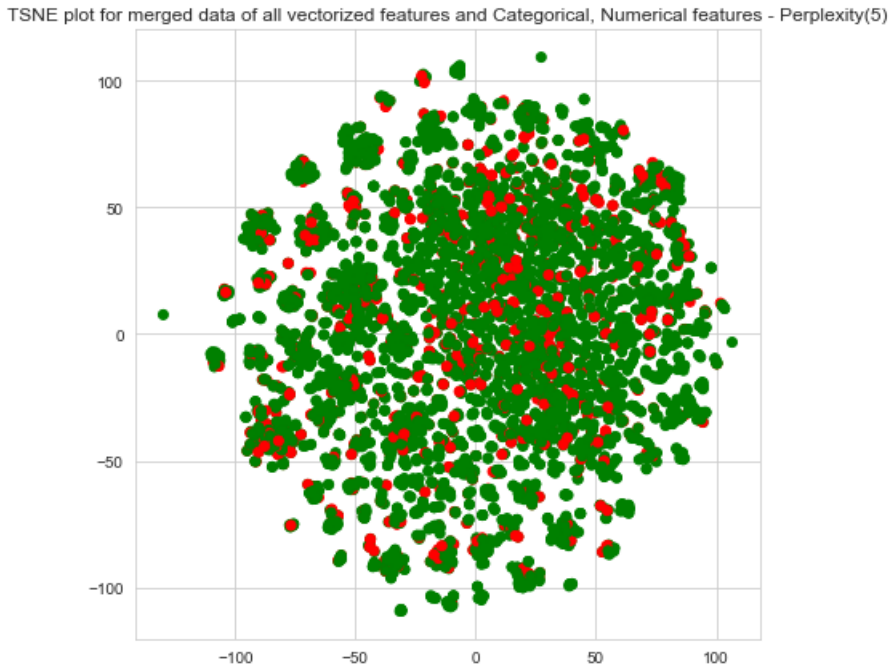
In [252]:

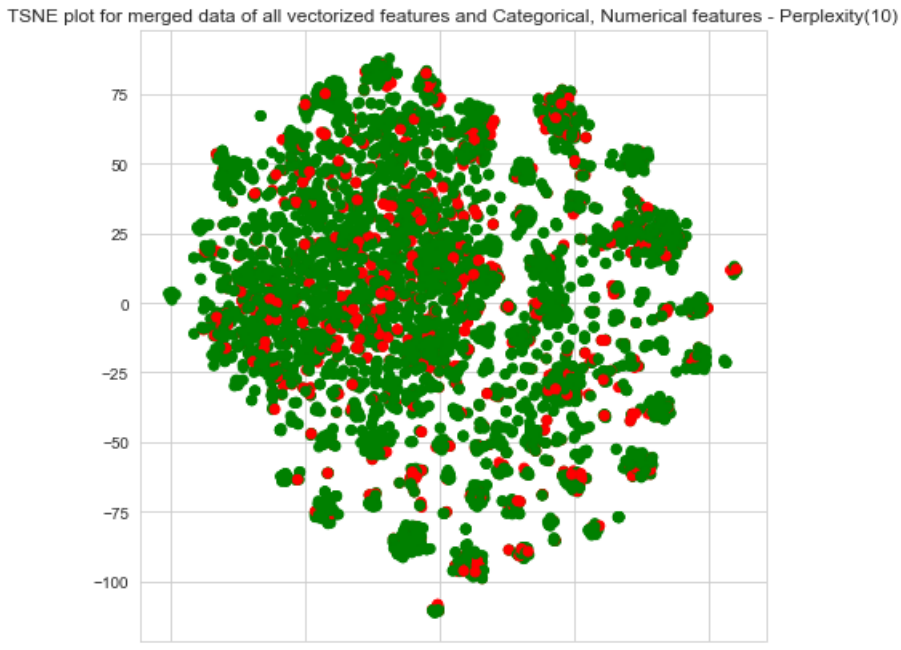
```
allFeatures = hstack((bowTitleModelSub, tfIdfTitleModelSub, word2VecTitlesVectorsSub, tfIdfWeighted
Word2VecTitlesVectorsSub, categoriesVectorsSub, subCategoriesVectorsSub, teacherPrefixVectorsSub,
schoolStateVectorsSub, projectGradeVectorsSub, priceStandardizedSub,
previouslyPostedStandardizedSub))
print(allFeatures.shape)
```

```
(6000, 4249)
```

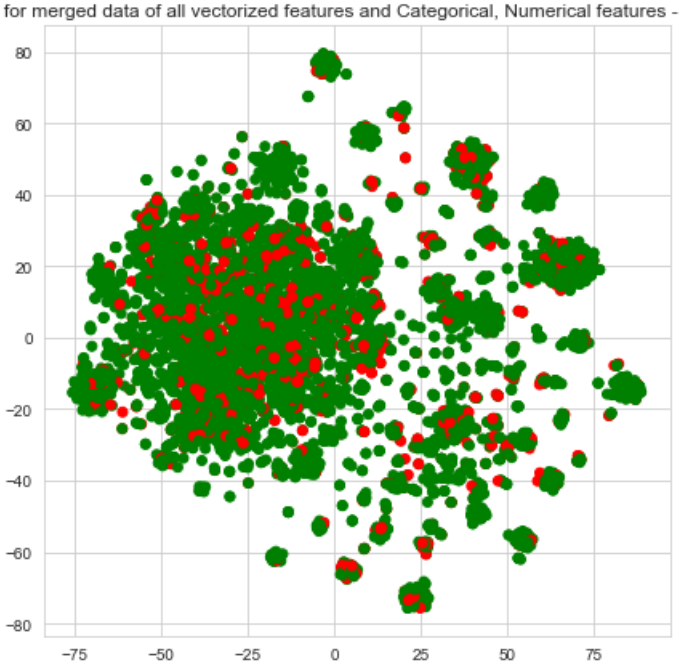
In [253]:

```
perplexityValues = [5, 10, 30, 50, 80, 100]
for perplexityValue in perplexityValues:
    tsne = TSNE(n_components = 2, perplexity = perplexityValue, learning_rate = 200);
    allFeaturesEmbedded = tsne.fit_transform(allFeatures.toarray());
    allFeaturesTsneData = np.hstack((allFeaturesEmbedded, classesDataSub.reshape(-1, 1)));
    allFeaturesTsneDataFrame = pd.DataFrame(allFeaturesTsneData, columns = ['Dimension1', 'Dimensio
n2', 'Class']);
    colors = {0.0:'red', 1.0:'green'}
    plt.title("TSNE plot for merged data of all vectorized features and Categorical, Numerical fea
tures - Perplexity({})".format(perplexityValue));
    plt.scatter(allFeaturesTsneDataFrame['Dimension1'], allFeaturesTsneDataFrame['Dimension2'], c =
allFeaturesTsneDataFrame['Class'].apply(lambda x: colors[x]));
    plt.show();
```

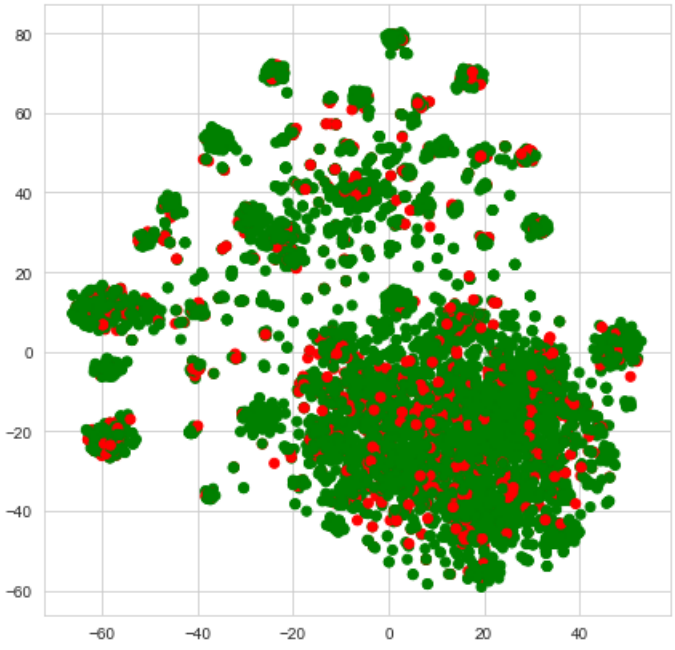




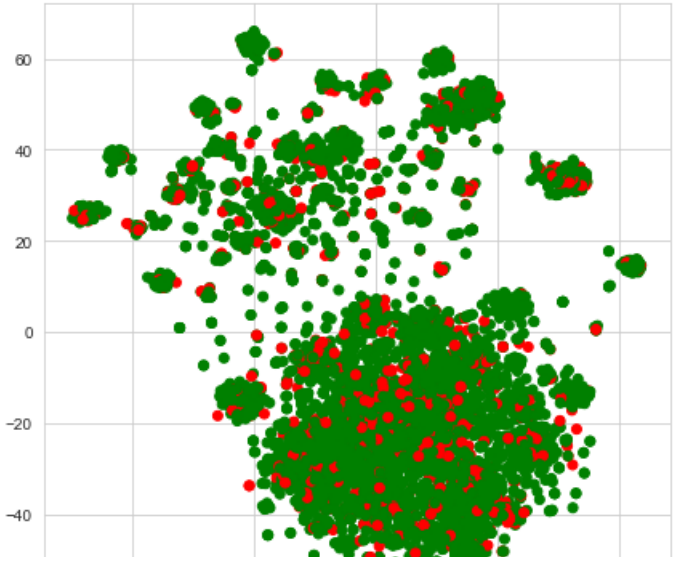
TSNE plot for merged data of all vectorized features and Categorical, Numerical features - Perplexity(30)

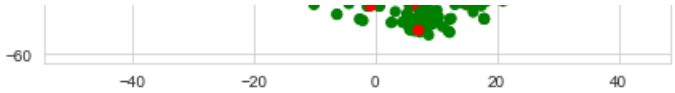


TSNE plot for merged data of all vectorized features and Categorical, Numerical features - Perplexity(50)

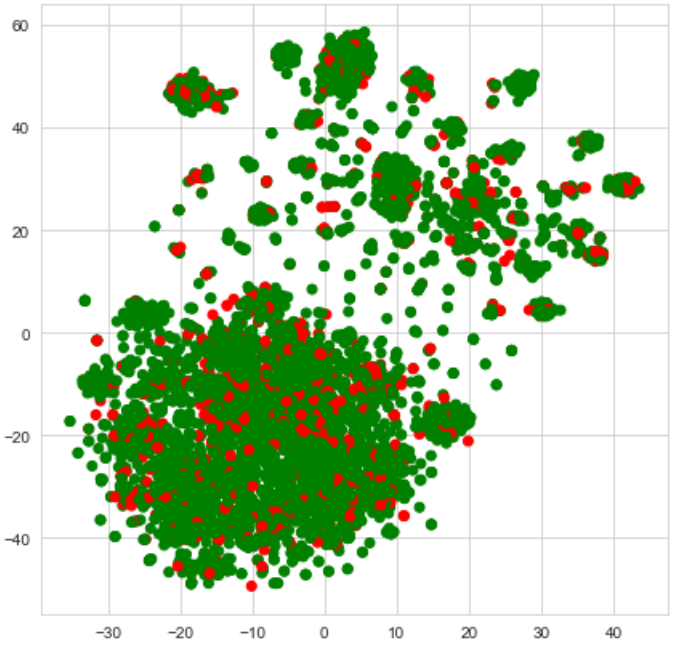


TSNE plot for merged data of all vectorized features and Categorical, Numerical features - Perplexity(80)





TSNE plot for merged data of all vectorized features and Categorical, Numerical features - Perplexity(100)



Conclusion about data visualization using t-sne:

1. Bag of Words, Tf-Idf are better than word2vec vectorizations because of forming some small group of clusters with less overlap of overall data when compared to others.
2. Higher perplexity values seems better in data visualization because of less overlap of data than others.
3. None of the techniques are useful for classification because of huge overlap of data.
4. It is not seperable problem in 2-dimensions but it may be seperable in higher dimensions.

Classification & Modelling using K-NN

Classification using K-NN(Simple Cross Validation)

Splitting Data

In [121]:

```
projectsData = projectsData.dropna(subset = ['teacher_prefix']);
projectsData.shape
```

Out[121]:

```
(109245, 22)
```

In [122]:

```
classesData = projectsData['project_is_approved']
print(classesData.shape)
```

```
(109245,)
```

In [123]:

```
trainingAndCrossValidateData, testData, classesTrainingAndCrossValidate, classesTest =
cross_validation.train_test_split(projectsData[0:10000], classesData[0:10000], test_size = 0.3, ra
```

```
ndom_state = 0);
trainingData, crossValidateData, classesTraining, classesCrossValidate =
cross_validation.train_test_split(trainingAndCrossValidateData, classesTrainingAndCrossValidate, t
est_size =  0.3, random_state = 0);
```

In [124]:

```
print("Shapes of splitted data: ");
equalsBorder(70);
print("trainingAndCrossValidateData shape: ", trainingAndCrossValidateData.shape);
print("classesTrainingAndCrossValidate shape: ", classesTrainingAndCrossValidate.shape);
print("testData shape: ", testData.shape);
print("classesTest: ", classesTest.shape);
print("trainingData shape: ", trainingData.shape);
print("classesTraining shape: ", classesTraining.shape);
print("crossValidateData shape: ", crossValidateData.shape);
print("classesCrossValidate shape: ", classesCrossValidate.shape);
```

```
Shapes of splitted data:
======================================================================
trainingAndCrossValidateData shape:  (7000, 22)
classesTrainingAndCrossValidate shape:  (7000,)
testData shape:  (3000, 22)
classesTest:  (3000,)
trainingData shape:  (4900, 22)
classesTraining shape:  (4900,)
crossValidateData shape:  (2100, 22)
classesCrossValidate shape:  (2100,)
```

In [125]:

```
print("Number of negative points: ", trainingData[trainingData['project_is_approved'] == 0].shape)
;
print("Number of positive points: ", trainingData[trainingData['project_is_approved'] == 1].shape)
;
```

```
Number of negative points:  (713, 22)
Number of positive points:  (4187, 22)
```

Balancing Data

Note: Instead of displaying whole vectorization process for balanced and imbalanced data, we have simply disabled below cell while performing analysis on imbalanced data and enabled while performing analysis on balanced data

In [ ]:

```
negativeData = trainingData[trainingData['project_is_approved'] == 0];
positiveData = trainingData[trainingData['project_is_approved'] == 1];
negativeDataBalanced = resample(negativeData, replace = True, n_samples = 8319, random_state = 44);
trainingData = pd.concat([positiveData, negativeDataBalanced]);
trainingData = shuffle(trainingData);
classesTraining = trainingData['project_is_approved'];
```

Vectorizing categorical data

1. Vectorizing cleaned_categories(project_subject_categories cleaned) - One Hot Encoding

In [127]:

```
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e cleaned_categories
subjectsCategoriesVectorizer = CountVectorizer(vocabulary = list(sortedCategoriesDictionary.keys()
), lowercase = False, binary = True);
# Fitting CountVectorizer with cleaned_categories values
subjectsCategoriesVectorizer.fit(trainingData['cleaned_categories'].values);
```

```
# Vectorizing categories using one-hot-encoding
categoriesVectors = subjectsCategoriesVectorizer.transform(trainingData['cleaned_categories'].valu
es);
```

In [128]:

```
print("Features used in vectorizing categories: ");
equalsBorder(70);
print(subjectsCategoriesVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of cleaned_categories matrix after vectorization(one-hot-encoding): ",
categoriesVectors.shape);
equalsBorder(70);
print("Sample vectors of categories: ");
equalsBorder(70);
print(categoriesVectors[0:4])
```

```
Features used in vectorizing categories:
======================================================================
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
======================================================================
Shape of cleaned_categories matrix after vectorization(one-hot-encoding):  (12506, 9)
======================================================================
Sample vectors of categories:
======================================================================
  (0, 8) 1
  (1, 7) 1
  (2, 7) 1
  (2, 8) 1
  (3, 8) 1
```

2. Vectorizing cleaned_sub_categories(project_subject_sub_categories cleaned) - One Hot Encoding

In [129]:

```
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e cleaned_sub_categories
subjectsSubCategoriesVectorizer = CountVectorizer(vocabulary = list(sortedDictionarySubCategories.
keys()), lowercase = False, binary = True);
# Fitting CountVectorizer with cleaned_sub_categories values
subjectsSubCategoriesVectorizer.fit(trainingData['cleaned_sub_categories'].values);
# Vectorizing sub categories using one-hot-encoding
subCategoriesVectors =
subjectsSubCategoriesVectorizer.transform(trainingData['cleaned_sub_categories'].values);
```

In [130]:

```
print("Features used in vectorizing subject sub categories: ");
equalsBorder(70);
print(subjectsSubCategoriesVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of cleaned_categories matrix after vectorization(one-hot-encoding): ",
subCategoriesVectors.shape);
equalsBorder(70);
print("Sample vectors of categories: ");
equalsBorder(70);
print(subCategoriesVectors[0:4])
```

```
Features used in vectorizing subject sub 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',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
======================================================================
Shape of cleaned_categories matrix after vectorization(one-hot-encoding):  (12506, 30)
======================================================================
Sample vectors of categories:
```

```
Sample vectors of categories:
======================================================================
  (0, 27) 1
  (0, 29) 1
  (1, 28) 1
  (2, 27) 1
  (2, 28) 1
  (3, 29) 1
```

3. Vectorizing teacher_prefix - One Hot Encoding

In [131]:

```
def giveCounter(data):
    counter = Counter();
    for dataValue in data:
        counter.update(str(dataValue).split());
    return counter
```

In [132]:

```
giveCounter(trainingData['teacher_prefix'].values)
```

Out[132]:

```
Counter({'Mrs.': 6354, 'Teacher': 298, 'Ms.': 4624, 'Mr.': 1230})
```

In [133]:

```
teacherPrefixDictionary = dict(giveCounter(trainingData['teacher_prefix'].values));
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e teacher_prefix
teacherPrefixVectorizer = CountVectorizer(vocabulary = list(teacherPrefixDictionary.keys()),
lowercase = False, binary = True);
# Fitting CountVectorizer with teacher_prefix values
teacherPrefixVectorizer.fit(trainingData['teacher_prefix'].values);
# Vectorizing teacher_prefix using one-hot-encoding
teacherPrefixVectors = teacherPrefixVectorizer.transform(trainingData['teacher_prefix'].values);
```

In [134]:

```
print("Features used in vectorizing teacher_prefix: ");
equalsBorder(70);
print(teacherPrefixVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of teacher_prefix matrix after vectorization(one-hot-encoding): ",
teacherPrefixVectors.shape);
equalsBorder(70);
print("Sample vectors of teacher_prefix: ");
equalsBorder(70);
print(teacherPrefixVectors[0:100]);
```

```
Features used in vectorizing teacher_prefix:
======================================================================
['Mrs.', 'Teacher', 'Ms.', 'Mr.']
======================================================================
Shape of teacher_prefix matrix after vectorization(one-hot-encoding):  (12506, 4)
======================================================================
Sample vectors of teacher_prefix:
======================================================================
  (1, 1) 1
  (4, 1) 1
  (7, 1) 1
  (43, 1) 1
```

In [135]:

```
teacherPrefixes = [prefix.replace('.', '') for prefix in trainingData['teacher_prefix'].values];
teacherPrefixes[0:5]
```

Out[135]:

```
['Mrs', 'Teacher', 'Mrs', 'Ms', 'Teacher']
```

In [136]:

```
trainingData['teacher_prefix'] = teacherPrefixes;
trainingData.head(3)
```

Out[136]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	p
4420	113319	p143796	f519f379c4aa3dde01bcce86203ff61d	Mrs	SC	2017-03-25 10:01:44	
953	96493	p064010	9f4d0a39b55f33c15f028736a8f086f1	Teacher	NE	2017-01-02 18:59:12	
2307	126229	p178744	47aed895c7f60471962bc271d079e89f	Mrs	PA	2017-02-11 21:03:00	

3 rows × 22 columns

In [137]:

```
teacherPrefixDictionary = dict(giveCounter(trainingData['teacher_prefix'].values));
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e teacher_prefix
teacherPrefixVectorizer = CountVectorizer(vocabulary = list(teacherPrefixDictionary.keys()),
lowercase = False, binary = True);
# Fitting CountVectorizer with teacher_prefix values
teacherPrefixVectorizer.fit(trainingData['teacher_prefix'].values);
# Vectorizing teacher_prefix using one-hot-encoding
teacherPrefixVectors = teacherPrefixVectorizer.transform(trainingData['teacher_prefix'].values);
```

In [138]:

```
print("Features used in vectorizing teacher_prefix: ");
equalsBorder(70);
print(teacherPrefixVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of teacher_prefix matrix after vectorization(one-hot-encoding): ",
teacherPrefixVectors.shape);
equalsBorder(70);
print("Sample vectors of teacher_prefix: ");
equalsBorder(70);
print(teacherPrefixVectors[0:4]);
```

```
Features used in vectorizing teacher_prefix:
======================================================================
['Mrs', 'Teacher', 'Ms', 'Mr']
======================================================================
Shape of teacher_prefix matrix after vectorization(one-hot-encoding):  (12506, 4)
======================================================================
Sample vectors of teacher_prefix:
======================================================================
  (0, 0) 1
  (1, 1) 1
  (2, 0) 1
  (3, 2) 1
```

4. Vectorizing school_state - One Hot Encoding

In [139]:

```
schoolStateDictionary = dict(giveCounter(trainingData['school_state'].values));
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e school states
schoolStateVectorizer = CountVectorizer(vocabulary = list(schoolStateDictionary.keys()), lowercase
= False, binary = True);
# Fitting CountVectorizer with school_state values
schoolStateVectorizer.fit(trainingData['school_state'].values);
# Vectorizing school_state using one-hot-encoding
schoolStateVectors = schoolStateVectorizer.transform(trainingData['school_state'].values);
```

In [140]:

```
print("Features used in vectorizing school_state: ");
equalsBorder(70);
print(schoolStateVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of school_state matrix after vectorization(one-hot-encoding): ", schoolStateVectors.s
hape);
equalsBorder(70);
print("Sample vectors of school_state: ");
equalsBorder(70);
print(schoolStateVectors[0:4]);
```

```
Features used in vectorizing school_state:
======================================================================
['SC', 'NE', 'PA', 'FL', 'TX', 'CA', 'NY', 'IL', 'UT', 'MI', 'GA', 'MO', 'AZ', 'LA', 'TN', 'CO', 'N
V', 'NC', 'OR', 'WI', 'VA', 'OH', 'CT', 'MA', 'NJ', 'DE', 'AL', 'IN', 'MS', 'DC', 'KY', 'OK', 'NM',
'MT', 'IA', 'HI', 'NH', 'MN', 'SD', 'KS', 'ME', 'MD', 'RI', 'AR', 'WA', 'AK', 'ND', 'WV', 'ID', 'WY
', 'VT']
======================================================================
Shape of school_state matrix after vectorization(one-hot-encoding):  (12506, 51)
======================================================================
Sample vectors of school_state:
======================================================================
  (0, 0) 1
  (1, 1) 1
  (2, 2) 1
  (3, 3) 1
```

5. Vectorizing project_grade_category - One Hot Encoding

In [141]:

```
giveCounter(trainingData['project_grade_category'])
```

Out[141]:

```
Counter({'Grades': 12506,
         '9-12': 1241,
         'PreK-2': 5165,
         '3-5': 4083,
         '6-8': 2017})
```

In [142]:

```
cleanedGrades = []
for grade in trainingData['project_grade_category'].values:
    grade = grade.replace(' ', '');
    grade = grade.replace('-', 'to');
    cleanedGrades.append(grade);
cleanedGrades[0:4]
```

Out[142]:

```
['Grades9to12', 'GradesPreKto2', 'Grades3to5', 'GradesPreKto2']
```

In [143]:

```
trainingData['project_grade_category'] = cleanedGrades
trainingData.head(4)
```

Out[143]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime
4420	113319	p143796	f519f379c4aa3dde01bcce86203ff61d	Mrs	SC	2017-03-25 10:01:44
953	96493	p064010	9f4d0a39b55f33c15f028736a8f086f1	Teacher	NE	2017-01-02 18:59:12
2307	126229	p178744	47aed895c7f60471962bc271d079e89f	Mrs	PA	2017-02-11 21:03:00
6076	144766	p151022	814440f0a77c50e525786ce009c34e19	Ms	FL	2016-05-31 22:48:41

4 rows × 22 columns

In [144]:

```
projectGradeDictionary = dict(giveCounter(trainingData['project_grade_category'].values));
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e project grade categories
projectGradeVectorizer = CountVectorizer(vocabulary = list(projectGradeDictionary.keys()),
lowercase = False, binary = True);
# Fitting CountVectorizer with project_grade_category values
projectGradeVectorizer.fit(trainingData['project_grade_category'].values);
# Vectorizing project_grade_category using one-hot-encoding
projectGradeVectors =
projectGradeVectorizer.transform(trainingData['project_grade_category'].values);
```

In [145]:

```
print("Features used in vectorizing project_grade_category: ");
equalsBorder(70);
print(projectGradeVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of school_state matrix after vectorization(one-hot-encoding): ", projectGradeVectors.
shape);
equalsBorder(70);
print("Sample vectors of school_state: ");
equalsBorder(70);
print(projectGradeVectors[0:4]);
```

```
Features used in vectorizing project_grade_category:
======================================================================
['Grades9to12', 'GradesPreKto2', 'Grades3to5', 'Grades6to8']
======================================================================
Shape of school_state matrix after vectorization(one-hot-encoding):  (12506, 4)
======================================================================
Sample vectors of school_state:
======================================================================
  (0, 0) 1
  (1, 1) 1
```

```
  (2, 2) 1
  (3, 1) 1
```

In [146]:

```
preProcessedEssaysWithStopWords, preProcessedEssaysWithoutStopWords =
preProcessingWithAndWithoutStopWords(trainingData['project_essay']);
preProcessedProjectTitlesWithStopWords, preProcessedProjectTitlesWithoutStopWords =
preProcessingWithAndWithoutStopWords(trainingData['project_title']);
```

Vectorizing Text Data

Bag of Words

1. Vectorizing project_essay

In [147]:

```
# Initializing countvectorizer for bag of words vectorization of preprocessed project essays
bowEssayVectorizer = CountVectorizer(min_df = 10);
# Transforming the preprocessed essays to bag of words vectors
bowEssayModel = bowEssayVectorizer.fit_transform(preProcessedEssaysWithoutStopWords);
```

In [148]:

```
print("Some of the Features used in vectorizing preprocessed essays: ");
equalsBorder(70);
print(bowEssayVectorizer.get_feature_names()[-40:]);
equalsBorder(70);
print("Shape of preprocessed essay matrix after vectorization: ", bowEssayModel.shape);
equalsBorder(70);
print("Sample bag-of-words vector of preprocessed essay: ");
equalsBorder(70);
print(bowEssayModel[0])
```

```
Some of the Features used in vectorizing preprocessed essays:
======================================================================
['wrong', 'wrote', 'www', 'xtramath', 'yale', 'yard', 'year', 'yearly', 'yearn', 'yearning', 'year
s', 'yelled', 'yelling', 'yellow', 'yemen', 'yes', 'yet', 'yields', 'yikes', 'yoga', 'york', 'youn
g', 'younger', 'youngest', 'yousafzai', 'youth', 'youthful', 'youtube', 'zaretta', 'zeal', 'zen',
'zentangle', 'zero', 'zip', 'ziploc', 'zipping', 'zone', 'zoo', 'zoology', 'zoos']
======================================================================
Shape of preprocessed essay matrix after vectorization:  (12506, 7148)
======================================================================
Sample bag-of-words vector of preprocessed essay:
======================================================================
  (0, 4260) 1
  (0, 6366) 1
  (0, 533) 1
  (0, 2640) 1
  (0, 6092) 1
  (0, 1182) 1
  (0, 6514) 1
  (0, 807) 1
  (0, 2934) 1
  (0, 347) 1
  (0, 518) 1
  (0, 5404) 1
  (0, 5860) 1
  (0, 3824) 1
  (0, 6297) 1
  (0, 3277) 1
  (0, 4259) 1
  (0, 6887) 1
  (0, 4319) 1
  (0, 5031) 1
  (0, 1504) 1
```

```
  (0, 579) 1
  (0, 3151) 1
  (0, 4369) 1
  (0, 3684) 1
  : :
  (0, 1776) 1
  (0, 2314) 1
  (0, 2995) 1
  (0, 7068) 1
  (0, 3147) 1
  (0, 3296) 1
  (0, 3889) 1
  (0, 1291) 1
  (0, 4690) 1
  (0, 3690) 1
  (0, 5003) 1
  (0, 2206) 2
  (0, 3807) 1
  (0, 1894) 1
  (0, 704) 1
  (0, 1861) 1
  (0, 3731) 2
  (0, 4294) 1
  (0, 5989) 1
  (0, 3960) 2
  (0, 2882) 1
  (0, 12) 1
  (0, 137) 1
  (0, 5150) 1
  (0, 6189) 10
```

2. Vectorizing project_title

In [149]:

```
# Initializing countvectorizer for bag of words vectorization of preprocessed project titles
bowTitleVectorizer = CountVectorizer(min_df = 10);
# Transforming the preprocessed project titles to bag of words vectors
bowTitleModel = bowTitleVectorizer.fit_transform(preProcessedProjectTitlesWithoutStopWords);
```

In [150]:

```
print("Some of the Features used in vectorizing preprocessed titles: ");
equalsBorder(70);
print(bowTitleVectorizer.get_feature_names()[-40:]);
equalsBorder(70);
print("Shape of preprocessed title matrix after vectorization: ", bowTitleModel.shape);
equalsBorder(70);
print("Sample bag-of-words vector of preprocessed title: ");
equalsBorder(70);
print(bowTitleModel[0])
```

```
Some of the Features used in vectorizing preprocessed titles:
======================================================================
['week', 'weekly', 'weighty', 'welcoming', 'whole', 'wi', 'width', 'wiggle', 'wiggles', 'wiggly',
'win', 'winner', 'winter', 'wish', 'without', 'wizards', 'wo', 'wobble', 'wobbling', 'wonderful',
'woodwinds', 'word', 'words', 'work', 'working', 'workshop', 'world', 'worms', 'would', 'write', '
writer', 'writers', 'writing', 'year', 'yes', 'yoga', 'yogi', 'young', 'zen', 'zone']
======================================================================
Shape of preprocessed title matrix after vectorization:  (12506, 987)
======================================================================
Sample bag-of-words vector of preprocessed title:
======================================================================
  (0, 725) 1
  (0, 928) 1
  (0, 423) 1
  (0, 863) 1
  (0, 61) 1
  (0, 527) 1
```

Tf-Idf Vectorization

1. Vectorizing project_essay

In [151]:

```
# Intializing tfidf vectorizer for tf-idf vectorization of preprocessed project essays
tfIdfEssayVectorizer = TfidfVectorizer(min_df = 10);
# Transforming the preprocessed project essays to tf-idf vectors
tfIdfEssayModel = tfIdfEssayVectorizer.fit_transform(preProcessedEssaysWithoutStopWords);
```

In [152]:

```
print("Some of the Features used in tf-idf vectorizing preprocessed essays: ");
equalsBorder(70);
print(tfIdfEssayVectorizer.get_feature_names()[-40:]);
equalsBorder(70);
print("Shape of preprocessed title matrix after tf-idf vectorization: ", tfIdfEssayModel.shape);
equalsBorder(70);
print("Sample Tf-Idf vector of preprocessed essay: ");
equalsBorder(70);
print(tfIdfEssayModel[0])
```

```
Some of the Features used in tf-idf vectorizing preprocessed essays:
======================================================================
['wrong', 'wrote', 'www', 'xtramath', 'yale', 'yard', 'year', 'yearly', 'yearn', 'yearning', 'year
s', 'yelled', 'yelling', 'yellow', 'yemen', 'yes', 'yet', 'yields', 'yikes', 'yoga', 'york', 'youn
g', 'younger', 'youngest', 'yousafzai', 'youth', 'youthful', 'youtube', 'zaretta', 'zeal', 'zen',
'zentangle', 'zero', 'zip', 'ziploc', 'zipping', 'zone', 'zoo', 'zoology', 'zoos']
======================================================================
Shape of preprocessed title matrix after tf-idf vectorization:  (12506, 7148)
======================================================================
Sample Tf-Idf vector of preprocessed essay:
======================================================================
  (0, 6189) 0.20651330549908437
  (0, 5150) 0.07231195951563689
  (0, 137) 0.1293328601650488
  (0, 12) 0.1207988522691422
  (0, 2882) 0.044418826049789424
  (0, 3960) 0.06674139889078706
  (0, 5989) 0.05463438854840181
  (0, 4294) 0.04865939188625093
  (0, 3731) 0.05637831290932271
  (0, 1861) 0.0750502257384875
  (0, 704) 0.08341183604072445
  (0, 1894) 0.11219533848131488
  (0, 3807) 0.06941029374022084
  (0, 2206) 0.11706627121517518
  (0, 5003) 0.11080923426745438
  (0, 3690) 0.07212822626249868
  (0, 4690) 0.10926083597294575
  (0, 1291) 0.035879108973742205
  (0, 3889) 0.05219598661220253
  (0, 3296) 0.05716586442716464
  (0, 3147) 0.09461635097927058
  (0, 7068) 0.03534146006765951
  (0, 2995) 0.05111762898626292
  (0, 2314) 0.07767164528994024
  (0, 1776) 0.06683734964774671
  : :
  (0, 3684) 0.05449629687944413
  (0, 4369) 0.03015222495273649
  (0, 3151) 0.06606691586109437
  (0, 579) 0.11814255764916896
  (0, 1504) 0.12436385048019485
  (0, 5031) 0.1598481649019681
  (0, 4319) 0.043457687322987826
  (0, 6887) 0.08648503456255673
  (0, 4259) 0.11720602342452928
  (0, 3277) 0.09805311813937595
  (0, 6297) 0.1456341177428218
  (0, 3824) 0.08306248286669435
  (0, 5860) 0.04193870689773057
  (0, 5404) 0.057924343433348596
  (0, 518) 0.09275612700302466
```

```
  (0, 347) 0.046557783374576674
  (0, 2934) 0.0491770791619131
  (0, 807) 0.06499164113220852
  (0, 6514) 0.04252998691335947
  (0, 1182) 0.04086239459033046
  (0, 6092) 0.07399954981954261
  (0, 2640) 0.07493337906708962
  (0, 533) 0.11994428964737991
  (0, 6366) 0.08120901891534808
  (0, 4260) 0.021401325156307216
```

2. Vectorizing project_title

In [153]:

```
# Intializing tfidf vectorizer for tf-idf vectorization of preprocessed project titles
tfIdfTitleVectorizer = TfidfVectorizer(min_df = 10);
# Transforming the preprocessed project titles to tf-idf vectors
tfIdfTitleModel = tfIdfTitleVectorizer.fit_transform(preProcessedProjectTitlesWithoutStopWords);
```

In [154]:

```
print("Some of the Features used in tf-idf vectorizing preprocessed titles: ");
equalsBorder(70);
print(tfIdfTitleVectorizer.get_feature_names()[-40:]);
equalsBorder(70);
print("Shape of preprocessed title matrix after tf-idf vectorization: ", tfIdfTitleModel.shape);
equalsBorder(70);
print("Sample Tf-Idf vector of preprocessed title: ");
equalsBorder(70);
print(tfIdfTitleModel[0])
```

```
Some of the Features used in tf-idf vectorizing preprocessed titles:
======================================================================
['week', 'weekly', 'weighty', 'welcoming', 'whole', 'wi', 'width', 'wiggle', 'wiggles', 'wiggly',
'win', 'winner', 'winter', 'wish', 'without', 'wizards', 'wo', 'wobble', 'wobbling', 'wonderful',
'woodwinds', 'word', 'words', 'work', 'working', 'workshop', 'world', 'worms', 'would', 'write', '
writer', 'writers', 'writing', 'year', 'yes', 'yoga', 'yogi', 'young', 'zen', 'zone']
======================================================================
Shape of preprocessed title matrix after tf-idf vectorization:  (12506, 987)
======================================================================
Sample Tf-Idf vector of preprocessed title:
======================================================================
  (0, 527) 0.4906866543285391
  (0, 61) 0.5603866308062844
  (0, 863) 0.32395969284155895
  (0, 423) 0.3206129261043845
  (0, 928) 0.3403650899243185
  (0, 725) 0.3487155084980362
```

Average Word2Vector Vectorization

In [155]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/
# We should have glove_vectors file for creating below model
with open('glove_vectors', 'rb') as f:
    gloveModel = pickle.load(f)
    gloveWords =  set(gloveModel.keys())
```

In [156]:

```
print("Glove vector of sample word: ");
equalsBorder(70);
print(gloveModel['technology']);
equalsBorder(70);
print("Shape of glove vector: ", gloveModel['technology'].shape);
```

```
Glove vector of sample word:
================================================================================
[-0.26078   -0.36898   -0.022831   0.21666    0.16672   -0.20268
 -3.1219     0.33057    0.71512    0.28874    0.074368  -0.033203
  0.23783    0.21052    0.076562   0.13007   -0.31706   -0.45888
 -0.45463   -0.13191    0.49761    0.072704   0.16811    0.18846
 -0.16688   -0.21973    0.08575   -0.19577   -0.2101    -0.32436
 -0.56336    0.077996  -0.22758   -0.66569    0.14824    0.038945
  0.50881   -0.1352     0.49966   -0.4401    -0.022335  -0.22744
  0.22086    0.21865    0.36647    0.30495   -0.16565    0.038759
  0.28108   -0.2167     0.12453    0.65401    0.34584   -0.2557
 -0.046363  -0.31111   -0.020936  -0.17122   -0.77114    0.29289
 -0.14625    0.39541   -0.078938   0.051127   0.15076    0.085126
  0.183     -0.06755    0.26312    0.0087276  0.0066415  0.37033
  0.03496   -0.12627   -0.052626  -0.34897    0.14672    0.14799
 -0.21821   -0.042785   0.2661    -1.1105     0.31789    0.27278
  0.054468  -0.27458    0.42732   -0.44101   -0.19302   -0.32948
  0.61501   -0.22301   -0.36354   -0.34983   -0.16125   -0.17195
 -3.363      0.45146   -0.13753    0.31107    0.2061     0.33063
  0.45879    0.24256    0.042342   0.074837  -0.12869    0.12066
  0.42843   -0.4704    -0.18937    0.32685    0.26079    0.20518
 -0.18432   -0.47658    0.69193    0.18731   -0.12516    0.35447
 -0.1969    -0.58981   -0.88914    0.5176     0.13177   -0.078557
  0.032963  -0.19411    0.15109    0.10547   -0.1113    -0.61533
  0.0948    -0.3393    -0.20071   -0.30197    0.29531    0.28017
  0.16049    0.25294   -0.44266   -0.39412    0.13486    0.25178
 -0.044114   1.1519     0.32234   -0.34323   -0.10713   -0.15616
  0.031206   0.46636   -0.52761   -0.39296   -0.068424  -0.04072
  0.41508   -0.34564    0.71001   -0.364      0.2996     0.032281
  0.34035    0.23452    0.78342    0.48045   -0.1609     0.40102
 -0.071795  -0.16531    0.082153   0.52065    0.24194    0.17113
  0.33552   -0.15725   -0.38984    0.59337   -0.19388   -0.39864
 -0.47901    1.0835     0.24473    0.41309    0.64952    0.46846
  0.024386  -0.72087   -0.095061   0.10095   -0.025229   0.29435
 -0.57696    0.53166   -0.0058338 -0.3304     0.19661   -0.085206
  0.34225    0.56262    0.19924   -0.027111  -0.44567    0.17266
  0.20887   -0.40702    0.63954    0.50708   -0.31862   -0.39602
 -0.1714    -0.040006  -0.45077   -0.32482   -0.0316     0.54908
 -0.1121     0.12951   -0.33577   -0.52768   -0.44592   -0.45388
  0.66145    0.33023   -1.9089     0.5318     0.21626   -0.13152
  0.48258    0.68028   -0.84115   -0.51165    0.40017    0.17233
 -0.033749   0.045275   0.37398   -0.18252    0.19877    0.1511
  0.029803   0.16657   -0.12987   -0.50489    0.55311   -0.22504
  0.13085   -0.78459    0.36481   -0.27472    0.031805   0.53052
 -0.20078    0.46392   -0.63554    0.040289  -0.19142   -0.0097011
  0.068084  -0.10602    0.25567    0.096125  -0.10046    0.15016
 -0.26733   -0.26494    0.057888   0.062678  -0.11596    0.28115
  0.25375   -0.17954    0.20615    0.24189    0.062696   0.27719
 -0.42601   -0.28619   -0.44697   -0.082253  -0.73415   -0.20675
 -0.60289   -0.06728    0.15666   -0.042614   0.41368   -0.17367
 -0.54012    0.23883    0.23075    0.13608   -0.058634  -0.089705
  0.18469    0.023634   0.16178    0.23384    0.24267    0.091846 ]
================================================================================
Shape of glove vector:  (300,)
```

In [157]:

```
def getWord2VecVectors(texts):
    word2VecTextsVectors = [];
    for preProcessedText in tqdm(texts):
        word2VecTextVector = np.zeros(300);
        numberOfWordsInText = 0;
        for word in preProcessedText.split():
            if word in gloveWords:
                word2VecTextVector += gloveModel[word];
                numberOfWordsInText += 1;
        if numberOfWordsInText != 0:
            word2VecTextVector = word2VecTextVector / numberOfWordsInText;
        word2VecTextsVectors.append(word2VecTextVector);
    return word2VecTextsVectors;
```

1. Vectorizing project_essay

In [158]:

```
word2VecEssaysVectors = getWord2VecVectors(preProcessedEssaysWithoutStopWords);
```

In [159]:

```
print("Shape of Word2Vec vectorization matrix of essays: {},{}".format(len(word2VecEssaysVectors),
len(word2VecEssaysVectors[0])));
equalsBorder(70);
print("Sample essay: ");
equalsBorder(70);
print(preProcessedEssaysWithoutStopWords[0]);
equalsBorder(70);
print("Word2Vec vector of sample essay: ");
equalsBorder(70);
print(word2VecEssaysVectors[0]);
```

```
Shape of Word2Vec vectorization matrix of essays: 12506,300
======================================================================
Sample essay:
======================================================================
students range 9th 12th grade many special needs learning disabilities behavior disorders limited
english proficiency large percentage students come low income households work hard everything desp
ite difficulties dedicated students strive help esol students created clothes closet student counc
il organized school supply drives school neighboring schools pulled together raise money fellow st
udents serious illnesses inspire sense community often teens today refuse read anything longer
text message books reading level uninteresting books might interesting difficult give many high
school students listen higher comprehension level read particularly true students learning
difficulties students whose primary language not english however audio books help students
correctly pronouncing new vocabulary difficult names improving sustained listening skills
resources asked allow group students listen book time class help stay focused assigned task nannan
======================================================================
Word2Vec vector of sample essay:
======================================================================
[ 8.04053247e-02  2.71278403e-02  1.86810710e-02 -1.06767394e-01
 -6.99076960e-02 -3.88411532e-02 -3.05673048e+00  9.00463823e-02
 -2.26695073e-02  1.40511637e-01 -2.49908113e-02  2.18118590e-02
  3.96181379e-02 -1.60081977e-01 -7.50343540e-02 -1.69161548e-02
  1.06292274e-01 -7.48664452e-02  6.19590712e-02 -2.37311853e-02
  6.18812710e-02  8.20055355e-02 -4.84908516e-02 -4.78329468e-02
 -5.35464597e-02 -7.83328403e-02  1.42383595e-01 -9.23467194e-02
 -1.22140431e-01 -8.63069975e-02 -2.47617283e-01 -1.31099987e-01
 -1.07580484e-03  1.67824383e-01 -1.12144013e-01  3.46360403e-02
 -2.72766677e-02  2.00066824e-02 -9.23159619e-02 -5.85383508e-02
 -2.47435282e-02  1.19199242e-01 -3.57649919e-02 -1.29539698e-01
 -3.34161927e-02 -5.03548468e-02  5.91076282e-02  1.32662258e-02
  6.11301048e-03 -1.03835650e-01 -8.44336516e-02 -3.44283476e-02
  3.20616758e-02  4.85287113e-02  9.68183468e-03 -9.86782867e-02
  1.02881823e-01  4.16374677e-02 -1.42481061e-01 -1.98113944e-02
 -1.17003298e-01 -3.66207516e-02  1.40023010e-01 -4.12748839e-02
 -5.42611218e-02  1.76407265e-01  8.18976825e-02 -7.37007379e-02
  1.38213008e-01 -1.22521072e-01 -1.00791393e-01 -3.12209653e-02
  7.09197726e-03 -6.65999226e-02  2.74347008e-02 -1.45298136e-01
 -3.70908456e-02 -5.57335863e-02  1.73374984e-02 -2.71891621e-03
  5.60142268e-02 -4.81937613e-01  7.04253899e-02 -6.34147411e-02
 -9.45220250e-02 -8.15679839e-03  8.39531766e-02 -9.39358742e-02
  1.35498718e-01  6.06153476e-02  1.99985892e-02  6.50966129e-03
  1.70413952e-02 -1.95293116e-02 -1.11627016e-02 -8.72914698e-02
 -2.16736118e+00  7.84120031e-02  1.36583847e-01  1.53895598e-01
 -1.38013669e-01 -9.82099387e-02  4.05109096e-02 -8.65420065e-02
  1.23484977e-01  1.05060561e-01  5.55753960e-02 -1.94225535e-01
  5.30683669e-02  2.59358008e-02 -9.09505282e-02 -2.25075685e-02
  2.71481876e-02  2.19464767e-01  7.72835468e-02  6.34063306e-02
 -2.77994656e-01  8.42697145e-02  9.11616292e-02 -2.04185952e-02
  1.64286355e-02  7.14955097e-02  2.26737694e-02 -1.21612850e-01
  9.83905250e-02  2.86544242e-02  6.58204032e-02 -2.79655619e-02
  5.63599298e-02  1.92169481e-01  1.23362784e-01  6.32544347e-02
  5.57104274e-03 -2.02274798e-02  1.38208078e-02 -1.19226675e-01
  1.62928887e-01  4.09455951e-02  6.57385403e-02  2.27387536e-01
  1.07813208e-01 -8.78158629e-03  7.70487653e-02 -1.39009677e-02
 -5.66978569e-02 -7.00267234e-02  5.86138782e-02 -4.44312790e-02
  1.26711623e-01 -4.64740984e-02  3.25674468e-02 -6.43038952e-02
  9.83041900e-02 -2.86437984e-02 -2.35641081e-02  1.88243169e-02
  8.14130565e-02 -9.23944677e-03 -3.19389628e-02  1.97608705e-02
 -1.04124766e-02  4.03202240e-02 -3.29723008e-02 -5.60458903e-02
 -2.02139540e-02  4.06498315e-02 -1.08784093e-01  2.36387484e-02
```

```
  3.41461008e-02 -4.53625613e-02 -3.00004395e-02 -3.46059532e-02
 -8.54896548e-02 -1.57291592e-01  1.25815001e-02  5.47955927e-02
  3.34662702e-02  7.20953379e-02 -1.15523221e-01 -3.56347323e-02
 -4.44542258e-03  3.22374582e-01  5.52287097e-02  6.17586532e-03
 -1.49852860e-01 -6.34300492e-02 -6.98496748e-02 -8.77702726e-02
 -1.42558573e-02  4.71279274e-02  6.40853871e-03 -6.26968371e-02
 -6.60173669e-02 -4.26437879e-02  1.86767742e-04 -1.41862311e-01
 -1.00047040e-01  5.02840355e-02  4.51023419e-02 -1.20395030e-02
  1.68979310e-01 -2.28463168e-02 -4.21645331e-02  1.17911556e-01
 -1.16244620e-01  7.58341583e-02  8.04125145e-02 -9.69748976e-02
  1.61022525e-01  2.56078165e-02  2.61108282e-02  6.59638871e-02
  4.51038869e-02 -1.38105373e-01 -6.18844526e-02 -3.60016823e-02
 -2.76896903e-02 -2.76960640e-02  2.81076029e-02  2.03690734e-02
 -2.16812194e-01 -9.51436379e-02 -8.13094323e-02 -1.05158589e-01
 -1.91139270e+00  1.02521410e-01 -6.87829774e-02 -6.54025806e-04
 -2.24635056e-02 -1.54119855e-01 -5.04179895e-02  2.35751694e-03
 -3.64165298e-02 -7.31459790e-02 -5.61160040e-02 -4.14431734e-02
  8.19691532e-02 -9.36473758e-02  3.78327105e-02  1.08726787e-01
 -1.03098382e-01  2.65237548e-02 -2.62651035e-01  3.77857933e-02
 -7.97668774e-02 -5.79083871e-03 -1.20930944e-02 -9.17186182e-02
 -4.85489806e-02 -4.95635589e-02 -5.59186815e-02  1.18247589e-01
  9.50581694e-02 -5.80078065e-02  7.84473655e-02  1.13646435e-02
  4.38541508e-02 -7.43886855e-03  2.06446052e-01 -1.18897573e-01
 -4.63937326e-02  8.86716694e-03 -2.17241605e-02  4.11679266e-02
 -9.82906774e-03 -7.32796771e-02 -1.00514504e-01 -4.87719911e-02
  2.99347629e-02  1.44317709e-01 -5.23644887e-02  2.64863113e-02
 -1.45071448e-01  1.14036415e-01 -4.70707031e-02  2.29268484e-02
  7.15158702e-02  2.95910976e-02 -6.58551115e-02  8.87382242e-02
  3.94299274e-02  1.63121766e-02  5.36513629e-03  9.91709030e-02
 -3.31421048e-02  1.16199074e-01 -2.17670081e-02  4.57292839e-02
  2.44867887e-02 -1.99877197e-02  9.69959333e-02 -4.08975895e-02
 -4.05193597e-02 -7.35723226e-02  1.92458016e-02  1.56100490e-02
 -3.53157032e-02  2.06868844e-01  1.61429365e-01  9.13081210e-04]
```

2. Vectorizing project_title

In [160]:

```
word2VecTitlesVectors = getWord2VecVectors(preProcessedProjectTitlesWithoutStopWords);
```

In [161]:

```
print("Shape of Word2Vec vectorization matrix of project titles: {},
{}".format(len(word2VecTitlesVectors), len(word2VecTitlesVectors[0])));
equalsBorder(70);
print("Sample title: ");
equalsBorder(70);
print(preProcessedProjectTitlesWithoutStopWords[0]);
equalsBorder(70);
print("Word2Vec vector of sample title: ");
equalsBorder(70);
print(word2VecTitlesVectors[0]);
```

```
Shape of Word2Vec vectorization matrix of project titles: 12506, 300
======================================================================
Sample title:
======================================================================
listen audio supplies help us read
======================================================================
Word2Vec vector of sample title:
======================================================================
[ 7.98570667e-02 -4.65450000e-02 -3.76171667e-02 -7.31983333e-02
  1.11857500e-01  1.44636167e-01 -3.26398333e+00 -2.32417500e-01
  8.63553333e-02  3.44450000e-03 -1.30397667e-01 -7.60791667e-02
  2.42820000e-02  5.66993333e-02 -1.24239167e-01  2.62551667e-01
 -7.20953333e-02 -1.01589050e-01  1.92389000e-01  4.93213333e-02
  1.41266667e-01  2.00892667e-01 -1.63388333e-02 -4.48531667e-02
 -1.38146833e-01 -2.31317833e-01  9.67265000e-02  6.29203333e-02
 -2.25269333e-02 -1.51104083e-01 -5.20228333e-02  1.78300000e-02
  3.90050000e-02  2.83352167e-01  4.01120333e-02 -2.18770000e-02
  5.13500000e-02 -1.72878167e-01  1.05205000e-02 -2.64521667e-02
 -7.22511667e-02  1.53466667e-01 -5.56799667e-01  3.96451833e-02
 -5.38655000e-02  1.59274333e-01 -1.62432667e-01  1.51413667e-01
```

```
 -4.24883667e-02 -2.93152667e-01 -2.31540500e-01 -2.34066333e-01
 -2.43578333e-02  5.50681667e-02  5.83358333e-02  1.89183333e-02
  2.77866333e-01 -6.29300000e-02 -2.23638167e-01  3.35985000e-02
  1.14439167e-01  1.01550067e-01  1.87210000e-01 -3.20146667e-02
  2.58119667e-02  3.99919000e-01 -5.96788333e-02 -1.27184500e-01
 -6.91308333e-02 -1.94224333e-01  1.19180333e-01 -3.64303333e-02
  2.81331667e-02  1.44296667e-01 -2.85351667e-01 -1.74243167e-01
  5.00068333e-02  7.27918333e-02 -1.42686833e-01 -2.06733167e-01
  2.29274167e-01 -3.44450000e-02  1.22326667e-01  3.98724333e-02
  7.47781667e-02  6.56395000e-02  2.25084350e-01 -6.22579833e-02
  1.46614500e-01  9.62163333e-02 -1.51521833e-01  1.70759833e-01
 -1.67347167e-01  2.40296667e-01 -2.15395000e-02 -5.40800167e-01
 -2.11471667e+00 -7.27083333e-03  3.42198333e-01 -1.31817167e-01
 -2.29053333e-01  7.62038333e-02  9.30415000e-02  3.00506167e-02
  9.54086667e-02 -1.59811667e-02  7.63633333e-02  7.40311667e-02
  1.91654833e-01 -1.72218000e-01 -4.67771667e-02 -1.16485167e-01
 -1.62875728e-01  1.02005333e-01 -2.90292167e-01 -1.44476000e-01
  4.77841667e-02  2.35931667e-01 -3.41715333e-01 -3.45678333e-02
 -7.03500000e-03  8.44800000e-02  5.17833333e-03  1.31455000e-01
  1.16240000e-02  4.66800000e-02 -1.37122500e-01 -9.56596667e-02
  1.79570000e-01 -7.07951667e-02 -2.05940667e-01  1.14097333e-01
 -4.67410000e-02  1.71998333e-02  1.16424167e-01 -3.33013333e-01
 -1.68443333e-02  1.84866667e-03 -5.49270000e-02  5.37568050e-01
 -1.47383333e-03  3.90170833e-02  2.90549050e-01 -9.51247667e-02
 -1.57791167e-01 -2.21547000e-01 -1.86461967e-01 -8.25616667e-02
  3.11390833e-01 -1.26283667e-01 -3.05506500e-02 -1.90590500e-01
 -1.34447000e-01 -6.30850000e-02  1.40980000e-01 -7.83135000e-02
  4.88033333e-02  2.41715167e-01  1.86862000e-01  1.11505000e-01
 -7.24000000e-02 -1.59977167e-01 -1.57939500e-01 -4.00166667e-02
 -7.07062167e-02  1.46756667e-01 -3.40312667e-01  2.76253333e-01
  6.15863333e-02 -1.66749833e-01  5.23533333e-02 -2.07918333e-01
  1.40006867e-01 -2.54408167e-01  1.33175000e-02  2.25575333e-01
 -6.15436667e-02  3.10330500e-01  1.74742450e-01 -2.01009917e-01
 -1.29471667e-01  2.25081667e-01 -1.15617500e-01  1.35472167e-01
  8.80446667e-02 -3.63966667e-03  1.14940000e-01  3.87833333e-02
  2.46954500e-01  1.21291833e-01  1.32383667e-01 -1.20566333e-02
  1.79574833e-01 -1.07924617e-01 -8.74433333e-03 -4.38111667e-01
 -4.73868833e-02 -7.21473333e-02 -3.18661667e-03 -1.11531167e-01
  1.22203333e-01 -2.87569333e-01 -2.35429167e-01  1.24479833e-01
 -1.35696500e-02  4.51011667e-02  1.06295350e-01 -1.64480333e-01
  1.23836667e-01 -1.29548333e-02 -1.38785000e-01  2.80600000e-02
 -2.34735667e-01 -1.45072500e-01 -1.93016667e-02 -1.08647000e-01
  2.12495833e-01 -9.06095000e-02  3.89010500e-02 -1.42743333e-02
 -3.10239333e-01 -2.30244667e-02 -1.23720233e-01 -5.64761667e-02
 -2.59105000e+00  1.81343583e-01  4.62650000e-02  2.05858333e-01
 -1.77741000e-01 -1.55835000e-02 -2.18966667e-01 -1.74802900e-01
  9.90233333e-02  2.21530500e-01  3.93650000e-03  3.27433333e-01
  1.71638333e-02  1.11447000e-02  3.73068333e-02 -1.02175667e-02
 -9.01521667e-02 -3.82250000e-02 -4.43125000e-01 -3.01933333e-02
  1.69581667e-02 -2.63056167e-01  2.59191667e-02  3.40135000e-02
  1.52370000e-02 -2.73441818e-01 -2.79435000e-02  1.99005000e-01
 -7.88505000e-02 -1.13785500e-01  1.61408833e-01 -3.43983333e-02
 -7.21753333e-02  2.60264667e-01 -1.25152500e-01 -1.52103333e-01
 -9.02178333e-02 -6.90556667e-02 -1.65110667e-01 -4.41763333e-02
  4.33226667e-02 -1.50483500e-01  9.48083333e-02 -1.03709167e-01
 -1.57386167e-01  1.14041500e-01 -1.18139000e-01  3.92002500e-02
 -9.24768833e-02 -9.56626667e-02 -9.91246667e-02 -3.48065667e-01
  2.09311000e-01 -3.94355667e-02 -3.03979333e-02 -1.79162833e-01
 -2.57253500e-01 -3.74520000e-02 -6.84520000e-02  1.11695333e-01
 -1.45516667e-02  2.52513333e-02 -1.48378667e-01 -7.63836667e-02
 -6.34224333e-02 -1.86963333e-01 -4.38638333e-02 -9.77609000e-02
 -3.69500000e-02 -4.30113333e-02 -1.82664833e-01 -1.46422333e-01
 -1.92661667e-01  4.02389667e-01  2.54475000e-01 -2.93631667e-02]
```

Tf-Idf Weighted Word2Vec Vectorization

1. Vectorizing project_essay

In [162]:

```
# Initializing tfidf vectorizer
tfIdfEssayTempVectorizer = TfidfVectorizer();
# Vectorizing preprocessed essays using tfidf vectorizer initialized above
```

```
tfIdfEssayTempVectorizer.fit(preProcessedEssaysWithoutStopWords);
# Saving dictionary in which each word is key and it's idf is value
tfIdfEssayDictionary = dict(zip(tfIdfEssayTempVectorizer.get_feature_names(),
list(tfIdfEssayTempVectorizer.idf_)));
# Creating set of all unique words used by tfidf vectorizer
tfIdfEssayWords = set(tfIdfEssayTempVectorizer.get_feature_names());
```

In [163]:

```
# Creating list to save tf-idf weighted vectors of essays
tfIdfWeightedWord2VecEssaysVectors = [];
# Iterating over each essay
for essay in tqdm(preProcessedEssaysWithoutStopWords):
    # Sum of tf-idf values of all words in a particular essay
    cumulativeSumTfIdfWeightOfEssay = 0;
    # Tf-Idf weighted word2vec vector of a particular essay
    tfIdfWeightedWord2VecEssayVector = np.zeros(300);
    # Splitting essay into list of words
    splittedEssay = essay.split();
    # Iterating over each word
    for word in splittedEssay:
        # Checking if word is in glove words and set of words used by tfIdf essay vectorizer
        if (word in gloveWords) and (word in tfIdfEssayWords):
            # Tf-Idf value of particular word in essay
            tfIdfValueWord = tfIdfEssayDictionary[word] * (essay.count(word) / len(splittedEssay));
            # Making tf-idf weighted word2vec
            tfIdfWeightedWord2VecEssayVector += tfIdfValueWord * gloveModel[word];
            # Summing tf-idf weight of word to cumulative sum
            cumulativeSumTfIdfWeightOfEssay += tfIdfValueWord;
    if cumulativeSumTfIdfWeightOfEssay != 0:
        # Taking average of sum of vectors with tf-idf cumulative sum
        tfIdfWeightedWord2VecEssayVector = tfIdfWeightedWord2VecEssayVector /
cumulativeSumTfIdfWeightOfEssay;
    # Appending the above calculated tf-idf weighted vector of particular essay to list of vectors
of essays
    tfIdfWeightedWord2VecEssaysVectors.append(tfIdfWeightedWord2VecEssayVector);
```

In [164]:

```
print("Shape of Tf-Idf weighted Word2Vec vectorization matrix of project essays: {}, {}".format(le
n(tfIdfWeightedWord2VecEssaysVectors), len(tfIdfWeightedWord2VecEssaysVectors[0])));
equalsBorder(70);
print("Sample Essay: ");
equalsBorder(70);
print(preProcessedEssaysWithoutStopWords[0]);
equalsBorder(70);
print("Tf-Idf Weighted Word2Vec vector of sample essay: ");
equalsBorder(70);
print(tfIdfWeightedWord2VecEssaysVectors[0]);
```

```
Shape of Tf-Idf weighted Word2Vec vectorization matrix of project essays: 12506, 300
======================================================================
Sample Essay:
======================================================================
students range 9th 12th grade many special needs learning disabilities behavior disorders limited
english proficiency large percentage students come low income households work hard everything desp
ite difficulties dedicated students strive help esol students created clothes closet student counc
il organized school supply drives school neighboring schools pulled together raise money fellow st
udents serious illnesses inspire sense community often teens today refuse read anything longer
text message books reading level uninteresting books might interesting difficult give many high
school students listen higher comprehension level read particularly true students learning
difficulties students whose primary language not english however audio books help students
correctly pronouncing new vocabulary difficult names improving sustained listening skills
resources asked allow group students listen book time class help stay focused assigned task nannan
======================================================================
Tf-Idf Weighted Word2Vec vector of sample essay:
======================================================================
[ 8.07417917e-02  4.94947263e-03  4.20268615e-03 -9.65532643e-02
 -1.05325441e-01 -2.65419644e-02 -3.02745845e+00  4.98774551e-02
 -4.09733002e-02  2.14305364e-01 -6.70012547e-03  3.09923538e-02
  7.14302687e-03 -1.75054482e-01 -7.52570527e-02  2.35961504e-02
  1.62548960e-01 -8.05365069e-02  6.32701122e-02 -1.26233286e-02
  8.08305010e-02  1.31298449e-01 -3.28845712e-02 -5.04202672e-02
 -8.94122429e-02 -4.86225554e-02  1.73189139e-01 -7.60760721e-02
```

```
-1.72814894e-01 -9.65607738e-02 -1.98766751e-01 -1.80939120e-01
 1.77156382e-02  2.01805691e-01 -1.27519945e-01  7.24390798e-02
-3.45095951e-02  3.57591844e-02 -1.24147627e-01 -2.59390061e-02
 7.27593343e-03  8.90272940e-02 -2.99745873e-02 -1.66820230e-01
-2.06268635e-02 -6.38168793e-02  8.16756596e-02  3.14567030e-02
-9.66218092e-03 -1.12162897e-01 -1.09100472e-01 -3.89674684e-02
 6.01755957e-02  7.45478243e-02  1.50101418e-02 -1.39151193e-01
 1.47992714e-01  5.94804360e-02 -1.46739675e-01 -3.40062345e-02
-1.25895989e-01 -6.22624373e-03  1.85382697e-01 -7.38857022e-02
-5.95621650e-02  2.55485466e-01  6.44327470e-02 -8.80225502e-02
 1.79957764e-01 -1.69456706e-01 -1.38251170e-01 -7.10831410e-02
-3.13125326e-02 -1.12092284e-01  4.83295066e-02 -2.05565767e-01
-7.64956643e-02 -6.38769748e-02  1.12524735e-02  4.94878810e-03
 8.28626875e-02 -4.77244072e-01  1.22721685e-01 -7.02175896e-02
-1.01046319e-01  6.39168796e-03  1.28111688e-01 -1.10101856e-01
 2.12507925e-01  1.13630798e-01  5.15945107e-02  2.31770341e-02
-1.35818736e-02 -2.73186621e-02 -2.85813226e-02 -7.52318139e-02
-2.16642081e+00  9.75695266e-02  1.86665898e-01  1.81880119e-01
-1.91247549e-01 -1.11139062e-01  4.47704642e-02 -1.40571652e-01
 1.06711329e-01  1.08608005e-01  1.09878608e-01 -2.39813275e-01
 7.05818134e-02  3.51624033e-02 -1.40890975e-01 -8.57466529e-03
 3.84069250e-02  2.36860494e-01  9.32674783e-02  9.03248508e-02
-3.34606033e-01  8.60409748e-02  1.32203432e-01 -3.60614851e-02
 2.89550289e-04  9.16294299e-02  1.71825156e-02 -1.02376507e-01
 1.15588169e-01  7.41920698e-03  1.13163316e-01 -2.45263347e-02
 7.71375908e-02  2.80528404e-01  1.91408797e-01  3.56315965e-02
 2.93162882e-02 -2.50094364e-02  2.12345907e-02 -1.54046747e-01
 2.27395696e-01  6.70580371e-02  6.92447569e-02  2.30532232e-01
 1.09225853e-01 -2.65141215e-02  7.32638143e-02 -1.11600611e-02
-7.64742393e-02 -9.94287454e-02  6.32937036e-02 -5.71915047e-02
 1.09519446e-01 -6.68813915e-02  5.13679303e-02 -5.97378371e-02
 1.25405545e-01 -3.41681292e-02 -4.43743678e-02  3.93036696e-02
 1.09506948e-01  2.06317891e-03 -4.21010510e-02  3.33235859e-02
-3.96936760e-02  4.55378902e-02 -5.79571799e-02 -1.10886567e-01
-6.21691791e-02  5.99869298e-02 -1.11761865e-01  5.58014563e-02
 5.09611681e-02  5.53362535e-03 -4.66394542e-02 -4.25643619e-02
-1.30498091e-01 -1.78594467e-01 -4.65450808e-03  7.05394942e-02
 4.22513476e-02  8.29784424e-02 -1.63094070e-01 -3.74437064e-02
 5.08614265e-03  3.64360319e-01  7.71148691e-02  6.48838379e-03
-1.86708395e-01 -3.14051288e-02 -9.03827753e-02 -1.38490904e-01
-1.18013420e-02  4.40929188e-02  5.44210744e-03 -5.10792067e-02
-9.81974446e-02 -4.44694621e-02  2.01988109e-03 -1.71239766e-01
-1.33973746e-01  3.86062351e-02  5.31282640e-02  1.09220094e-02
 1.75732866e-01 -2.32577557e-02 -1.89566334e-03  1.52394153e-01
-1.36144719e-01  9.60460434e-02  9.51817229e-02 -9.55399773e-02
 2.05965177e-01  3.47438771e-02  5.31350248e-02  5.79152673e-02
 2.90349145e-02 -1.69266165e-01 -8.73269030e-02 -3.30124474e-02
-4.43212770e-02 -3.99803074e-02  4.53557991e-02  3.83101532e-02
-2.38196010e-01 -9.79806908e-02 -1.08015965e-01 -1.04793166e-01
-1.87387003e+00  8.08189282e-02 -8.00214805e-02  5.74331342e-03
-3.32940465e-02 -1.93599324e-01 -8.01713341e-02  4.20135499e-02
-2.62630021e-02 -8.61798518e-02 -4.00808657e-02 -8.00096340e-02
 1.03023819e-01 -1.02591187e-01  2.62036834e-02  1.42605720e-01
-1.59518873e-01  4.46548740e-02 -2.73315307e-01  4.62184659e-02
-1.04771120e-01 -1.64405141e-02 -6.66654398e-03 -8.13961553e-02
-6.78995741e-02 -8.51860170e-02 -6.58951361e-02  1.42238912e-01
 1.13011793e-01 -1.12436789e-01  8.07684045e-02  1.75441027e-02
 4.05546136e-02 -2.51335151e-02  2.71216393e-01 -1.51128459e-01
-7.55102182e-02  5.06721638e-02 -2.32395075e-02  8.82905243e-02
 7.36134125e-05 -6.37782516e-02 -1.06553121e-01 -6.34886338e-02
 4.16016079e-02  1.44455043e-01 -3.08784223e-02  3.78495943e-02
-1.73012568e-01  1.46634731e-01 -6.03210808e-02  1.48852575e-02
 7.49960099e-02  1.07904293e-02 -6.94577606e-02  1.17700733e-01
-1.14967232e-02  4.82344847e-02  3.33147702e-02  1.18952649e-01
-3.40942446e-02  1.40809708e-01 -5.29859415e-02  3.39566851e-02
 5.55208347e-02  3.55071296e-03  1.18094251e-01 -2.99136268e-02
-4.08510523e-02 -6.51141111e-02  1.41185125e-02  1.76016533e-02
-4.32594481e-02  2.51611043e-01  1.70779150e-01 -1.27838721e-02]
```

2. Vectorizing project_title

In [165]:

```
# Initializing tfidf vectorizer
tfIdfTitleTempVectorizer = TfidfVectorizer();
```

```
# Vectorizing preprocessed titles using tfidf vectorizer initialized above
tfIdfTitleTempVectorizer.fit(preProcessedProjectTitlesWithoutStopWords);
# Saving dictionary in which each word is key and it's idf is value
tfIdfTitleDictionary = dict(zip(tfIdfTitleTempVectorizer.get_feature_names(),
list(tfIdfTitleTempVectorizer.idf_)));
# Creating set of all unique words used by tfidf vectorizer
tfIdfTitleWords = set(tfIdfTitleTempVectorizer.get_feature_names());
```

In [166]:

```
# Creating list to save tf-idf weighted vectors of project titles
tfIdfWeightedWord2VecTitlesVectors = [];
# Iterating over each title
for title in tqdm(preProcessedProjectTitlesWithoutStopWords):
    # Sum of tf-idf values of all words in a particular project title
    cumulativeSumTfIdfWeightOfTitle = 0;
    # Tf-Idf weighted word2vec vector of a particular project title
    tfIdfWeightedWord2VecTitleVector = np.zeros(300);
    # Splitting title into list of words
    splittedTitle = title.split();
    # Iterating over each word
    for word in splittedTitle:
        # Checking if word is in glove words and set of words used by tfIdf title vectorizer
        if (word in gloveWords) and (word in tfIdfTitleWords):
            # Tf-Idf value of particular word in title
            tfIdfValueWord = tfIdfTitleDictionary[word] * (title.count(word) / len(splittedTitle));
            # Making tf-idf weighted word2vec
            tfIdfWeightedWord2VecTitleVector += tfIdfValueWord * gloveModel[word];
            # Summing tf-idf weight of word to cumulative sum
            cumulativeSumTfIdfWeightOfTitle += tfIdfValueWord;
    if cumulativeSumTfIdfWeightOfTitle != 0:
        # Taking average of sum of vectors with tf-idf cumulative sum
        tfIdfWeightedWord2VecTitleVector = tfIdfWeightedWord2VecTitleVector /
cumulativeSumTfIdfWeightOfTitle;
    # Appending the above calculated tf-idf weighted vector of particular title to list of vectors
of project titles
    tfIdfWeightedWord2VecTitlesVectors.append(tfIdfWeightedWord2VecTitleVector);
```

In [167]:

```
print("Shape of Tf-Idf weighted Word2Vec vectorization matrix of project titles: {}, {}".format(le
n(tfIdfWeightedWord2VecTitlesVectors), len(tfIdfWeightedWord2VecTitlesVectors[0])));
equalsBorder(70);
print("Sample Title: ");
equalsBorder(70);
print(preProcessedProjectTitlesWithoutStopWords[0]);
equalsBorder(70);
print("Tf-Idf Weighted Word2Vec vector of sample title: ");
equalsBorder(70);
print(tfIdfWeightedWord2VecTitlesVectors[0]);
```

```
Shape of Tf-Idf weighted Word2Vec vectorization matrix of project titles: 12506, 300
======================================================================
Sample Title:
======================================================================
listen audio supplies help us read
======================================================================
Tf-Idf Weighted Word2Vec vector of sample title:
======================================================================
[ 1.13681123e-01 -8.79667892e-02 -8.49680130e-02 -1.43604791e-02
  4.79478932e-02  1.35679761e-01 -3.21729604e+00 -2.19263883e-01
  9.20123892e-02 -1.81425987e-02 -8.51087335e-02 -8.38436422e-02
  1.30997300e-02  9.17590277e-02 -6.32389192e-02  2.78042755e-01
 -5.20654011e-02 -1.20769711e-01  2.10761397e-01  2.81098281e-02
  2.02254504e-01  2.26544882e-01  4.09948048e-02 -8.12726833e-02
 -1.47351385e-01 -2.46545553e-01  8.92049155e-02  1.17334499e-01
 -6.46631618e-02 -1.27686666e-01  6.40473188e-02  6.87940754e-02
 -9.98935534e-03  3.05227995e-01  2.82769186e-02  1.94452742e-02
  1.47647031e-01 -1.19005403e-01  2.25482358e-02  3.28239026e-03
 -1.00593813e-01  1.74006690e-01 -6.61577520e-01  3.03362895e-02
 -4.57622742e-02  1.95725136e-01 -1.80496637e-01  1.85440399e-01
 -7.24327807e-02 -2.96402438e-01 -2.54874085e-01 -1.89358305e-01
  1.49252778e-02 -1.67736411e-02  2.21725130e-02  5.83027949e-02
```

```
 2.66631408e-01 -1.15649342e-01 -2.56113990e-01 -1.23659563e-02
 1.26454580e-01  1.02401652e-01  2.26540966e-01 -2.48137198e-02
 3.77200134e-02  3.70666003e-01 -1.30708934e-01 -1.90202227e-01
-7.99640549e-02 -1.99278779e-01  1.88952302e-01 -5.28262387e-02
 8.94427229e-03  9.73452919e-02 -3.12781196e-01 -1.69794875e-01
 2.49004144e-02  9.09954469e-02 -1.41001179e-01 -2.59597216e-01
 2.17856378e-01 -5.44537121e-03  1.96734069e-01  4.33980288e-02
 1.50592999e-01  4.86817876e-02  2.23672943e-01 -5.85483370e-02
 1.81668965e-01  4.35235753e-02 -1.78701768e-01  2.17963049e-01
-1.83948726e-01  2.79406408e-01 -9.83039325e-02 -4.59391843e-01
-2.06472054e+00  6.55718559e-02  3.84570802e-01 -1.50736528e-01
-2.45024474e-01  9.54872308e-02  3.93022822e-02 -2.84601389e-03
 4.30746341e-02 -4.23682819e-02  4.24131671e-02  1.11888045e-01
 2.21643169e-01 -2.03699899e-01 -5.62467061e-02 -1.05412955e-01
-2.00608919e-01  8.15515530e-02 -2.86854852e-01 -1.23250427e-01
 1.01324855e-01  2.08094532e-01 -2.85102213e-01 -5.02233893e-02
-5.89330620e-02  8.48416861e-02  3.99899937e-02  2.19888761e-01
 2.69947733e-02  6.37483795e-02 -1.63953056e-01 -5.44146090e-02
 1.74876741e-01 -8.45327103e-02 -1.89025352e-01  1.25898040e-01
-7.00013824e-02 -3.92090101e-03  1.02184029e-01 -2.96449413e-01
 9.24024349e-03  2.60543238e-03 -6.45232259e-02  4.55837258e-01
-1.95166006e-02  4.29111197e-02  3.40995763e-01 -1.27754647e-01
-2.28798799e-01 -2.67147427e-01 -1.93529910e-01 -1.04209551e-01
 2.79514726e-01 -1.43789955e-01 -2.91374475e-02 -2.29476794e-01
-1.99324188e-01 -6.55582686e-02  1.39288329e-01 -7.72124376e-02
 7.32921485e-02  3.02730432e-01  2.31300575e-01  1.24397202e-01
-5.84432913e-02 -1.82942572e-01 -1.28491465e-01 -3.61454852e-02
-7.35113656e-02  1.90406137e-01 -3.81739114e-01  2.45446874e-01
-4.89411497e-02 -1.52600038e-01  1.08712387e-01 -2.16709435e-01
 1.71316119e-01 -2.79418052e-01  5.13338876e-02  2.05302961e-01
-7.57570565e-02  3.30449232e-01  1.75821796e-01 -2.35416838e-01
-1.02836283e-01  2.18368948e-01 -6.32439431e-02  9.63049280e-02
 1.57055749e-01  1.30164743e-02  1.05368413e-01  1.88753687e-02
 2.24688418e-01  1.33015623e-01  1.84785756e-01 -3.71227950e-02
 2.03598798e-01 -7.00647259e-02 -2.61961085e-02 -4.66537709e-01
-1.72049870e-02 -3.86819385e-02 -4.04192978e-02 -1.55184799e-01
 1.05858524e-01 -4.00966162e-01 -2.22768230e-01  1.62766950e-01
 1.42797676e-02  7.25957783e-02  1.46475104e-01 -1.99874784e-01
 9.95366688e-02 -8.21043134e-03 -1.81367921e-01  8.87537176e-03
-2.65719939e-01 -1.27624203e-01 -2.95885132e-02 -1.23267580e-01
 2.23209452e-01 -8.22534656e-02  6.31007037e-02 -2.16852776e-02
-3.30762860e-01 -5.59438052e-02 -1.12191047e-01 -5.47383018e-02
-2.57472166e+00  1.61482830e-01  4.59702857e-02  1.76699055e-01
-2.18170968e-01 -2.62395711e-02 -2.92279349e-01 -1.67183040e-01
 1.93731095e-01  1.78716888e-01  3.56821539e-04  3.23676931e-01
-3.23126694e-02  5.94701577e-02  5.40123148e-02 -1.52538840e-02
-1.03867784e-01 -1.16091132e-01 -4.18103807e-01  2.01820992e-02
 1.96609350e-02 -3.00953903e-01  3.47754035e-02  1.01313745e-01
 1.13843868e-03 -2.97253917e-01 -2.56986739e-02  2.53707063e-01
-1.72799611e-01 -1.06102455e-01  1.45092265e-01 -1.41456005e-02
-1.06498996e-01  2.77194057e-01 -1.13590336e-01 -2.18564925e-01
-1.19370964e-01 -5.26163535e-02 -1.32481928e-01 -1.41306951e-03
 7.95607297e-02 -2.01947786e-01  3.14543523e-02 -1.26723536e-01
-2.21698273e-01  8.34701807e-02 -1.23249129e-01  9.71243915e-02
-1.07317642e-01 -6.07471879e-02 -1.11283685e-01 -4.11182461e-01
 2.02445086e-01 -4.46027582e-02 -3.89674115e-02 -1.54096906e-01
-3.28446199e-01 -1.01072257e-01 -6.03397465e-02  1.33439131e-01
-6.87762625e-02 -1.71472161e-02 -1.70246544e-01 -1.08540839e-01
-3.49590512e-02 -1.54225735e-01 -7.83420663e-02 -1.03708102e-01
-1.66811267e-02 -5.33282573e-02 -1.99718139e-01 -1.53478362e-01
-1.93844277e-01  4.32699812e-01  2.45761237e-01 -1.71351913e-02]
```

Vectorizing numerical features

1. Vectorizing price

In [168]:

```
# Standardizing the price data using StandardScaler(Uses mean and std for standardization)
priceScaler = StandardScaler();
priceScaler.fit(trainingData['price'].values.reshape(-1, 1));
priceStandardized = priceScaler.transform(trainingData['price'].values.reshape(-1, 1));
```

In [169]:

```
print("Shape of standardized matrix of prices: ", priceStandardized.shape);
equalsBorder(70);
print("Sample original prices: ");
equalsBorder(70);
print(trainingData['price'].values[0:5]);
print("Sample standardized prices: ");
equalsBorder(70);
print(priceStandardized[0:5]);
```

```
Shape of standardized matrix of prices:  (12506, 1)
======================================================================
Sample original prices:
======================================================================
[ 74.41 304.7  363.84 251.84 792.92]
Sample standardized prices:
======================================================================
[[-0.70727539]
 [-0.05510812]
 [ 0.11237276]
 [-0.20480442]
 [ 1.32750121]]
```

2. Vectorizing quantity

In [170]:

```
# Standardizing the quantity data using StandardScaler(Uses mean and std for standardization)
quantityScaler = StandardScaler();
quantityScaler.fit(trainingData['quantity'].values.reshape(-1, 1));
quantityStandardized = quantityScaler.transform(trainingData['quantity'].values.reshape(-1, 1));
```

In [171]:

```
print("Shape of standardized matrix of quantities: ", quantityStandardized.shape);
equalsBorder(70);
print("Sample original quantities: ");
equalsBorder(70);
print(trainingData['quantity'].values[0:5]);
print("Sample standardized quantities: ");
equalsBorder(70);
print(quantityStandardized[0:5]);
```

```
Shape of standardized matrix of quantities:  (12506, 1)
======================================================================
Sample original quantities:
======================================================================
[ 8  1 17 34 42]
Sample standardized quantities:
======================================================================
[[-0.37621124]
 [-0.61673036]
 [-0.06697236]
 [ 0.51714552]
 [ 0.79202452]]
```

3. Vectorizing teacher_number_of_previously_posted_projects

In [172]:

```
# Standardizing the teacher_number_of_previously_posted_projects data using StandardScaler(Uses me
an and std for standardization)
previouslyPostedScaler = StandardScaler();
previouslyPostedScaler.fit(trainingData['teacher_number_of_previously_posted_projects'].values.res
hape(-1, 1));
previouslyPostedStandardized =
previouslyPostedScaler.transform(trainingData['teacher_number_of_previously_posted_projects'].valu
es.reshape(-1, 1));
```

In [173]:

```
print("Shape of standardized matrix of teacher_number_of_previously_posted_projects: ",
previouslyPostedStandardized.shape);
equalsBorder(70);
print("Sample original quantities: ");
equalsBorder(70);
print(trainingData['teacher_number_of_previously_posted_projects'].values[0:5]);
print("Sample standardized teacher_number_of_previously_posted_projects: ");
equalsBorder(70);
print(previouslyPostedStandardized[0:5]);
```

```
Shape of standardized matrix of teacher_number_of_previously_posted_projects:  (12506, 1)
======================================================================
Sample original quantities:
======================================================================
[10  1  1  0  0]
Sample standardized teacher_number_of_previously_posted_projects:
======================================================================
[[ 0.07504662]
 [-0.34626102]
 [-0.34626102]
 [-0.39307298]
 [-0.39307298]]
```

In [174]:

```
numberOfPoints = 16638;
# Categorical data
categoriesVectorsSub = categoriesVectors[0:numberOfPoints];
subCategoriesVectorsSub = subCategoriesVectors[0:numberOfPoints];
teacherPrefixVectorsSub = teacherPrefixVectors[0:numberOfPoints];
schoolStateVectorsSub = schoolStateVectors[0:numberOfPoints];
projectGradeVectorsSub = projectGradeVectors[0:numberOfPoints];

# Text data
bowEssayModelSub = bowEssayModel[0:numberOfPoints];
bowTitleModelSub = bowTitleModel[0:numberOfPoints];
tfIdfEssayModelSub = tfIdfEssayModel[0:numberOfPoints];
tfIdfTitleModelSub = tfIdfTitleModel[0:numberOfPoints];
word2VecEssaysVectorsSub = word2VecEssaysVectors[0:numberOfPoints];
word2VecTitlesVectorsSub = word2VecTitlesVectors[0:numberOfPoints];
tfIdfWeightedWord2VecEssaysVectorsSub = tfIdfWeightedWord2VecEssaysVectors[0:numberOfPoints];
tfIdfWeightedWord2VecTitlesVectorsSub = tfIdfWeightedWord2VecTitlesVectors[0:numberOfPoints];

# Numerical data
priceStandardizedSub = priceStandardized[0:numberOfPoints];
quantityStandardizedSub = quantityStandardized[0:numberOfPoints];
previouslyPostedStandardizedSub = previouslyPostedStandardized[0:numberOfPoints];
```

In [175]:

```
def getAvgTfIdfEssayVectors(arrayOfTexts):
    # Creating list to save tf-idf weighted vectors of essays
    tfIdfWeightedWord2VecEssaysVectors = [];
    # Iterating over each essay
    for essay in tqdm(arrayOfTexts):
        # Sum of tf-idf values of all words in a particular essay
        cumulativeSumTfIdfWeightOfEssay = 0;
        # Tf-Idf weighted word2vec vector of a particular essay
        tfIdfWeightedWord2VecEssayVector = np.zeros(300);
        # Splitting essay into list of words
        splittedEssay = essay.split();
        # Iterating over each word
        for word in splittedEssay:
            # Checking if word is in glove words and set of words used by tfIdf essay vectorizer
            if (word in gloveWords) and (word in tfIdfEssayWords):
                # Tf-Idf value of particular word in essay
                tfIdfValueWord = tfIdfEssayDictionary[word] * (essay.count(word) /
len(splittedEssay));
                # Making tf-idf weighted word2vec
                tfIdfWeightedWord2VecEssayVector += tfIdfValueWord * gloveModel[word];
                # Summing tf-idf weight of word to cumulative sum
                cumulativeSumTfIdfWeightOfEssay += tfIdfValueWord;
```

```
        if cumulativeSumTfIdfWeightOfEssay != 0:
            # Taking average of sum of vectors with tf-idf cumulative sum
            tfIdfWeightedWord2VecEssayVector = tfIdfWeightedWord2VecEssayVector /
cumulativeSumTfIdfWeightOfEssay;
        # Appending the above calculated tf-idf weighted vector of particular essay to list of
vectors of essays
        tfIdfWeightedWord2VecEssaysVectors.append(tfIdfWeightedWord2VecEssayVector);
    return tfIdfWeightedWord2VecEssaysVectors;
```

In [176]:

```
def getAvgTfIdfTitleVectors(arrayOfTexts):
    # Creating list to save tf-idf weighted vectors of project titles
    tfIdfWeightedWord2VecTitlesVectors = [];
    # Iterating over each title
    for title in tqdm(arrayOfTexts):
        # Sum of tf-idf values of all words in a particular project title
        cumulativeSumTfIdfWeightOfTitle = 0;
        # Tf-Idf weighted word2vec vector of a particular project title
        tfIdfWeightedWord2VecTitleVector = np.zeros(300);
        # Splitting title into list of words
        splittedTitle = title.split();
        # Iterating over each word
        for word in splittedTitle:
            # Checking if word is in glove words and set of words used by tfIdf title vectorizer
            if (word in gloveWords) and (word in tfIdfTitleWords):
                # Tf-Idf value of particular word in title
                tfIdfValueWord = tfIdfTitleDictionary[word] * (title.count(word) /
len(splittedTitle));
                # Making tf-idf weighted word2vec
                tfIdfWeightedWord2VecTitleVector += tfIdfValueWord * gloveModel[word];
                # Summing tf-idf weight of word to cumulative sum
                cumulativeSumTfIdfWeightOfTitle += tfIdfValueWord;
        if cumulativeSumTfIdfWeightOfTitle != 0:
            # Taking average of sum of vectors with tf-idf cumulative sum
            tfIdfWeightedWord2VecTitleVector = tfIdfWeightedWord2VecTitleVector /
cumulativeSumTfIdfWeightOfTitle;
        # Appending the above calculated tf-idf weighted vector of particular title to list of
vectors of project titles
        tfIdfWeightedWord2VecTitlesVectors.append(tfIdfWeightedWord2VecTitleVector);
    return tfIdfWeightedWord2VecTitlesVectors;
```

In [177]:

```
resultsDataFrame = pd.DataFrame(columns =  ['Vectorizer', 'Model', 'Hyper Parameter - K', 'AUC']);
resultsDataFrame
```

Out[177]:

	Vectorizer	Model	Hyper Parameter - K	AUC

Preparing cross-validation data to perform analysis

In [178]:

```
# Cross-validate data categorical features transformation
categoriesTransformedCrossValidateData = subjectsCategoriesVectorizer.transform(crossValidateData[
'cleaned_categories']);
subCategoriesTransformedCrossValidateData =
subjectsSubCategoriesVectorizer.transform(crossValidateData['cleaned_sub_categories']);
teacherPrefixTransformedCrossValidateData = teacherPrefixVectorizer.transform(crossValidateData['t
eacher_prefix']);
schoolStateTransformedCrossValidateData =
schoolStateVectorizer.transform(crossValidateData['school_state']);
projectGradeTransformedCrossValidateData = projectGradeVectorizer.transform(crossValidateData['pro
ject_grade_category']);

# Cross-validate data text features transformation
preProcessedEssaysTemp = preProcessingWithAndWithoutStopWords(crossValidateData['project_essay'])[
1];
preProcessedTitlesTemp = preProcessingWithAndWithoutStopWords(crossValidateData['project_title'])[
```

```
1];
bowEssayTransformedCrossValidateData = bowEssayVectorizer.transform(preProcessedEssaysTemp);
bowTitleTransformedCrossValidateData = bowTitleVectorizer.transform(preProcessedTitlesTemp);
tfIdfEssayTransformedCrossValidateData = tfIdfEssayVectorizer.transform(preProcessedEssaysTemp);
tfIdfTitleTransformedCrossValidateData = tfIdfTitleVectorizer.transform(preProcessedTitlesTemp);
word2VecEssayTransformedCrossValidateData = getWord2VecVectors(preProcessedEssaysTemp);
word2VecTitleTransformedCrossValidateData = getWord2VecVectors(preProcessedTitlesTemp);
tfIdfWeightedEssayTransformedCrossValidateData = getAvgTfIdfEssayVectors(preProcessedEssaysTemp);
tfIdfWeightedTitleTransformedCrossValidateData = getAvgTfIdfTitleVectors(preProcessedTitlesTemp);

# Cross-validate numerical features transformation
priceTransformedCrossValidateData =
priceScaler.transform(crossValidateData['price'].values.reshape(-1, 1));
quantityTransformedCrossValidateData =
quantityScaler.transform(crossValidateData['quantity'].values.reshape(-1, 1));
previouslyPostedTransformedCrossValidateData = previouslyPostedScaler.transform(crossValidateData[
'teacher_number_of_previously_posted_projects'].values.reshape(-1, 1));
```

Preparing test data to perform analysis

In [179]:

```
# Test data categorical features transformation
categoriesTransformedTestData =
subjectsCategoriesVectorizer.transform(testData['cleaned_categories']);
subCategoriesTransformedTestData =
subjectsSubCategoriesVectorizer.transform(testData['cleaned_sub_categories']);
teacherPrefixTransformedTestData = teacherPrefixVectorizer.transform(testData['teacher_prefix']);
schoolStateTransformedTestData = schoolStateVectorizer.transform(testData['school_state']);
projectGradeTransformedTestData =
projectGradeVectorizer.transform(testData['project_grade_category']);

# Test data text features transformation
preProcessedEssaysTemp = preProcessingWithAndWithoutStopWords(testData['project_essay'])[1];
preProcessedTitlesTemp = preProcessingWithAndWithoutStopWords(testData['project_title'])[1];
bowEssayTransformedTestData = bowEssayVectorizer.transform(preProcessedEssaysTemp);
bowTitleTransformedTestData = bowTitleVectorizer.transform(preProcessedTitlesTemp);
tfIdfEssayTransformedTestData = tfIdfEssayVectorizer.transform(preProcessedEssaysTemp);
tfIdfTitleTransformedTestData = tfIdfTitleVectorizer.transform(preProcessedTitlesTemp);
word2VecEssayTransformedTestData = getWord2VecVectors(preProcessedEssaysTemp);
word2VecTitleTransformedTestData = getWord2VecVectors(preProcessedTitlesTemp);
tfIdfWeightedEssayTransformedTestData = getAvgTfIdfEssayVectors(preProcessedEssaysTemp);
tfIdfWeightedTitleTransformedTestData = getAvgTfIdfTitleVectors(preProcessedTitlesTemp);

# Test data numerical features transformation
priceTransformedTestData = priceScaler.transform(testData['price'].values.reshape(-1, 1));
quantityTransformedTestData = quantityScaler.transform(testData['quantity'].values.reshape(-1, 1));
previouslyPostedTransformedTestData =
previouslyPostedScaler.transform(testData['teacher_number_of_previously_posted_projects'].values.r
eshape(-1, 1));
```

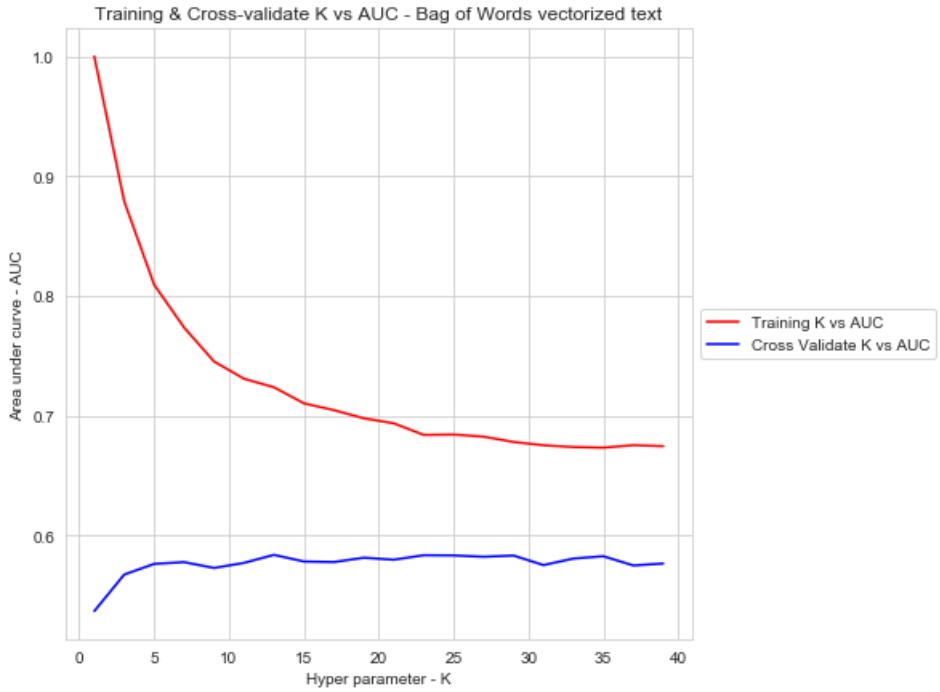
Analysis on Imbalanced Dataset Using K-NN(Simple Cross Validation)

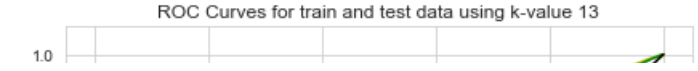
In [133]:

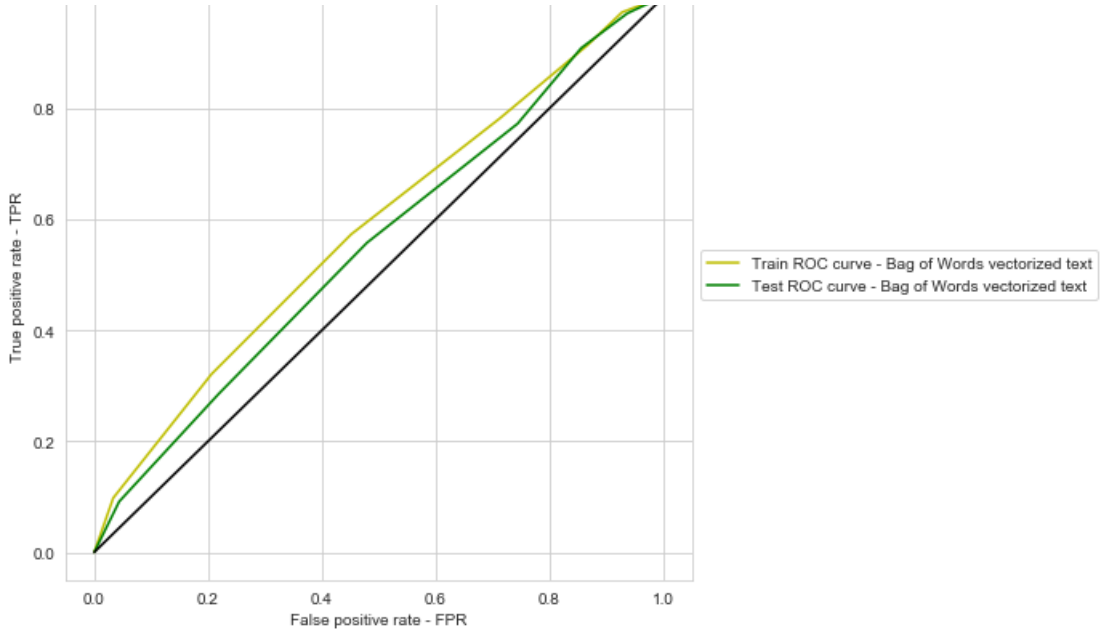
```
testKValues = np.arange(1, 40, 2);
techniques = ['Bag of Words', 'Tf-Idf', 'Average Word2Vec', 'Tf-Idf Weighted Word2Vec'];
for index, technique in enumerate(techniques):
    areaUnderRocValuesTrain = [];
    areaUnderRocValuesCrossValidate = [];
    trainingMergedData = hstack((categoriesVectorsSub,\
                                 subCategoriesVectorsSub,\
                                 teacherPrefixVectorsSub,\
                                 schoolStateVectorsSub,\
```

```
                                    projectGradeVectorsSub,\
                                    priceStandardizedSub,\
                                    previouslyPostedStandardizedSub));
    crossValidateMergedData = hstack((categoriesTransformedCrossValidateData,\
                                        subCategoriesTransformedCrossValidateData,\
                                        teacherPrefixTransformedCrossValidateData,\
                                        schoolStateTransformedCrossValidateData,\
                                        projectGradeTransformedCrossValidateData,\
                                        priceTransformedCrossValidateData,\
                                        previouslyPostedTransformedCrossValidateData));
    testMergedData = hstack((categoriesTransformedTestData,\
                                        subCategoriesTransformedTestData,\
                                        teacherPrefixTransformedTestData,\
                                        schoolStateTransformedTestData,\
                                        projectGradeTransformedTestData,\
                                        priceTransformedTestData,\
                                        previouslyPostedTransformedTestData));

    if(index == 0):
        trainingMergedData = hstack((trainingMergedData,\
                                    bowTitleModelSub,\
                                    bowEssayModelSub));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                        bowTitleTransformedCrossValidateData,\
                                        bowEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                bowTitleTransformedTestData,\
                                bowEssayTransformedTestData));
    elif(index == 1):
        trainingMergedData = hstack((trainingMergedData,\
                                    tfIdfTitleModelSub,\
                                    tfIdfEssayModelSub));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                        tfIdfTitleTransformedCrossValidateData,\
                                        tfIdfEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                tfIdfTitleTransformedTestData,\
                                tfIdfEssayTransformedTestData));
    elif(index == 2):
        trainingMergedData = hstack((trainingMergedData,\
                                    word2VecTitlesVectorsSub,\
                                    word2VecEssaysVectorsSub));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                        word2VecTitleTransformedCrossValidateData,\
                                        word2VecEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                word2VecTitleTransformedTestData,\
                                word2VecEssayTransformedTestData));
    elif(index == 3):
        trainingMergedData = hstack((trainingMergedData,\
                                    tfIdfWeightedWord2VecTitlesVectorsSub,\
                                    tfIdfWeightedWord2VecEssaysVectorsSub));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                        tfIdfWeightedTitleTransformedCrossValidateData,\
                                        tfIdfWeightedEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                tfIdfWeightedTitleTransformedTestData,\
                                tfIdfWeightedEssayTransformedTestData));
    for testKValue in tqdm(testKValues):
        knnClassifier = KNeighborsClassifier(n_neighbors = testKValue, algorithm = 'brute');
        knnClassifier.fit(trainingMergedData, classesTraining);
        predProbScores = knnClassifier.predict_proba(trainingMergedData);
        fpr, tpr, threshold = roc_curve(classesTraining, predProbScores[:, 1]);
        areaUnderRocValuesTrain.append(auc(fpr, tpr));
        predProbScores = knnClassifier.predict_proba(crossValidateMergedData);
        fpr, tpr, threshold = roc_curve(classesCrossValidate, predProbScores[:, 1]);
        areaUnderRocValuesCrossValidate.append(auc(fpr, tpr));

    plt.plot(testKValues, areaUnderRocValuesTrain, 'r', label = "Training K vs AUC");
    plt.plot(testKValues, areaUnderRocValuesCrossValidate, 'b', label = "Cross Validate K vs AUC");
    plt.title("Training & Cross-validate K vs AUC - {} vectorized text".format(technique));
    plt.xlabel("Hyper parameter - K");
    plt.ylabel("Area under curve - AUC");
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.show();

    optimalKValue = testKValues[np.argmax(areaUnderRocValuesCrossValidate)];
```

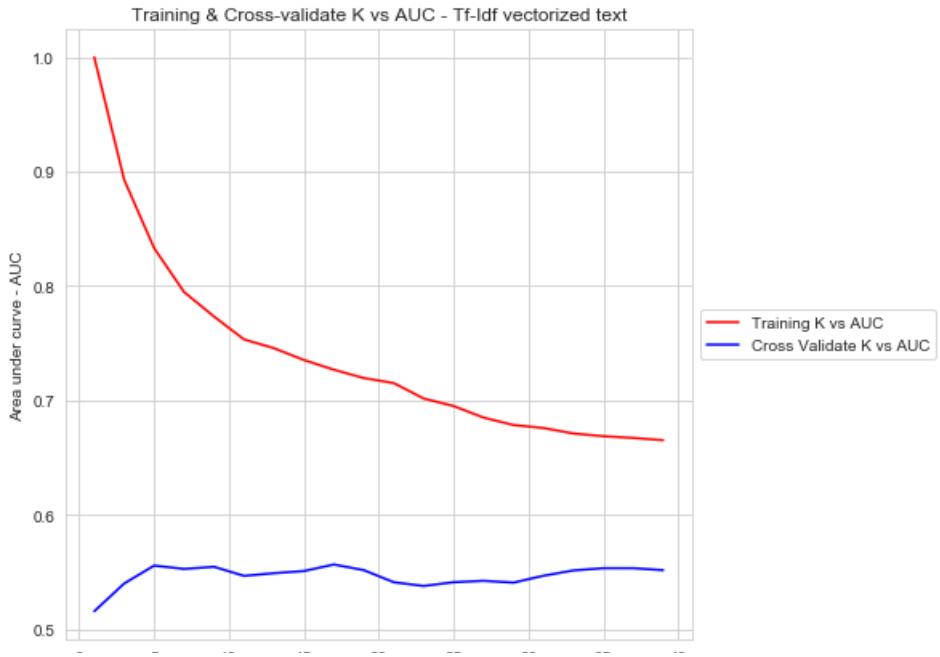
```
    knnClassifier = KNeighborsClassifier(n_neighbors = optimalKValue, algorithm = 'brute');
    knnClassifier.fit(trainingMergedData, classesTraining);
    predProbScoresCrossValidate = knnClassifier.predict_proba(crossValidateMergedData);
    fprCrossValidate, tprCrossValidate, thresholdTrain = roc_curve(classesCrossValidate, predProbScoresCrossValidate[:, 1]);
    predProbScoresTest = knnClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predProbScoresTest[:, 1]);
    areaUnderRocValueTest = auc(fprTest, tprTest);
    plt.plot(fprCrossValidate, tprCrossValidate, 'y', label="Train ROC curve - {} vectorized text".format(technique));
    plt.plot(fprTest, tprTest, 'g', label="Test ROC curve - {} vectorized text".format(technique));
    plt.plot([0, 1], [0, 1], 'k-');
    plt.title("ROC Curves for train and test data using k-value {}".format(optimalKValue))
    plt.xlabel('False positive rate - FPR');
    plt.ylabel('True positive rate - TPR');
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.show();

    print("Results of analysis using {} vectorized text features merged with other features using K-NN brute force algorithm:".format(technique));
    equalsBorder(70);
    print("AUC values of cross-validate data: ");
    equalsBorder(40);
    print(areaUnderRocValuesCrossValidate);
    equalsBorder(40);
    print("Optimal K-Value: ", optimalKValue);
    equalsBorder(40);
    print("AUC value of test data: ", areaUnderRocValueTest);
    # Predicting classes of test data projects
    predictionClassesTest = knnClassifier.predict(testMergedData);
    equalsBorder(40);
    # Adding results to results dataframe
    resultsDataFrame = resultsDataFrame.append({'Vectorizer': technique, 'Model': 'Brute', 'Hyper Parameter - K': optimalKValue, 'AUC': areaUnderRocValueTest}, ignore_index = True);
    # Printing confusion matrix
    confusionMatrix = confusion_matrix(classesTest, predictionClassesTest);
    # Creating dataframe for generated confusion matrix
    confusionMatrixDataFrame = pd.DataFrame(data = confusionMatrix, index = ['Actual: NO', 'Actual: YES'], columns = ['Predicted: NO', 'Predicted: YES']);
    print("Confusion Matrix : ");
    equalsBorder(60);
    print(confusionMatrixDataFrame);
    equalsBorder(110);
    equalsBorder(110);
    equalsBorder(110);
```

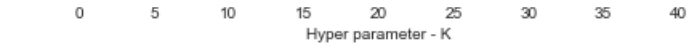


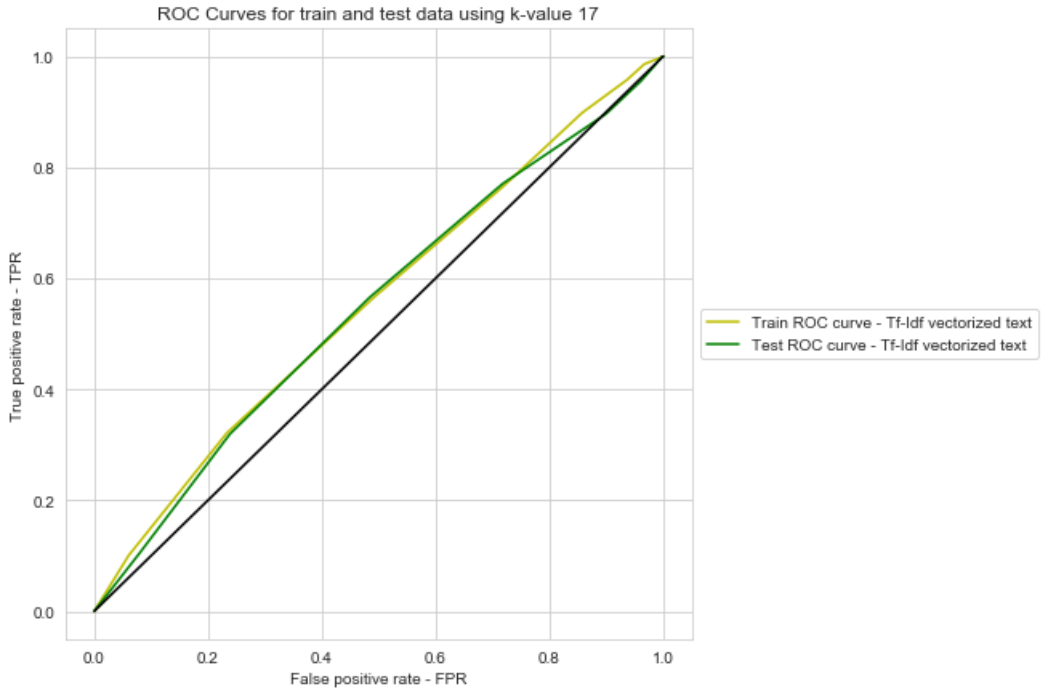




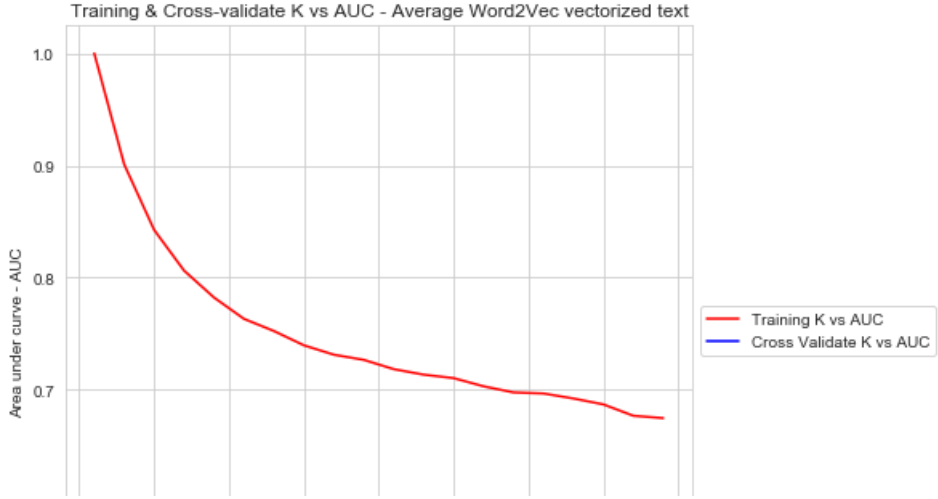

```
Results of analysis using Bag of Words vectorized text features merged with other features using K
-NN brute force algorithm:
========================================================================
AUC values of cross-validate data:
========================================
[0.5368002054442733, 0.5673591850710494, 0.5762857387433659, 0.5777306967984934,
0.5728668036295155, 0.5771143639787708, 0.5838212634822804, 0.5782691319979456,
0.5778051703475433, 0.5814244136278036, 0.5797508988186953, 0.5835225132682761,
0.5833504536894368, 0.5822282143468585, 0.5832451634994008, 0.5752722136620442,
0.5807327512412258, 0.5826596473206642, 0.5749606231809622, 0.5766016093134738]
========================================
Optimal K-Value:  13
========================================
AUC value of test data:  0.5532327554680933
========================================
Confusion Matrix :
============================================================
             Predicted: NO  Predicted: YES
Actual: NO                9              448
Actual: YES              20             2523
================================================================================================================
=======
================================================================================================================
=======
================================================================================================================
=======
```

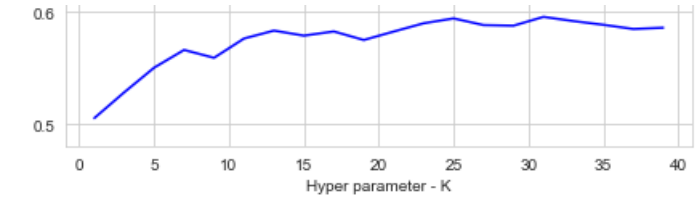


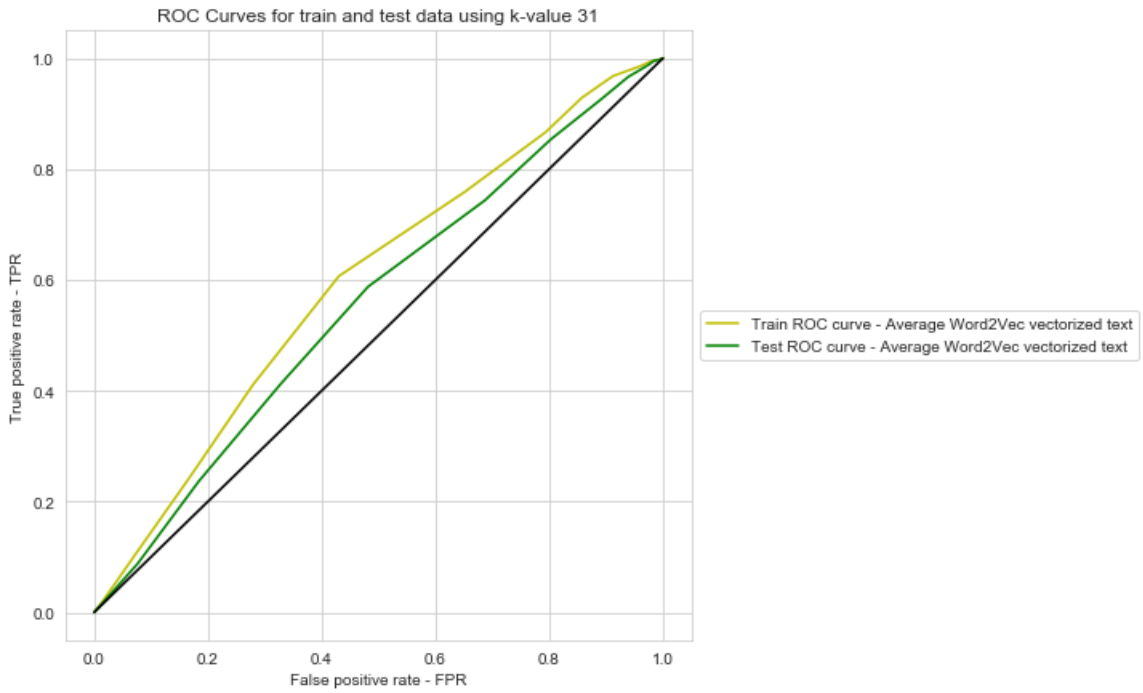




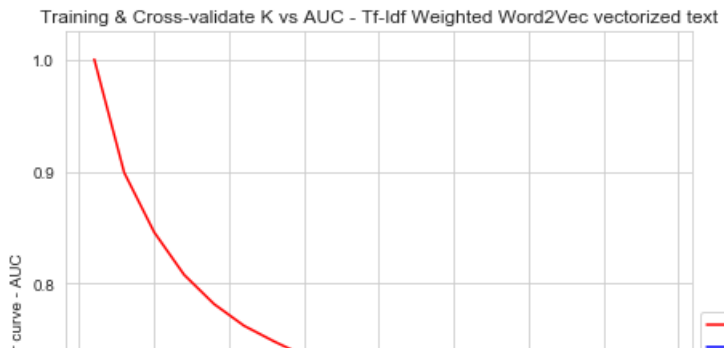

```
Results of analysis using Tf-Idf vectorized text features merged with other features using K-NN br
ute force algorithm:
========================================================================
AUC values of cross-validate data:
========================================
[0.5154340010272214, 0.5397722992638248, 0.5554143126177025, 0.5525055641157336,
0.5543374422187981, 0.5464252696456087, 0.5487647663071391, 0.550637733264852, 0.5563670604348571,
0.5514363978770759, 0.5408483136449238, 0.5375817497003939, 0.5408782742681048,
0.5421023797294983, 0.5405230268789593, 0.5465237116932031, 0.5511736004108885,
0.5531296010957027, 0.5531390172915597, 0.5514098613251156]
========================================
Optimal K-Value:  17
========================================
AUC value of test data:  0.5491278672048641
========================================
Confusion Matrix :
============================================================
             Predicted: NO  Predicted: YES
Actual: NO               1             456
Actual: YES              2            2541
================================================================================================================
=======
================================================================================================================
=======
================================================================================================================
=======
```

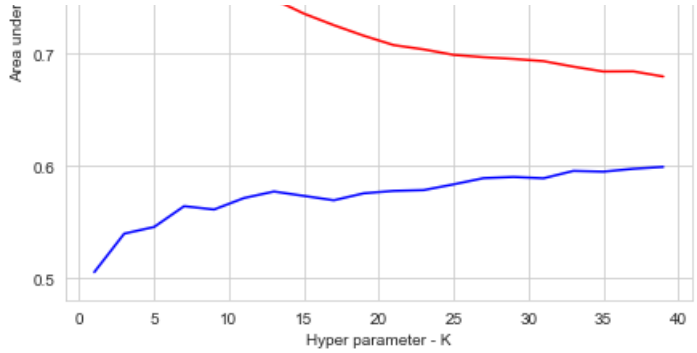


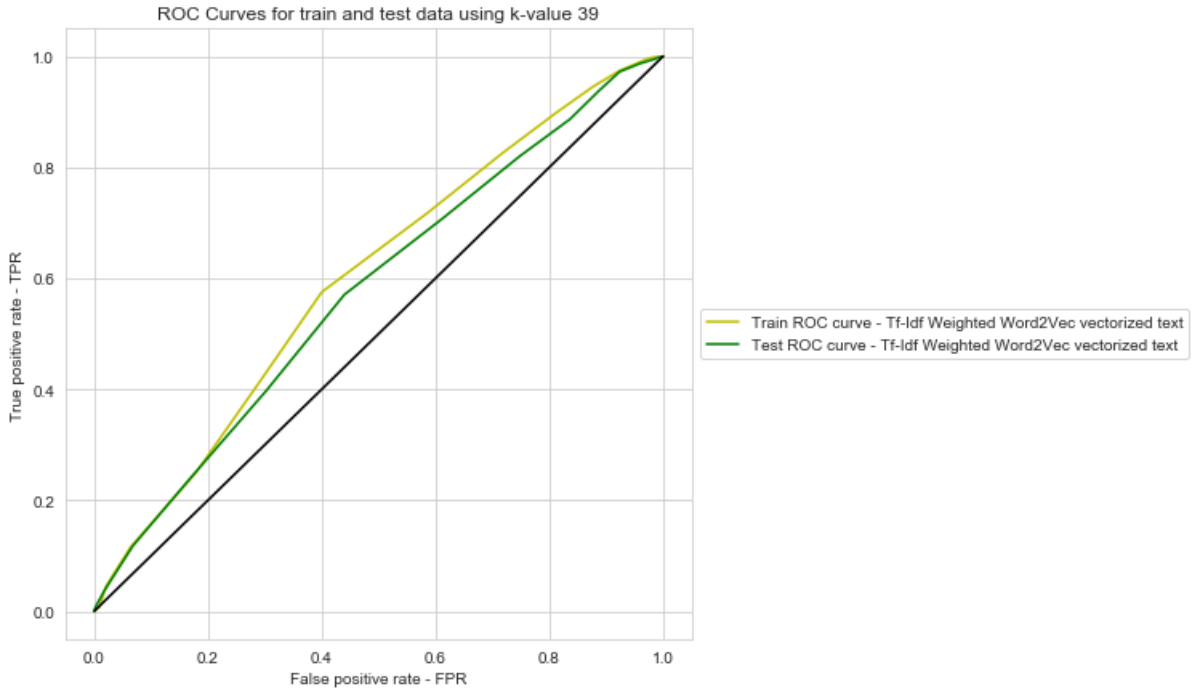





```
Results of analysis using Average Word2Vec vectorized text features merged with other features usi
ng K-NN brute force algorithm:
========================================================================
AUC values of cross-validate data:
========================================
[0.5055213148433487, 0.5285430576956002, 0.5505410032528676, 0.5663413799007021,
0.5593100496490326, 0.5765057353192946, 0.5835593220338983, 0.579085772984078, 0.5828060263653483,
0.5751926040061632, 0.5827212806026365, 0.5900992980653997, 0.5944213319637047, 0.588482280431433,
0.5879472693032014, 0.595789248416367, 0.5920475945899675, 0.588748501968841, 0.5850008560178052,
0.5861273754494094]
========================================
Optimal K-Value:  31
========================================
AUC value of test data:  0.5574258422528571
========================================
Confusion Matrix :
============================================================
              Predicted: NO  Predicted: YES
Actual: NO                0              457
Actual: YES               0             2543
============================================================================================================
=======
============================================================================================================
=======
============================================================================================================
=======
```








```
Results of analysis using Tf-Idf Weighted Word2Vec vectorized text features merged with other feat
ures using K-NN brute force algorithm:
========================================================================
AUC values of cross-validate data:
========================================
[0.5054699537750386, 0.5398151001540832, 0.5457644239000171, 0.5642809450436568,
0.5613199794555727, 0.5715399760315014, 0.5773549049820237, 0.5734565998972778,
0.5695505906522854, 0.5757712720424585, 0.5779061804485532, 0.5786218113336757,
0.5837502140044513, 0.5892723848656053, 0.5903329909262114, 0.5891379900701935, 0.595733607259031,
0.5950522170861154, 0.5976254066084574, 0.599316897791474]
========================================
Optimal K-Value:  39
========================================
AUC value of test data:  0.5771384269341937
========================================
Confusion Matrix :
============================================================
             Predicted: NO  Predicted: YES
Actual: NO               0             457
Actual: YES              0            2543
============================================================================================================
=======
============================================================================================================
=======
============================================================================================================
=======
```

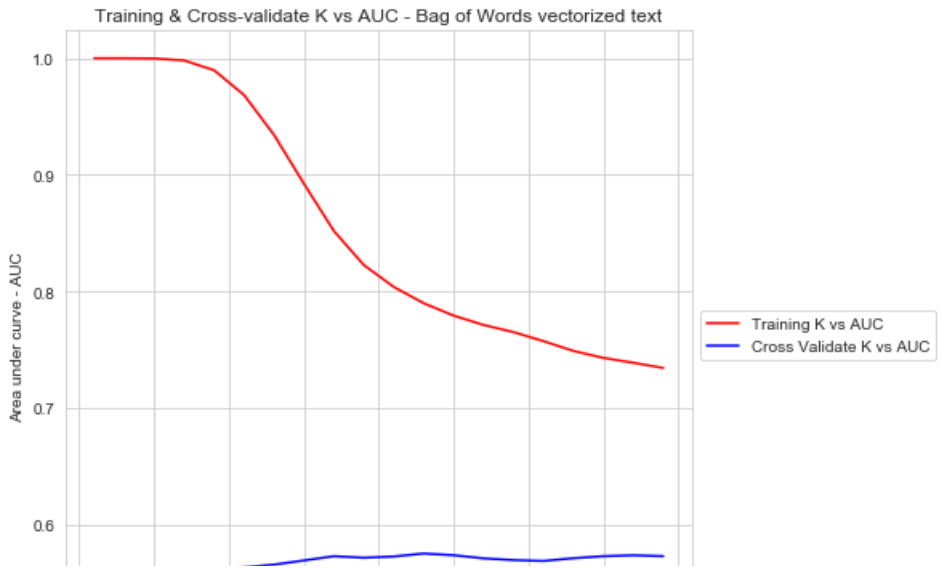
Analysis on Balanced Dataset Using K-NN(Simple Cross Validation)

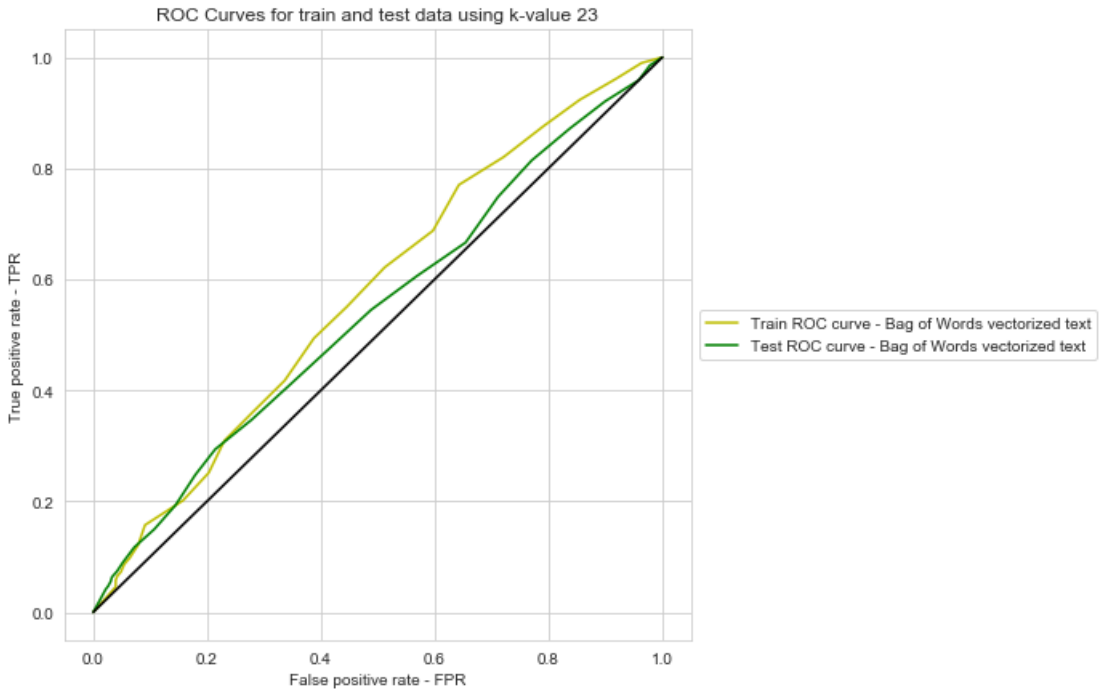
In [181]:

```
testKValues = np.arange(1, 40, 2);
```

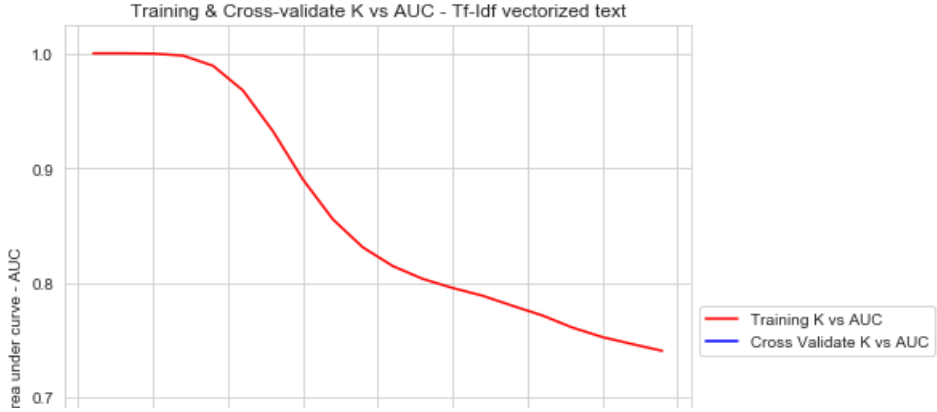
```
techniques = ['Bag of Words', 'Tf-Idf', 'Average Word2Vec', 'Tf-Idf Weighted Word2Vec'];
for index, technique in enumerate(techniques):
    areaUnderRocValuesTrain = [];
    areaUnderRocValuesCrossValidate = [];
    trainingMergedData = hstack((categoriesVectorsSub,\
                                 subCategoriesVectorsSub,\
                                 teacherPrefixVectorsSub,\
                                 schoolStateVectorsSub,\
                                 projectGradeVectorsSub,\
                                 priceStandardizedSub,\
                                 previouslyPostedStandardizedSub));
    crossValidateMergedData = hstack((categoriesTransformedCrossValidateData,\
                                      subCategoriesTransformedCrossValidateData,\
                                      teacherPrefixTransformedCrossValidateData,\
                                      schoolStateTransformedCrossValidateData,\
                                      projectGradeTransformedCrossValidateData,\
                                      priceTransformedCrossValidateData,\
                                      previouslyPostedTransformedCrossValidateData));
    testMergedData = hstack((categoriesTransformedTestData,\
                                      subCategoriesTransformedTestData,\
                                      teacherPrefixTransformedTestData,\
                                      schoolStateTransformedTestData,\
                                      projectGradeTransformedTestData,\
                                      priceTransformedTestData,\
                                      previouslyPostedTransformedTestData));

    if(index == 0):
        trainingMergedData = hstack((trainingMergedData,\
                                     bowTitleModelSub,\
                                     bowEssayModelSub));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                          bowTitleTransformedCrossValidateData,\
                                          bowEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                 bowTitleTransformedTestData,\
                                 bowEssayTransformedTestData));
    elif(index == 1):
        trainingMergedData = hstack((trainingMergedData,\
                                     tfIdfTitleModelSub,\
                                     tfIdfEssayModelSub));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                          tfIdfTitleTransformedCrossValidateData,\
                                          tfIdfEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                 tfIdfTitleTransformedTestData,\
                                 tfIdfEssayTransformedTestData));
    elif(index == 2):
        trainingMergedData = hstack((trainingMergedData,\
                                     word2VecTitlesVectorsSub,\
                                     word2VecEssaysVectorsSub));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                          word2VecTitleTransformedCrossValidateData,\
                                          word2VecEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                 word2VecTitleTransformedTestData,\
                                 word2VecEssayTransformedTestData));
    elif(index == 3):
        trainingMergedData = hstack((trainingMergedData,\
                                     tfIdfWeightedWord2VecTitlesVectorsSub,\
                                     tfIdfWeightedWord2VecEssaysVectorsSub));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                          tfIdfWeightedTitleTransformedCrossValidateData,\
                                          tfIdfWeightedEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                 tfIdfWeightedTitleTransformedTestData,\
                                 tfIdfWeightedEssayTransformedTestData));
    for testKValue in tqdm(testKValues):
        knnClassifier = KNeighborsClassifier(n_neighbors = testKValue, algorithm = 'brute');
        knnClassifier.fit(trainingMergedData, classesTraining);
        predProbScores = knnClassifier.predict_proba(trainingMergedData);
        fpr, tpr, threshold = roc_curve(classesTraining, predProbScores[:, 1]);
        areaUnderRocValuesTrain.append(auc(fpr, tpr));
        predProbScores = knnClassifier.predict_proba(crossValidateMergedData);
        fpr, tpr, threshold = roc_curve(classesCrossValidate, predProbScores[:, 1]);
        areaUnderRocValuesCrossValidate.append(auc(fpr, tpr));

    plt.plot(testKValues, areaUnderRocValuesTrain, 'r', label = "Training K vs AUC");
```

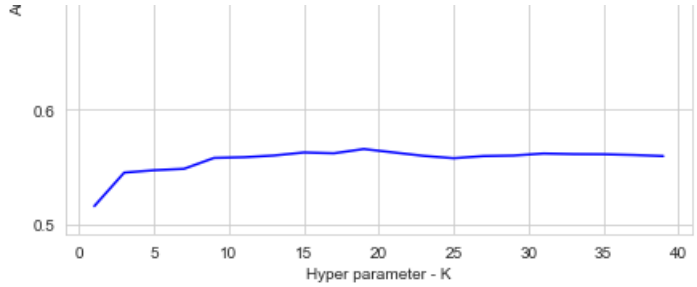
```
    plt.plot(testKValues, areaUnderRocValuesCrossValidate, 'b', label = "Cross Validate K vs AUC");
    plt.title("Training & Cross-validate K vs AUC - {} vectorized text".format(technique));
    plt.xlabel("Hyper parameter - K");
    plt.ylabel("Area under curve - AUC");
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.show();

    optimalKValue = testKValues[np.argmax(areaUnderRocValuesCrossValidate)];
    knnClassifier = KNeighborsClassifier(n_neighbors = optimalKValue, algorithm = 'brute');
    knnClassifier.fit(trainingMergedData, classesTraining);
    predProbScoresCrossValidate = knnClassifier.predict_proba(crossValidateMergedData);
    fprCrossValidate, tprCrossValidate, thresholdTrain = roc_curve(classesCrossValidate, predProbSc
oresCrossValidate[:, 1]);
    predProbScoresTest = knnClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predProbScoresTest[:, 1]);
    areaUnderRocValueTest = auc(fprTest, tprTest);
    plt.plot(fprCrossValidate, tprCrossValidate, 'y', label="Train ROC curve - {} vectorized text".
format(technique));
    plt.plot(fprTest, tprTest, 'g', label="Test ROC curve - {} vectorized text".format(technique));
    plt.plot([0, 1], [0, 1], 'k-');
    plt.title("ROC Curves for train and test data using k-value {}".format(optimalKValue))
    plt.xlabel('False positive rate - FPR');
    plt.ylabel('True positive rate - TPR');
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.show();

    print("Results of analysis using {} vectorized text features merged with other features using
K-NN brute force algorithm:".format(technique));
    equalsBorder(70);
    print("AUC values of cross-validate data: ");
    equalsBorder(40);
    print(areaUnderRocValuesCrossValidate);
    equalsBorder(40);
    print("Optimal K-Value: ", optimalKValue);
    equalsBorder(40);
    print("AUC value of test data: ", areaUnderRocValueTest);
    # Predicting classes of test data projects
    predictionClassesTest = knnClassifier.predict(testMergedData);
    equalsBorder(40);
    # Adding results to results dataframe
    balancedDataResultsDataFrame = balancedDataResultsDataFrame.append({'Vectorizer': technique,
'Model': 'Brute', 'Hyper Parameter - K': optimalKValue, 'AUC': areaUnderRocValueTest},
ignore_index = True);
    # Printing confusion matrix
    confusionMatrix = confusion_matrix(classesTest, predictionClassesTest);
    # Creating dataframe for generated confusion matrix
    confusionMatrixDataFrame = pd.DataFrame(data = confusionMatrix, index = ['Actual: NO', 'Actual:
YES'], columns = ['Predicted: NO', 'Predicted: YES']);
    print("Confusion Matrix : ");
    equalsBorder(60);
    print(confusionMatrixDataFrame);
    equalsBorder(110);
    equalsBorder(110);
    equalsBorder(110);
```

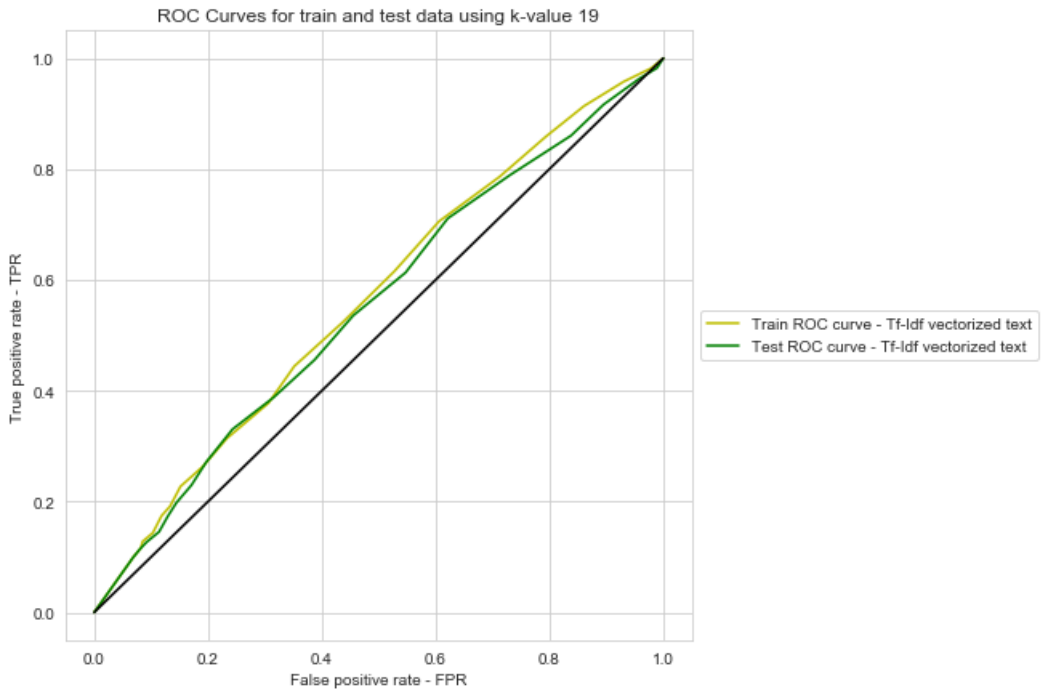




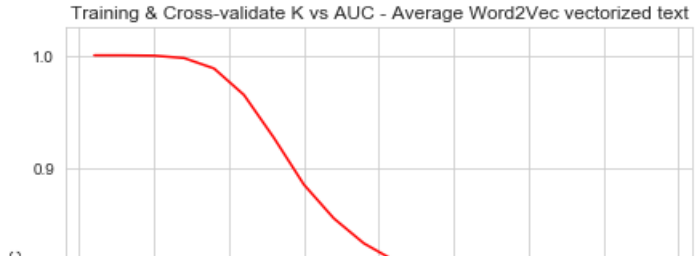

```
Results of analysis using Bag of Words vectorized text features merged with other features using K
-NN brute force algorithm:
========================================================================
AUC values of cross-validate data:
========================================
[0.524884437596302, 0.5517223078240027, 0.5509604519774011, 0.5572376305427154,
0.5616863550761856, 0.5633008046567369, 0.5657609998287965, 0.5693374422187981, 0.57299007019346,
0.5716675226844719, 0.5727409690121555, 0.5752884780003424, 0.5738298236603322,
0.5711427837699025, 0.5696456086286595, 0.5689085772984077, 0.5712746105118987,
0.5729977743537065, 0.5737579181646978, 0.5729969183359013]
========================================
Optimal K-Value:  23
========================================
AUC value of test data:  0.5423327089164833
========================================
Confusion Matrix :
============================================================
             Predicted: NO  Predicted: YES
Actual: NO             330             127
Actual: YES           1662             881
================================================================================================================
=======
================================================================================================================
=======
================================================================================================================
=======
```

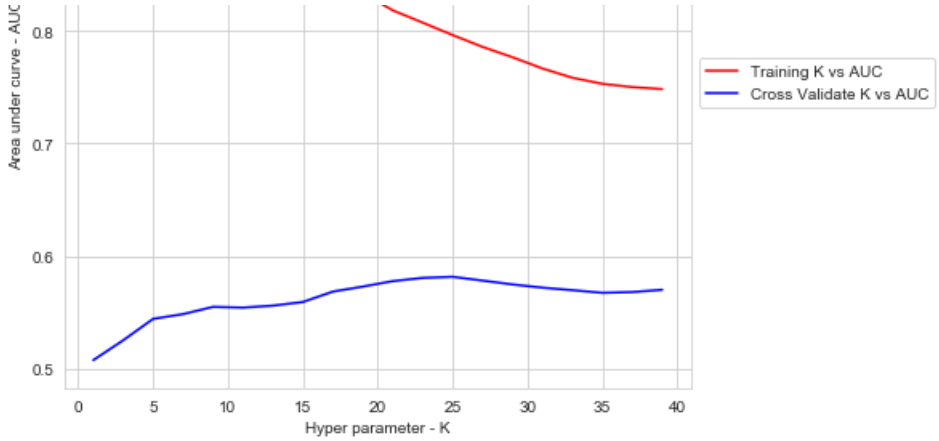


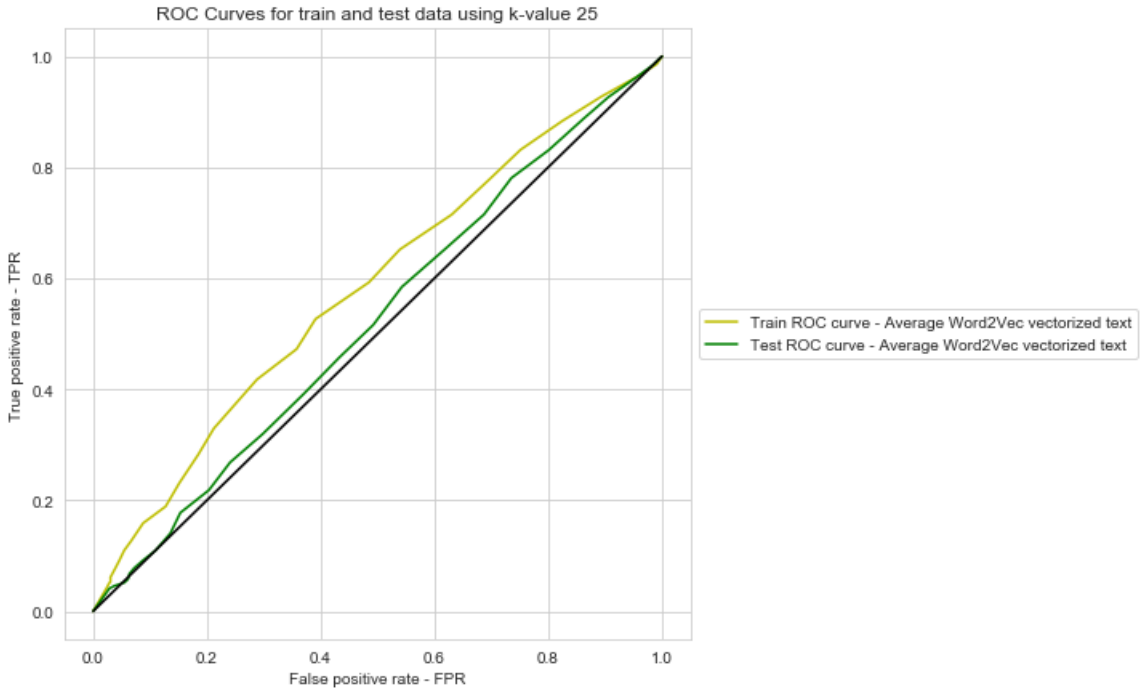




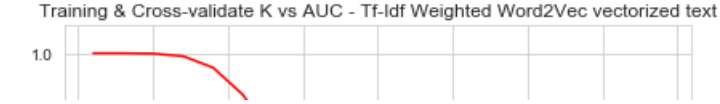

```
Results of analysis using Tf-Idf vectorized text features merged with other features using K-NN br
ute force algorithm:
========================================================================
AUC values of cross-validate data:
========================================
[0.516050333846944, 0.5453372710152371, 0.5473942818010614, 0.5486611881527135,
0.5581381612737545, 0.5587442218798151, 0.5602473891456943, 0.5628822119500085,
0.5621751412429379, 0.565894538606403, 0.5629078924841636, 0.5599272384865606, 0.5578813559322034,
0.5597551789077213, 0.5602114363978771, 0.5619936654682417, 0.5614680705358671,
0.5613439479541175, 0.5607002225646294, 0.5597508988186953]
========================================
Optimal K-Value:  19
========================================
AUC value of test data:  0.5536750387858377
========================================
Confusion Matrix :
============================================================
             Predicted: NO  Predicted: YES
Actual: NO             313             144
Actual: YES           1557             986
============================================================================================================
=======
============================================================================================================
=======
============================================================================================================
=======
```

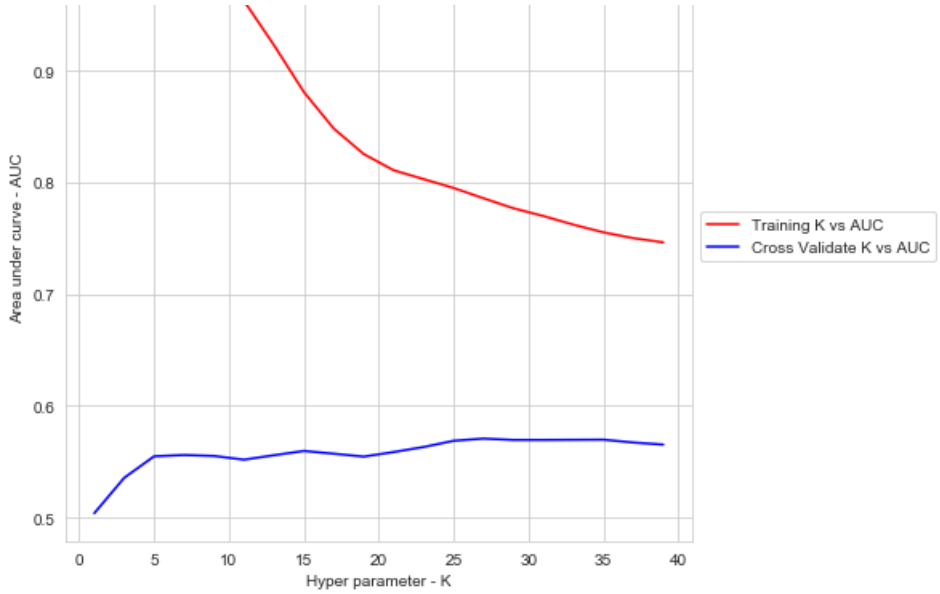


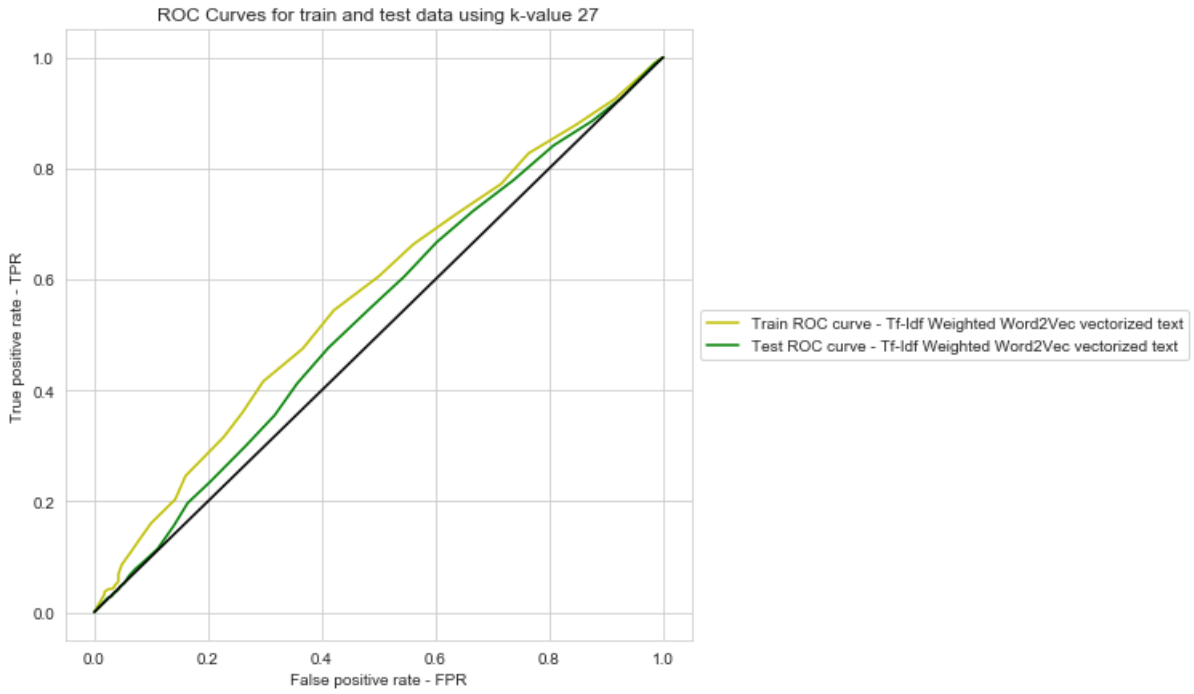





```
Results of analysis using Average Word2Vec vectorized text features merged with other features usi
ng K-NN brute force algorithm:
======================================================================
AUC values of cross-validate data:
========================================
[0.5076014381099128, 0.5252345488786168, 0.5442732408834103, 0.5484737202533814,
0.554909262112652, 0.5542432802602294, 0.5559921246361924, 0.5592158876904639, 0.5684463276836158,
0.5728950522170861, 0.5777469611367916, 0.5806171888375278, 0.5815939051532272,
0.5782306111967129, 0.5747988358157848, 0.5719577127204246, 0.5697183701420989, 0.567366889231296,
0.5680414312617702, 0.5700744735490498]
========================================
Optimal K-Value:  25
========================================
AUC value of test data:  0.5226962761293498
========================================
Confusion Matrix :
================================================================
             Predicted: NO  Predicted: YES
Actual: NO             321             136
Actual: YES           1732             811
================================================================================================================
=======
================================================================================================================
=======
================================================================================================================
=======
```








```
Results of analysis using Tf-Idf Weighted Word2Vec vectorized text features merged with other feat
ures using K-NN brute force algorithm:
========================================================================
AUC values of cross-validate data:
========================================
[0.5036723163841808, 0.5354357130628317, 0.5547320664269817, 0.5558859784283513,
0.5550273925697655, 0.5517068995035097, 0.5556574216743708, 0.5595360383495976,
0.5570441705187468, 0.554457284711522, 0.5585593220338982, 0.5630362951549392, 0.568613251155624,
0.5705932203389831, 0.5693400102722137, 0.5693562746105119, 0.5694881013525082,
0.5696216401301148, 0.5670672829994863, 0.565216572504708]
========================================
Optimal K-Value:  27
========================================
AUC value of test data:  0.5351765820448461
========================================
Confusion Matrix :
============================================================
             Predicted: NO  Predicted: YES
Actual: NO             335             122
Actual: YES           1779             764
============================================================================================================
=======
============================================================================================================
=======
============================================================================================================
=======
```

Classification of data using K-NN(k-fold cross validation)

Splitting Data(Only training and test)

In [224]:

```
projectsData = projectsData.dropna(subset = ['teacher_prefix']);
projectsData.shape
```

Out[224]:

```
(109245, 22)
```

In [225]:

```
classesData = projectsData['project_is_approved']
print(classesData.shape)
```

```
(109245,)
```

In [226]:

```
trainingData, testData, classesTraining, classesTest = cross_validation.train_test_split(projectsDa
ta[0:10000], classesData[0:10000], test_size =  0.3, random_state = 0);
```

In [227]:

```
print("Shapes of splitted data: ");
equalsBorder(70);

print("testData shape: ", testData.shape);
print("classesTest: ", classesTest.shape);
print("trainingData shape: ", trainingData.shape);
print("classesTraining shape: ", classesTraining.shape);
```

```
Shapes of splitted data:
======================================================================
testData shape:  (3000, 22)
classesTest:  (3000,)
trainingData shape:  (7000, 22)
classesTraining shape:  (7000,)
```

In [229]:

```
print("Number of negative points: ", trainingData[trainingData['project_is_approved'] == 0].shape)
;
print("Number of positive points: ", trainingData[trainingData['project_is_approved'] == 1].shape)
;
```

```
Number of negative points:  (1043, 22)
Number of positive points:  (5957, 22)
```

Balancing Data

Note: Instead of displaying whole vectorization process for balanced and imbalanced data, we have simply disabled below cell while performing analysis on imbalanced data and enabled while performing analysis on balanced data

In [131]:

```
negativeData = trainingData[trainingData['project_is_approved'] == 0];
positiveData = trainingData[trainingData['project_is_approved'] == 1];
negativeDataBalanced = resample(negativeData, replace = True, n_samples = 5957, random_state = 44);
trainingData = pd.concat([positiveData, negativeDataBalanced]);
```

```
trainingData = pd.concat([positiveData, negativeDataBalanced]);
trainingData = shuffle(trainingData);
classesTraining = trainingData['project_is_approved'];
print("Testing whether data is balanced: ");
equalsBorder(60);
print("Number of positive points: ", trainingData[trainingData['project_is_approved'] == 1].shape)
;
print("Number of negative points: ", trainingData[trainingData['project_is_approved'] == 0].shape)
;
```

```
Testing whether data is balanced:
============================================================
Number of positive points:  (5957, 22)
Number of negative points:  (5957, 22)
```

Vectorizing categorical data

1. Vectorizing cleaned_categories(project_subject_categories cleaned) - One Hot Encoding

In [230]:

```
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e cleaned_categories
subjectsCategoriesVectorizer = CountVectorizer(vocabulary = list(sortedCategoriesDictionary.keys()
), lowercase = False, binary = True);
# Fitting CountVectorizer with cleaned_categories values
subjectsCategoriesVectorizer.fit(trainingData['cleaned_categories'].values);
# Vectorizing categories using one-hot-encoding
categoriesVectors = subjectsCategoriesVectorizer.transform(trainingData['cleaned_categories'].valu
es);
```

In [231]:

```
print("Features used in vectorizing categories: ");
equalsBorder(70);
print(subjectsCategoriesVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of cleaned_categories matrix after vectorization(one-hot-encoding): ",
categoriesVectors.shape);
equalsBorder(70);
print("Sample vectors of categories: ");
equalsBorder(70);
print(categoriesVectors[0:4])
```

```
Features used in vectorizing categories:
======================================================================
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
======================================================================
Shape of cleaned_categories matrix after vectorization(one-hot-encoding):  (7000, 9)
======================================================================
Sample vectors of categories:
======================================================================
  (0, 7) 1
  (0, 8) 1
  (1, 8) 1
  (2, 7) 1
  (2, 8) 1
  (3, 7) 1
```

2. Vectorizing cleaned_sub_categories(project_subject_sub_categories cleaned) - One Hot Encoding

In [232]:

```
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e cleaned_sub_categories
subjectsSubCategoriesVectorizer = CountVectorizer(vocabulary = list(sortedDictionarySubCategories.
```

```
keys()), lowercase = False, binary = True);
# Fitting CountVectorizer with cleaned_sub_categories values
subjectsSubCategoriesVectorizer.fit(trainingData['cleaned_sub_categories'].values);
# Vectorizing sub categories using one-hot-encoding
subCategoriesVectors =
subjectsSubCategoriesVectorizer.transform(trainingData['cleaned_sub_categories'].values);
```

In [233]:

```
print("Features used in vectorizing subject sub categories: ");
equalsBorder(70);
print(subjectsSubCategoriesVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of cleaned_categories matrix after vectorization(one-hot-encoding): ",
subCategoriesVectors.shape);
equalsBorder(70);
print("Sample vectors of categories: ");
equalsBorder(70);
print(subCategoriesVectors[0:4])
```

```
Features used in vectorizing subject sub 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',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
======================================================================
Shape of cleaned_categories matrix after vectorization(one-hot-encoding):  (7000, 30)
======================================================================
Sample vectors of categories:
======================================================================
  (0, 27) 1
  (0, 28) 1
  (1, 29) 1
  (2, 27) 1
  (2, 28) 1
  (3, 22) 1
  (3, 25) 1
```

3. Vectorizing teacher_prefix - One Hot Encoding

In [234]:

```
def giveCounter(data):
    counter = Counter();
    for dataValue in data:
        counter.update(str(dataValue).split());
    return counter
```

In [235]:

```
giveCounter(trainingData['teacher_prefix'].values)
```

Out[235]:

```
Counter({'Mrs.': 3577, 'Ms.': 2569, 'Mr.': 705, 'Teacher': 149})
```

In [236]:

```
teacherPrefixDictionary = dict(giveCounter(trainingData['teacher_prefix'].values));
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e teacher_prefix
teacherPrefixVectorizer = CountVectorizer(vocabulary = list(teacherPrefixDictionary.keys()),
lowercase = False, binary = True);
# Fitting CountVectorizer with teacher_prefix values
teacherPrefixVectorizer.fit(trainingData['teacher_prefix'].values);
# Vectorizing teacher_prefix using one-hot-encoding
teacherPrefixVectors = teacherPrefixVectorizer.transform(trainingData['teacher_prefix'].values);
```

In [237]:

```
print("Features used in vectorizing teacher_prefix: ");
equalsBorder(70);
print(teacherPrefixVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of teacher_prefix matrix after vectorization(one-hot-encoding): ",
teacherPrefixVectors.shape);
equalsBorder(70);
print("Sample vectors of teacher_prefix: ");
equalsBorder(70);
print(teacherPrefixVectors[0:100]);
```

```
Features used in vectorizing teacher_prefix:
======================================================================
['Mrs.', 'Ms.', 'Mr.', 'Teacher']
======================================================================
Shape of teacher_prefix matrix after vectorization(one-hot-encoding):  (7000, 4)
======================================================================
Sample vectors of teacher_prefix:
======================================================================
  (44, 3) 1
  (50, 3) 1
```

In [238]:

```
teacherPrefixes = [prefix.replace('.', '') for prefix in trainingData['teacher_prefix'].values];
teacherPrefixes[0:5]
```

Out[238]:

```
['Mrs', 'Ms', 'Mrs', 'Ms', 'Ms']
```

In [239]:

```
trainingData['teacher_prefix'] = teacherPrefixes;
trainingData.head(3)
```

Out[239]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime
7681	35229	p237183	4d9242a296d9d801de0a3ba48524a6a9	Mrs	CO	2017-01-09 17:00:24
9032	42897	p232226	c2f342e359635429a948c3d02f28a2d0	Ms	MO	2017-01-31 09:45:47
3691	47711	p180884	73872f258c29d357a925f7518169b019	Mrs	TX	2016-08-12 15:20:19

3 rows × 22 columns

In [240]:

```
teacherPrefixDictionary = dict(giveCounter(trainingData['teacher_prefix'].values));
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e teacher_prefix
teacherPrefixVectorizer = CountVectorizer(vocabulary = list(teacherPrefixDictionary.keys()),
lowercase = False, binary = True);
# Fitting CountVectorizer with teacher_prefix values
teacherPrefixVectorizer.fit(trainingData['teacher_prefix'].values);
# Vectorizing teacher_prefix using one-hot-encoding
```

```
teacherPrefixVectors = teacherPrefixVectorizer.transform(trainingData['teacher_prefix'].values);
```

In [241]:

```
print("Features used in vectorizing teacher_prefix: ");
equalsBorder(70);
print(teacherPrefixVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of teacher_prefix matrix after vectorization(one-hot-encoding): ",
teacherPrefixVectors.shape);
equalsBorder(70);
print("Sample vectors of teacher_prefix: ");
equalsBorder(70);
print(teacherPrefixVectors[0:4]);
```

```
Features used in vectorizing teacher_prefix:
======================================================================
['Mrs', 'Ms', 'Mr', 'Teacher']
======================================================================
Shape of teacher_prefix matrix after vectorization(one-hot-encoding):  (7000, 4)
======================================================================
Sample vectors of teacher_prefix:
======================================================================
  (0, 0) 1
  (1, 1) 1
  (2, 0) 1
  (3, 1) 1
```

4. Vectorizing school_state - One Hot Encoding

In [242]:

```
schoolStateDictionary = dict(giveCounter(trainingData['school_state'].values));
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e school states
schoolStateVectorizer = CountVectorizer(vocabulary = list(schoolStateDictionary.keys()), lowercase
= False, binary = True);
# Fitting CountVectorizer with school_state values
schoolStateVectorizer.fit(trainingData['school_state'].values);
# Vectorizing school_state using one-hot-encoding
schoolStateVectors = schoolStateVectorizer.transform(trainingData['school_state'].values);
```

In [243]:

```
print("Features used in vectorizing school_state: ");
equalsBorder(70);
print(schoolStateVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of school_state matrix after vectorization(one-hot-encoding): ", schoolStateVectors.s
hape);
equalsBorder(70);
print("Sample vectors of school_state: ");
equalsBorder(70);
print(schoolStateVectors[0:4]);
```

```
Features used in vectorizing school_state:
======================================================================
['CO', 'MO', 'TX', 'UT', 'WI', 'OH', 'SC', 'NY', 'IL', 'AZ', 'KS', 'ID', 'TN', 'MS', 'CA', 'MI', 'K
Y', 'GA', 'FL', 'VA', 'CT', 'NC', 'NH', 'RI', 'MA', 'IN', 'NJ', 'WV', 'MD', 'MT', 'DE', 'LA', 'OK',
'PA', 'MN', 'AL', 'NE', 'SD', 'OR', 'WA', 'HI', 'NV', 'AR', 'IA', 'WY', 'NM', 'AK', 'ME', 'DC', 'VT
', 'ND']
======================================================================
Shape of school_state matrix after vectorization(one-hot-encoding):  (7000, 51)
======================================================================
Sample vectors of school_state:
======================================================================
  (0, 0) 1
  (1, 1) 1
  (2, 2) 1
  (3, 3) 1
```

5. Vectorizing project_grade_category - One Hot Encoding

In [244]:

```
giveCounter(trainingData['project_grade_category'])
```

Out[244]:

```
Counter({'Grades': 7000,
         'PreK-2': 2835,
         '3-5': 2394,
         '6-8': 1083,
         '9-12': 688})
```

In [245]:

```
cleanedGrades = []
for grade in trainingData['project_grade_category'].values:
    grade = grade.replace(' ', '');
    grade = grade.replace('-', 'to');
    cleanedGrades.append(grade);
cleanedGrades[0:4]
```

Out[245]:

```
['GradesPreKto2', 'Grades3to5', 'Grades3to5', 'Grades6to8']
```

In [246]:

```
trainingData['project_grade_category'] = cleanedGrades
trainingData.head(4)
```

Out[246]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime
7681	35229	p237183	4d9242a296d9d801de0a3ba48524a6a9	Mrs	CO	2017-01-09 17:00:24
9032	42897	p232226	c2f342e359635429a948c3d02f28a2d0	Ms	MO	2017-01-31 09:45:47
3691	47711	p180884	73872f258c29d357a925f7518169b019	Mrs	TX	2016-08-12 15:20:19
202	83193	p151416	833e60834433e807a89b9720b7bc1925	Ms	UT	2017-01-17 23:58:25

4 rows × 22 columns

In [247]:

```
projectGradeDictionary = dict(giveCounter(trainingData['project_grade_category'].values));
# Using CountVectorizer for performing one-hot-encoding by setting vocabulary as list of all uniqu
e project grade categories
projectGradeVectorizer = CountVectorizer(vocabulary = list(projectGradeDictionary.keys()),
lowercase = False, binary = True);
# Fitting CountVectorizer with project_grade_category values
projectGradeVectorizer.fit(trainingData['project_grade_category'].values);
# Vectorizing project_grade_category using one-hot-encoding
projectGradeVectors =
projectGradeVectorizer.transform(trainingData['project_grade_category'].values);
```

In [248]:

```
print("Features used in vectorizing project_grade_category: ");
equalsBorder(70);
print(projectGradeVectorizer.get_feature_names());
equalsBorder(70);
print("Shape of school_state matrix after vectorization(one-hot-encoding): ", projectGradeVectors.
shape);
equalsBorder(70);
print("Sample vectors of school_state: ");
equalsBorder(70);
print(projectGradeVectors[0:4]);
```

```
Features used in vectorizing project_grade_category:
======================================================================
['GradesPreKto2', 'Grades3to5', 'Grades6to8', 'Grades9to12']
======================================================================
Shape of school_state matrix after vectorization(one-hot-encoding):  (7000, 4)
======================================================================
Sample vectors of school_state:
======================================================================
  (0, 0) 1
  (1, 1) 1
  (2, 1) 1
  (3, 2) 1
```

In [249]:

```
preProcessedEssaysWithStopWords, preProcessedEssaysWithoutStopWords =
preProcessingWithAndWithoutStopWords(trainingData['project_essay']);
preProcessedProjectTitlesWithStopWords, preProcessedProjectTitlesWithoutStopWords =
preProcessingWithAndWithoutStopWords(trainingData['project_title']);
```

Vectorizing Text Data

Bag of Words

1. Vectorizing project_essay

In [250]:

```
# Initializing countvectorizer for bag of words vectorization of preprocessed project essays
bowEssayVectorizer = CountVectorizer(min_df = 10);
# Transforming the preprocessed essays to bag of words vectors
bowEssayModel = bowEssayVectorizer.fit_transform(preProcessedEssaysWithoutStopWords);
```

In [251]:

```
print("Some of the Features used in vectorizing preprocessed essays: ");
equalsBorder(70);
print(bowEssayVectorizer.get_feature_names()[-40:]);
equalsBorder(70);
print("Shape of preprocessed essay matrix after vectorization: ", bowEssayModel.shape);
equalsBorder(70);
print("Sample bag-of-words vector of preprocessed essay: ");
equalsBorder(70);
print(bowEssayModel[0])
```

```
Some of the Features used in vectorizing preprocessed essays:
======================================================================
['worn', 'worried', 'worry', 'worrying', 'worst', 'worth', 'worthwhile', 'worthy', 'would', 'wow',
'wrap', 'write', 'writer', 'writers', 'writing', 'writings', 'written', 'wrong', 'wrote', 'yard',
'year', 'yearbook', 'yearly', 'yearn', 'yearning', 'years', 'yes', 'yesterday', 'yet', 'yoga', 'yo
rk', 'young', 'younger', 'youngest', 'youth', 'youtube', 'zip', 'zone', 'zones', 'zoo']
```

```
['tx', 'young', 'younger', 'youngest', 'youth', 'youtube', 'zip', 'zone', 'zones', 'zoo']
======================================================================
Shape of preprocessed essay matrix after vectorization:  (7000, 5121)
======================================================================
Sample bag-of-words vector of preprocessed essay:
======================================================================
  (0, 3084) 1
  (0, 2888) 1
  (0, 3749) 1
  (0, 3890) 1
  (0, 4893) 1
  (0, 582) 1
  (0, 815) 1
  (0, 4869) 1
  (0, 766) 1
  (0, 800) 1
  (0, 2280) 1
  (0, 2262) 1
  (0, 3226) 1
  (0, 2864) 1
  (0, 3241) 1
  (0, 4698) 1
  (0, 3215) 1
  (0, 968) 2
  (0, 5062) 2
  (0, 4798) 1
  (0, 4450) 1
  (0, 4415) 1
  (0, 1687) 1
  (0, 2689) 1
  (0, 338) 1
  : :
  (0, 2063) 1
  (0, 4090) 2
  (0, 3349) 1
  (0, 5106) 1
  (0, 1678) 1
  (0, 4343) 1
  (0, 2078) 1
  (0, 3686) 1
  (0, 3760) 1
  (0, 2318) 1
  (0, 5101) 1
  (0, 3129) 1
  (0, 2600) 1
  (0, 4141) 1
  (0, 3552) 1
  (0, 4203) 2
  (0, 2090) 1
  (0, 4048) 5
  (0, 3164) 2
  (0, 442) 1
  (0, 1371) 1
  (0, 912) 1
  (0, 33) 1
  (0, 1770) 1
  (0, 4477) 5
```

2. Vectorizing project_title

In [252]:

```
# Initializing countvectorizer for bag of words vectorization of preprocessed project titles
bowTitleVectorizer = CountVectorizer(min_df = 10);
# Transforming the preprocessed project titles to bag of words vectors
bowTitleModel = bowTitleVectorizer.fit_transform(preProcessedProjectTitlesWithoutStopWords);
```

In [253]:

```
print("Some of the Features used in vectorizing preprocessed titles: ");
equalsBorder(70);
print(bowTitleVectorizer.get_feature_names()[-40:]);
equalsBorder(70);
print("Shape of preprocessed title matrix after vectorization: ", bowTitleModel.shape);
```

```
equalsBorder(70);
print("Sample bag-of-words vector of preprocessed title: ");
equalsBorder(70);
print(bowTitleModel[0])
```

```
Some of the Features used in vectorizing preprocessed titles:
======================================================================
['teach', 'teacher', 'teaching', 'team', 'tech', 'technology', 'texts', 'think', 'thinking', 'thir
d', 'time', 'today', 'together', 'tomorrow', 'tools', 'us', 'use', 'using', 'virtual', 'visual', '
want', 'wanted', 'way', 'wiggle', 'wiggles', 'wiggly', 'without', 'wobble', 'wobbling', 'wonder',
'words', 'work', 'working', 'world', 'write', 'writers', 'writing', 'year', 'yoga', 'young']
======================================================================
Shape of preprocessed title matrix after vectorization:  (7000, 450)
======================================================================
Sample bag-of-words vector of preprocessed title:
======================================================================
  (0, 179) 1
  (0, 154) 1
  (0, 414) 1
```

Tf-Idf Vectorization

1. Vectorizing project_essay

In [254]:

```
# Intializing tfidf vectorizer for tf-idf vectorization of preprocessed project essays
tfIdfEssayVectorizer = TfidfVectorizer(min_df = 10);
# Transforming the preprocessed project essays to tf-idf vectors
tfIdfEssayModel = tfIdfEssayVectorizer.fit_transform(preProcessedEssaysWithoutStopWords);
```

In [255]:

```
print("Some of the Features used in tf-idf vectorizing preprocessed essays: ");
equalsBorder(70);
print(tfIdfEssayVectorizer.get_feature_names()[-40:]);
equalsBorder(70);
print("Shape of preprocessed title matrix after tf-idf vectorization: ", tfIdfEssayModel.shape);
equalsBorder(70);
print("Sample Tf-Idf vector of preprocessed essay: ");
equalsBorder(70);
print(tfIdfEssayModel[0])
```

```
Some of the Features used in tf-idf vectorizing preprocessed essays:
======================================================================
['worn', 'worried', 'worry', 'worrying', 'worst', 'worth', 'worthwhile', 'worthy', 'would', 'wow',
'wrap', 'write', 'writer', 'writers', 'writing', 'writings', 'written', 'wrong', 'wrote', 'yard',
'year', 'yearbook', 'yearly', 'yearn', 'yearning', 'years', 'yes', 'yesterday', 'yet', 'yoga', 'yo
rk', 'young', 'younger', 'youngest', 'youth', 'youtube', 'zip', 'zone', 'zones', 'zoo']
======================================================================
Shape of preprocessed title matrix after tf-idf vectorization:  (7000, 5121)
======================================================================
Sample Tf-Idf vector of preprocessed essay:
======================================================================
  (0, 4477) 0.12700485794721247
  (0, 1770) 0.1501479429060923
  (0, 33) 0.14547080375188362
  (0, 912) 0.04369473310467847
  (0, 1371) 0.06559461630695812
  (0, 442) 0.06977084317612585
  (0, 3164) 0.07345381335528468
  (0, 4048) 0.14612875624054827
  (0, 2090) 0.14152773602400653
  (0, 4203) 0.192323071754554
  (0, 3552) 0.12557535513714596
  (0, 4141) 0.08799952468439892
  (0, 2600) 0.12130869322736769
  (0, 3129) 0.05121889307710767
  (0, 5101) 0.05033444847688412
  (0, 2318) 0.1816977855757463
  (0, 3760) 0.08817441571779058
  (0, 3686) 0.15791226835037236
```

```
  (0, 2078) 0.13071533496108556
  (0, 4343) 0.18557994829788635
  (0, 1678) 0.09278359574194336
  (0, 5106) 0.07522207983036418
  (0, 3349) 0.1703148983882007
  (0, 4090) 0.24461904382817837
  (0, 2063) 0.05426874568802244
  : :
  (0, 338) 0.08782583980565964
  (0, 2689) 0.034263064673674785
  (0, 1687) 0.11911738064587442
  (0, 4415) 0.08375890583944867
  (0, 4450) 0.12656309771933918
  (0, 4798) 0.09842964636327795
  (0, 5062) 0.08670396832130248
  (0, 968) 0.1777727683899534
  (0, 3215) 0.074294949088735O7
  (0, 4698) 0.09032587355509344
  (0, 3241) 0.0629788116565148
  (0, 2864) 0.03978365071513317
  (0, 3226) 0.05153439950124092
  (0, 2262) 0.06158238143507397
  (0, 2280) 0.0875675379762204
  (0, 800) 0.05997134319619762
  (0, 766) 0.09160005462314318
  (0, 4869) 0.045806377850231З4
  (0, 815) 0.14010853197114678
  (0, 582) 0.0629788116565148
  (0, 4893) 0.08390661422532483
  (0, 3890) 0.07962552699337769
  (0, 3749) 0.05246281048116416
  (0, 2888) 0.06267412793749919
  (0, 3084) 0.02627215225182836
```

2. Vectorizing project_title

In [256]:

```
# Intializing tfidf vectorizer for tf-idf vectorization of preprocessed project titles
tfIdfTitleVectorizer = TfidfVectorizer(min_df = 10);
# Transforming the preprocessed project titles to tf-idf vectors
tfIdfTitleModel = tfIdfTitleVectorizer.fit_transform(preProcessedProjectTitlesWithoutStopWords);
```

In [257]:

```
print("Some of the Features used in tf-idf vectorizing preprocessed titles: ");
equalsBorder(70);
print(tfIdfTitleVectorizer.get_feature_names()[-40:]);
equalsBorder(70);
print("Shape of preprocessed title matrix after tf-idf vectorization: ", tfIdfTitleModel.shape);
equalsBorder(70);
print("Sample Tf-Idf vector of preprocessed title: ");
equalsBorder(70);
print(tfIdfTitleModel[0])
```

```
Some of the Features used in tf-idf vectorizing preprocessed titles:
======================================================================
['teach', 'teacher', 'teaching', 'team', 'tech', 'technology', 'texts', 'think', 'thinking', 'thir
d', 'time', 'today', 'together', 'tomorrow', 'tools', 'us', 'use', 'using', 'virtual', 'visual', '
want', 'wanted', 'way', 'wiggle', 'wiggles', 'wiggly', 'without', 'wobble', 'wobbling', 'wonder',
'words', 'work', 'working', 'world', 'write', 'writers', 'writing', 'year', 'yoga', 'young']
======================================================================
Shape of preprocessed title matrix after tf-idf vectorization:  (7000, 450)
======================================================================
Sample Tf-Idf vector of preprocessed title:
======================================================================
  (0, 414) 0.6015041192948177
  (0, 154) 0.5796749237887321
  (0, 179) 0.5496997154828203
```

Average Word2Vector Vectorization

Average Word2Vector Vectorization

In [258]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/
# We should have glove_vectors file for creating below model
with open('glove_vectors', 'rb') as f:
    gloveModel = pickle.load(f)
    gloveWords =  set(gloveModel.keys())
```

In [259]:

```
print("Glove vector of sample word: ");
equalsBorder(70);
print(gloveModel['technology']);
equalsBorder(70);
print("Shape of glove vector: ", gloveModel['technology'].shape);
```

```
Glove vector of sample word:
======================================================================
[-0.26078   -0.36898   -0.022831   0.21666    0.16672   -0.20268
 -3.1219     0.33057    0.71512    0.28874    0.074368  -0.033203
  0.23783    0.21052    0.076562   0.13007   -0.31706   -0.45888
 -0.45463   -0.13191    0.49761    0.072704   0.16811    0.18846
 -0.16688   -0.21973    0.08575   -0.19577   -0.2101    -0.32436
 -0.56336    0.077996  -0.22758   -0.66569    0.14824    0.038945
  0.50881   -0.1352     0.49966   -0.4401    -0.022335  -0.22744
  0.22086    0.21865    0.36647    0.30495   -0.16565    0.038759
  0.28108   -0.2167     0.12453    0.65401    0.34584   -0.2557
 -0.046363  -0.31111   -0.020936  -0.17122   -0.77114    0.29289
 -0.14625    0.39541   -0.078938   0.051127   0.15076    0.085126
  0.183     -0.06755    0.26312    0.0087276  0.0066415  0.37033
  0.03496   -0.12627   -0.052626  -0.34897    0.14672    0.14799
 -0.21821   -0.042785   0.2661    -1.1105     0.31789    0.27278
  0.054468  -0.27458    0.42732   -0.44101   -0.19302   -0.32948
  0.61501   -0.22301   -0.36354   -0.34983   -0.16125   -0.17195
 -3.363      0.45146   -0.13753    0.31107    0.2061     0.33063
  0.45879    0.24256    0.042342   0.074837  -0.12869    0.12066
  0.42843   -0.4704    -0.18937    0.32685    0.26079    0.20518
 -0.18432   -0.47658    0.69193    0.18731   -0.12516    0.35447
 -0.1969    -0.58981   -0.88914    0.5176     0.13177   -0.078557
  0.032963  -0.19411    0.15109    0.10547   -0.1113    -0.61533
  0.0948    -0.3393    -0.20071   -0.30197    0.29531    0.28017
  0.16049    0.25294   -0.44266   -0.39412    0.13486    0.25178
 -0.044114   1.1519     0.32234   -0.34323   -0.10713   -0.15616
  0.031206   0.46636   -0.52761   -0.39296   -0.068424  -0.04072
  0.41508   -0.34564    0.71001   -0.364      0.2996     0.032281
  0.34035    0.23452    0.78342    0.48045   -0.1609     0.40102
 -0.071795  -0.16531    0.082153   0.52065    0.24194    0.17113
  0.33552   -0.15725   -0.38984    0.59337   -0.19388   -0.39864
 -0.47901    1.0835     0.24473    0.41309    0.64952    0.46846
  0.024386  -0.72087   -0.095061   0.10095   -0.025229   0.29435
 -0.57696    0.53166   -0.0058338 -0.3304     0.19661   -0.085206
  0.34225    0.56262    0.19924   -0.027111  -0.44567    0.17266
  0.20887   -0.40702    0.63954    0.50708   -0.31862   -0.39602
 -0.1714    -0.040006  -0.45077   -0.32482   -0.0316     0.54908
 -0.1121     0.12951   -0.33577   -0.52768   -0.44592   -0.45388
  0.66145    0.33023   -1.9089     0.5318     0.21626   -0.13152
  0.48258    0.68028   -0.84115   -0.51165    0.40017    0.17233
 -0.033749   0.045275   0.37398   -0.18252    0.19877    0.1511
  0.029803   0.16657   -0.12987   -0.50489    0.55311   -0.22504
  0.13085   -0.78459    0.36481   -0.27472    0.031805   0.53052
 -0.20078    0.46392   -0.63554    0.040289  -0.19142   -0.0097011
  0.068084  -0.10602    0.25567    0.096125  -0.10046    0.15016
 -0.26733   -0.26494    0.057888   0.062678  -0.11596    0.28115
  0.25375   -0.17954    0.20615    0.24189    0.062696   0.27719
 -0.42601   -0.28619   -0.44697   -0.082253  -0.73415   -0.20675
 -0.60289   -0.06728    0.15666   -0.042614   0.41368   -0.17367
 -0.54012    0.23883    0.23075    0.13608   -0.058634  -0.089705
  0.18469    0.023634   0.16178    0.23384    0.24267    0.091846 ]
======================================================================
Shape of glove vector:  (300,)
```

In [260]:

```
def getWord2VecVectors(texts):
    word2VecTextsVectors = [];
    for preProcessedText in tqdm(texts):
        word2VecTextVector = np.zeros(300);
        numberOfWordsInText = 0;
        for word in preProcessedText.split():
            if word in gloveWords:
                word2VecTextVector += gloveModel[word];
                numberOfWordsInText += 1;
        if numberOfWordsInText != 0:
            word2VecTextVector = word2VecTextVector / numberOfWordsInText;
        word2VecTextsVectors.append(word2VecTextVector);
    return word2VecTextsVectors;
```

1. Vectorizing project_essay

In [261]:

```
word2VecEssaysVectors = getWord2VecVectors(preProcessedEssaysWithoutStopWords);
```

In [262]:

```
print("Shape of Word2Vec vectorization matrix of essays: {},{}".format(len(word2VecEssaysVectors),
len(word2VecEssaysVectors[0])));
equalsBorder(70);
print("Sample essay: ");
equalsBorder(70);
print(preProcessedEssaysWithoutStopWords[0]);
equalsBorder(70);
print("Word2Vec vector of sample essay: ");
equalsBorder(70);
print(word2VecEssaysVectors[0]);
```

```
Shape of Word2Vec vectorization matrix of essays: 7000,300
======================================================================
Sample essay:
======================================================================
students fabulous 28 come diverse backgrounds not school gone school since preschool several kiddo
s new school year school ib school really puts global spin everything years passed seem get parent
involvement great district getting technology based however never seem enough equipment implement
awesome see kids computers though even able help well crucial develop skills needed succeed techno
logcally driven world students grow testing completely dependent upon online assessments computers
classroom would benefit areas learning example students still stress trying work computer often ti
mes students opportunity work computer since many students not one home hope children chance use c
hrome books classroom various research well supplementing reading math nannan
======================================================================
Word2Vec vector of sample essay:
======================================================================
[ 1.49061810e-02  6.52558757e-02 -1.43678057e-02 -1.28303819e-01
  9.42167810e-03 -1.30810600e-02 -3.22363848e+00  7.95128848e-02
  6.25677600e-02 -1.95817610e-02  2.54399438e-02  3.23909354e-02
  8.05384857e-02 -1.79061116e-01 -3.29969723e-02  2.64042429e-02
  2.90879133e-02 -8.70333171e-02  1.88762905e-02 -5.63153562e-02
  3.96180619e-02  1.57007429e-02  1.93120190e-02  7.76476190e-04
 -2.28322629e-02 -5.40534015e-02  1.54088990e-01 -4.77145505e-02
 -2.58009486e-02 -2.34970229e-02 -3.21928190e-01 -9.26814027e-02
  3.96414190e-03  7.07953648e-02 -7.53878905e-02 -1.78862771e-02
 -7.19549019e-02  2.13823990e-02  3.30768762e-03 -9.36592571e-02
 -9.74012829e-02  8.00393390e-02  4.49745784e-02 -1.39881925e-01
  4.47435810e-03 -5.73865452e-02  7.62741867e-02 -2.03076987e-02
  2.91002088e-02 -1.08104509e-01 -2.37050054e-02 -7.20011200e-03
  9.78194448e-02 -7.69557714e-03  2.29989157e-02 -7.31569357e-02
  1.11606086e-01  3.88722448e-03 -9.62250343e-02  9.52568248e-02
 -9.84247974e-02 -4.23952095e-02  4.77372253e-02 -1.33810905e-02
 -1.29412090e-01  9.77392295e-02  1.37441140e-02 -1.28603053e-01
  1.52135783e-01 -1.38746838e-01 -5.53740095e-02  5.43887124e-02
  1.29622743e-02 -9.00837010e-02 -1.75483657e-02 -1.36369810e-01
  6.26323876e-02 -5.89116105e-02 -3.38936210e-03  2.14175238e-03
  8.13706285e-02 -5.44890724e-01 -2.91990686e-02  1.54573803e-02
 -1.24788853e-01  1.75727410e-03  7.74660720e-02 -6.83139343e-02
  1.64887142e-01  2.40840648e-02  5.39219402e-02 -3.86499429e-03
```

```
  5.18825524e-03 -1.51815362e-02 -4.64992667e-03 -2.21532350e-01
 -2.18386057e+00  7.43904596e-02  8.01049876e-02  8.47093524e-02
 -8.52057571e-02 -2.82030857e-02  1.42488163e-01 -1.93963229e-02
  9.69581657e-02 -2.38013333e-03  3.60919390e-02 -1.93294182e-01
  3.79273971e-02 -4.05292410e-02 -3.77899629e-02 -1.36724829e-02
  5.98354752e-02  2.39942881e-01  1.91360670e-02  9.76250276e-02
 -1.62910524e-01  4.40213457e-02  1.05092801e-01  9.17391562e-02
  3.64910610e-02  6.31972333e-02  4.31353390e-03 -1.02185125e-01
  6.38848611e-02  4.77437143e-03  1.75592308e-01 -1.25817190e-02
  2.23183253e-02  1.15608188e-01  3.22266667e-02 -2.64332933e-02
 -1.41925457e-02 -7.81215695e-02 -1.78145410e-02 -9.08490676e-02
  1.52726933e-01 -1.50892364e-02  1.18049842e-01  3.26353952e-01
  7.64461118e-02 -1.14551314e-02  3.67904325e-02  1.65532105e-02
 -1.33358890e-02 -2.95486333e-02  7.46635038e-02 -6.20239524e-02
  9.15115905e-02 -1.37509829e-02  1.16829695e-02  5.30815629e-02
  5.53774640e-02 -9.74422590e-02  1.64010933e-02 -6.21165475e-02
  2.75945424e-02 -9.82623220e-02 -5.96905866e-02 -8.02854000e-03
 -5.07631381e-02  5.14076190e-02  4.29990276e-02 -8.99960952e-03
 -3.46881095e-02  2.45070971e-02 -3.55531933e-02  6.23711714e-02
  5.30126124e-02 -9.26758054e-02 -5.07322590e-02  1.15033495e-02
 -6.14035533e-02 -1.54005253e-01 -3.21370018e-02 -1.09313410e-02
 -2.83332929e-02  5.06590467e-02 -1.93806688e-01 -6.68872200e-02
 -4.50481419e-02  3.27339806e-01  8.35749288e-02  5.54999208e-02
 -9.76370076e-02 -6.61070924e-02 -6.25932733e-02 -8.96482248e-02
  5.12146724e-02  1.43687543e-02 -2.16561905e-04 -1.14627587e-01
 -4.99746962e-02 -5.21189048e-02 -1.83170952e-02 -5.41256248e-02
 -7.72621629e-02  6.93210100e-02 -3.18092971e-03  4.26500000e-02
  2.01932752e-01 -4.58118038e-02 -7.45845010e-02  1.14499295e-01
 -8.99270219e-02 -6.01932238e-03  8.59255732e-02 -7.82706476e-02
  1.58964628e-01  8.88997571e-02  3.74597790e-02  2.15635524e-02
  2.36393916e-02 -1.91459472e-01 -5.78065982e-02  6.77884762e-04
 -1.19065876e-01 -2.31327362e-02 -6.41118359e-03 -2.97078381e-02
 -1.08267936e-01 -5.67281524e-02 -1.01334053e-01 -7.04125448e-02
 -2.14719310e+00  7.75323248e-02  6.96190514e-02  2.71307229e-02
  2.92672467e-02 -1.61779657e-01 -1.17733416e-02 -1.03904924e-02
 -3.18773648e-02 -3.94456895e-02 -1.07951923e-01  2.99152571e-02
  6.40404000e-02 -4.94175419e-02  4.10479524e-02  1.41728784e-01
 -6.46432343e-02  8.02121781e-02 -2.80961586e-01  7.87245026e-02
 -4.06311524e-02  8.31997143e-03 -2.80615038e-02 -1.17425733e-01
 -3.78634962e-02  7.05004762e-05 -1.07941257e-02  1.20675778e-01
  9.71281629e-02 -1.26198470e-02  1.00502724e-01 -3.13208800e-02
  1.16299637e-01 -6.12492390e-02  1.05643487e-01 -6.80041333e-02
 -5.31333419e-02  3.19802057e-02 -1.74131848e-02 -1.52405522e-02
  8.25410981e-02 -1.12619040e-01 -1.51408959e-01  1.95956971e-02
  2.28653737e-02  1.04645468e-01 -2.42338952e-02 -4.79787371e-02
 -6.01754876e-02  8.14147655e-02 -4.02032410e-02  3.27735810e-02
  9.61516152e-02  4.31853200e-02 -7.34511146e-02  9.49344667e-02
  1.49184543e-01 -5.51600495e-02 -3.34311629e-02  8.74265766e-02
  1.83554657e-02  8.81127011e-02  2.73279552e-02  1.87663381e-02
  3.07020857e-02 -5.22588762e-03  8.20059333e-02  1.05391048e-03
 -1.05486146e-01 -1.71025709e-01  3.28232223e-02 -2.02283276e-02
 -2.83652638e-02  1.51836766e-01  1.55608536e-01  4.52351476e-02]
```

2. Vectorizing project_title

In [263]:

```
word2VecTitlesVectors = getWord2VecVectors(preProcessedProjectTitlesWithoutStopWords);
```

In [264]:

```
print("Shape of Word2Vec vectorization matrix of project titles: {},
{}".format(len(word2VecTitlesVectors), len(word2VecTitlesVectors[0])));
equalsBorder(70);
print("Sample title: ");
equalsBorder(70);
print(preProcessedProjectTitlesWithoutStopWords[0]);
equalsBorder(70);
print("Word2Vec vector of sample title: ");
equalsBorder(70);
print(word2VecTitlesVectors[0]);
```

```
Shape of Word2Vec vectorization matrix of project titles: 7000, 300
======================================================================
Sample title:
======================================================================
tech fabulous first graders
======================================================================
Word2Vec vector of sample title:
======================================================================
[ 1.00052500e-01  9.91145000e-02 -6.97675000e-02 -9.66325000e-03
  2.68280000e-02 -1.26450825e-01 -2.14432500e+00 -1.55777250e-01
  4.79625000e-02 -3.29375000e-02  1.04538000e-01  3.33990000e-03
  4.91177500e-02 -3.65375000e-02 -1.69414500e-01 -3.42555000e-01
  3.03137250e-02 -2.44207500e-01 -2.06472750e-01  3.72627500e-02
  1.07215000e-02  1.95850000e-01  2.62435000e-02 -1.05600000e-02
 -3.75005000e-02 -2.18662500e-02  3.68045000e-01  1.13390000e-02
 -3.82630000e-02  5.93500000e-02 -3.34545000e-01 -3.12827500e-01
  9.03075000e-02  2.39504750e-01 -1.01084000e-01  1.58775750e-01
 -6.67825000e-03 -1.93450000e-02  2.91756500e-01 -1.90900000e-02
 -3.45407500e-01  1.16209000e-01  3.72802500e-01 -1.32015250e-01
  8.28187500e-02 -4.74750000e-04  4.24945000e-02 -2.51698500e-01
 -6.14992500e-02  1.62057925e-01  1.11650750e-01  1.87273450e-01
  1.81227800e-01  1.43492900e-01 -1.44450000e-03  6.79977500e-02
  9.59970000e-02  1.92935750e-01  7.65750000e-03  1.22458250e-01
 -7.19917500e-02  9.40367500e-02  6.24847500e-02 -7.88814250e-02
  8.02253500e-02  2.85150000e-02  1.31234500e-01 -1.47464750e-01
  2.20777500e-01 -9.95242500e-02 -3.14325000e-01 -1.56425000e-02
  1.44319250e-01  1.08357450e-01 -2.17931000e-02 -5.52417500e-01
  5.80770000e-02 -2.18625000e-03 -8.96925000e-02 -7.08269500e-02
  1.81405500e-01 -1.49429500e-01 -2.08885000e-01  8.02750000e-02
 -3.48300000e-02 -1.79935000e-02  4.13762500e-01 -1.48015500e-01
  2.85275000e-02  3.61800000e-02  3.16125000e-02 -1.71342500e-01
 -2.01930000e-01  9.67630000e-02 -5.33325000e-02 -2.63218525e-01
 -1.72969500e+00 -1.42175000e-03 -9.00200000e-02  1.43765000e-01
 -3.05225000e-02 -2.56302725e-01  1.02675000e-02 -8.83537500e-02
  1.57870750e-01  4.66207500e-02  3.68882500e-01 -1.24530000e-02
  5.85325000e-03  9.13250000e-04 -1.25367500e-01 -1.03681750e-01
 -1.95895250e-01  3.14932250e-01 -7.48615000e-02  2.52950750e-01
  8.62825000e-02 -7.57522500e-02 -5.24555000e-02 -6.78635000e-02
 -1.54075000e-01 -6.73112000e-02  1.13137500e-01  1.40475000e-01
  1.20838500e-01  2.70200000e-02 -4.54650000e-02  1.24811250e-01
 -2.09042500e-01  3.57332500e-01  4.99700000e-02 -1.39805000e-01
  6.34735000e-02 -3.98488500e-01  1.19735000e-02 -3.37635000e-01
  1.56304250e-01 -1.91688500e-01  2.11543025e-01  6.39410000e-01
  5.88788600e-02  6.38805000e-02  2.26505000e-01 -9.91678750e-02
 -3.55072500e-01 -5.73875000e-02  6.59610250e-02  1.10930500e-01
  5.58467500e-02  2.36490000e-03 -3.90600000e-02  7.14600000e-02
  7.55475000e-02  1.60020000e-01 -4.36650000e-02  2.29007500e-02
  1.60401250e-01 -2.19985000e-01 -3.72655000e-02 -1.45885750e-01
  6.48322500e-02 -1.56200000e-02  2.56487000e-01 -2.83729750e-01
  1.35845000e-01  2.04923750e-01  1.66353500e-01  3.34024500e-01
 -6.22600000e-03  2.96970000e-02  2.23000000e-02  1.87552250e-01
  1.46681500e-01 -2.67049250e-01 -6.78085000e-02  5.53500000e-02
 -1.63967500e-01  1.26275000e-01 -2.10470000e-01  8.95160000e-02
  1.15760000e-01  3.86817250e-01  3.50937500e-02 -8.31210000e-02
 -5.49325000e-02 -9.73295000e-02 -3.48530000e-01 -2.86797500e-01
 -3.83325000e-02  1.56020000e-01  3.28373000e-01  8.37420000e-02
 -2.87845000e-01  1.72554750e-01  5.13729000e-02  1.94373500e-01
 -2.90799375e-01  7.12275000e-02 -7.78900000e-03  3.01036000e-01
 -1.31947500e-01 -1.32327500e-01 -1.91316750e-01  2.05357875e-01
 -1.66707500e-01  9.04914750e-02  1.44392500e-01 -3.23463000e-01
  3.32635000e-01  1.12041750e-01  2.98435000e-01  8.68175000e-02
 -1.44494000e-01 -7.06522750e-02  3.32960000e-02  1.74012500e-01
 -2.72890750e-01 -6.89556500e-02 -3.59497500e-01 -7.05425000e-02
 -2.71815000e-01  2.83750000e-02  1.29755000e-02 -1.86307900e-01
 -2.44947500e+00 -1.69311500e-01 -5.33465000e-02  1.51062750e-01
 -2.09152500e-01  1.75620000e-01 -2.77640000e-01 -3.17949250e-01
  2.27077500e-01 -1.27127250e-01  2.40700000e-02  1.64424750e-01
 -1.24310000e-01  1.63615000e-02 -4.71105000e-02  9.94997500e-02
  1.07012500e-01  1.69740950e-01 -2.34955750e-01  3.74030000e-01
 -1.86867500e-02 -1.32065000e-01 -1.52655000e-03 -8.69540000e-02
 -1.02921000e-01  4.25229500e-02  3.29115000e-01  1.58406750e-01
  1.22545500e-01 -1.82327500e-01  1.94307500e-01  2.37745000e-01
  1.48367500e-01  1.03240000e-01  4.59645000e-02 -4.16956000e-01
 -9.81750000e-02 -1.55917500e-01  3.00685000e-02 -5.33750000e-04
 -1.47285250e-01 -2.13601750e-01 -1.53705000e-01 -2.05827000e-01
  9.19325000e-02  3.81162500e-02 -5.28310000e-02  1.09647500e-01
 -4.74710000e-02  1.52787275e-01 -1.79792500e-01  1.15987250e-01
```

```
  1.31780750e-01  1.26174650e-01 -1.97060000e-01  9.89209750e-02
  4.23635000e-02 -4.31300000e-02 -1.49456500e-01  2.17650500e-01
  2.95432000e-01  2.96056750e-01  2.49189500e-01 -3.08932000e-02
 -2.36405000e-01  4.55415000e-02 -2.73627500e-02 -1.75581500e-01
  1.14092500e-02 -1.68607500e-01  9.61325000e-02  1.52908000e-01
  9.48382500e-02  1.57875000e-01 -2.11775000e-03 -6.03492500e-02]
```

Tf-Idf Weighted Word2Vec Vectorization

1. Vectorizing project_essay

In [265]:

```
# Initializing tfidf vectorizer
tfIdfEssayTempVectorizer = TfidfVectorizer();
# Vectorizing preprocessed essays using tfidf vectorizer initialized above
tfIdfEssayTempVectorizer.fit(preProcessedEssaysWithoutStopWords);
# Saving dictionary in which each word is key and it's idf is value
tfIdfEssayDictionary = dict(zip(tfIdfEssayTempVectorizer.get_feature_names(),
list(tfIdfEssayTempVectorizer.idf_)));
# Creating set of all unique words used by tfidf vectorizer
tfIdfEssayWords = set(tfIdfEssayTempVectorizer.get_feature_names());
```

In [266]:

```
# Creating list to save tf-idf weighted vectors of essays
tfIdfWeightedWord2VecEssaysVectors = [];
# Iterating over each essay
for essay in tqdm(preProcessedEssaysWithoutStopWords):
    # Sum of tf-idf values of all words in a particular essay
    cumulativeSumTfIdfWeightOfEssay = 0;
    # Tf-Idf weighted word2vec vector of a particular essay
    tfIdfWeightedWord2VecEssayVector = np.zeros(300);
    # Splitting essay into list of words
    splittedEssay = essay.split();
    # Iterating over each word
    for word in splittedEssay:
        # Checking if word is in glove words and set of words used by tfIdf essay vectorizer
        if (word in gloveWords) and (word in tfIdfEssayWords):
            # Tf-Idf value of particular word in essay
            tfIdfValueWord = tfIdfEssayDictionary[word] * (essay.count(word) / len(splittedEssay));
            # Making tf-idf weighted word2vec
            tfIdfWeightedWord2VecEssayVector += tfIdfValueWord * gloveModel[word];
            # Summing tf-idf weight of word to cumulative sum
            cumulativeSumTfIdfWeightOfEssay += tfIdfValueWord;
    if cumulativeSumTfIdfWeightOfEssay != 0:
        # Taking average of sum of vectors with tf-idf cumulative sum
        tfIdfWeightedWord2VecEssayVector = tfIdfWeightedWord2VecEssayVector /
cumulativeSumTfIdfWeightOfEssay;
    # Appending the above calculated tf-idf weighted vector of particular essay to list of vectors
of essays
    tfIdfWeightedWord2VecEssaysVectors.append(tfIdfWeightedWord2VecEssayVector);
```

In [267]:

```
print("Shape of Tf-Idf weighted Word2Vec vectorization matrix of project essays: {}, {}".format(le
n(tfIdfWeightedWord2VecEssaysVectors), len(tfIdfWeightedWord2VecEssaysVectors[0])));
equalsBorder(70);
print("Sample Essay: ");
equalsBorder(70);
print(preProcessedEssaysWithoutStopWords[0]);
equalsBorder(70);
print("Tf-Idf Weighted Word2Vec vector of sample essay: ");
equalsBorder(70);
print(tfIdfWeightedWord2VecEssaysVectors[0]);
```

```
Shape of Tf-Idf weighted Word2Vec vectorization matrix of project essays: 7000, 300
======================================================================
Sample Essay:
======================================================================
```

students fabulous 28 come diverse backgrounds not school gone school since preschool several kiddo
s new school year school ib school really puts global spin everything years passed seem get parent
involvement great district getting technology based however never seem enough equipment implement
awesome see kids computers though even able help well crucial develop skills needed succeed techno
logcally driven world students grow testing completely dependent upon online assessments computers
classroom would benefit areas learning example students still stress trying work computer often ti
mes students opportunity work computer since many students not one home hope children chance use c
hrome books classroom various research well supplementing reading math nannan
==========
Tf-Idf Weighted Word2Vec vector of sample essay:
==========
[4.13339263e-02 4.54468323e-02 -2.15963083e-02 -1.33372599e-01
  3.52767667e-02 -5.56711700e-02 -3.14476170e+00 9.64450904e-02
  9.85143156e-02 2.97745051e-02 4.42038534e-02 5.07920451e-02
  7.53332366e-02 -1.94844889e-01 -6.07536666e-02 5.02041173e-02
  2.09106497e-02 -6.38587962e-02 -1.23290810e-02 -6.72650725e-02
  5.57016794e-03 -6.98010311e-03 6.06463122e-02 8.80828491e-03
 -4.82301286e-02 -4.44096137e-02 1.82955359e-01 -3.39746873e-02
 -4.00856483e-02 -4.04733585e-02 -3.09549887e-01 -1.14509756e-01
 -3.68796725e-02 7.03677126e-02 -9.93691763e-02 4.04823854e-02
 -8.55008330e-02 4.11518156e-02 1.11421393e-02 -1.10246966e-01
 -1.14544283e-01 3.56162255e-02 6.05907492e-02 -1.36043255e-01
 -8.71257187e-03 -4.42502435e-02 8.31862461e-02 1.08621463e-02
  3.42925473e-02 -1.06292994e-01 -4.08213447e-02 3.32457023e-02
  1.11409632e-01 2.58576456e-02 6.69767905e-03 -8.38373134e-02
  1.09870981e-01 2.53415604e-02 -1.39142581e-01 8.45704726e-02
 -9.65988039e-02 1.10131398e-02 3.32181303e-02 -1.91572268e-02
 -1.23883509e-01 1.31087694e-01 -4.00919823e-02 -1.30279706e-01
  1.70962730e-01 -1.38158539e-01 9.30334593e-03 8.19545284e-02
  2.64314673e-02 -1.01520691e-01 -1.60520966e-02 -1.21896567e-01
  9.17956775e-02 -7.55330004e-02 -1.15602323e-02 -6.88251738e-05
  9.63455229e-02 -5.64492101e-01 -1.72637815e-02 3.26332272e-02
 -1.10519885e-01 -1.48620084e-03 5.80039223e-02 -5.39562180e-02
  1.75414476e-01 4.89414273e-02 7.39343536e-02 -1.92334442e-03
  1.22513847e-02 -4.28589491e-02 -7.13296581e-03 -2.07689163e-01
 -2.12935153e+00 9.72520416e-02 5.69438493e-02 7.63529019e-02
 -1.02369385e-01 -6.49243135e-02 1.28349346e-01 -3.67959569e-02
  8.05570243e-02 -1.16326555e-02 4.76741621e-02 -1.71951650e-01
  5.02304951e-02 -5.60924781e-02 -3.64488390e-02 -9.59348260e-03
  7.45917984e-02 2.24243180e-01 3.39052736e-02 1.10533300e-01
 -1.13756816e-01 3.67748007e-02 1.51660370e-01 9.53027167e-02
  1.64912687e-03 9.77162679e-02 -6.90842044e-03 -8.49915527e-02
  7.49741287e-02 -7.74266206e-03 2.00307378e-01 9.65336171e-03
  5.33310078e-02 1.22626493e-01 3.05360057e-02 -2.57168672e-02
  7.81868809e-03 -8.04501024e-02 -5.65111981e-02 -8.09453963e-02
  1.60366216e-01 1.40902247e-02 1.26079108e-01 3.05892986e-01
  4.96348405e-02 -4.10888188e-02 3.40021650e-02 2.97113820e-02
 -6.52308340e-04 -2.53399744e-02 6.67111205e-02 -1.00204279e-01
  2.36721629e-02 4.54912853e-03 5.40288069e-03 8.50803673e-02
  5.87243426e-02 -8.55671002e-02 -2.28193922e-02 -1.16688138e-01
  2.05709823e-02 -1.35576231e-01 -6.80868450e-02 -4.71451163e-03
 -1.04993533e-01 7.10526210e-02 6.22205038e-02 2.38584367e-02
 -3.18370318e-02 3.42745336e-02 5.48775837e-04 7.33160058e-02
  5.04124712e-02 -7.11952984e-02 -6.46023131e-02 1.73217869e-02
 -7.65619354e-02 -1.66682635e-01 -2.93144075e-02 -2.70161779e-02
 -2.67395397e-02 5.83509521e-02 -1.96843470e-01 -6.56704850e-02
 -4.61463885e-02 3.36967792e-01 1.04877774e-01 8.38073639e-02
 -1.23719370e-01 -4.55728981e-02 -4.20529981e-02 -1.13126897e-01
  5.34802412e-02 5.48877605e-03 3.74600791e-02 -1.16082396e-01
 -3.37850135e-02 -4.91725552e-02 -1.05707555e-02 -5.49577077e-02
 -7.64720529e-02 1.00761406e-01 -1.40224870e-02 5.42909032e-02
  2.06458719e-01 -5.63287666e-02 -5.03999531e-02 1.18943118e-01
 -9.43178369e-02 -2.59447225e-02 1.09081598e-01 -8.60561624e-02
  1.62058626e-01 1.04841576e-01 5.48102788e-02 2.45877179e-02
 -1.29798292e-03 -1.81372789e-01 -4.44567480e-02 4.41946224e-03
 -1.30801488e-01 -4.64021785e-04 -5.15929072e-03 -5.21829730e-02
 -1.19401785e-01 -2.62141872e-02 -9.75340159e-02 -5.99604466e-02
 -2.14494788e+00 6.93825782e-02 9.87253945e-02 1.31138518e-02
  3.36838964e-02 -1.23845478e-01 -2.81508728e-02 -8.66516382e-03
 -1.46423967e-04 -5.06974785e-02 -1.00393656e-01 -4.37749459e-03
  8.57197553e-02 -5.50337312e-02 4.78796439e-02 1.61393494e-01
 -9.77068400e-02 8.76481426e-02 -2.84127990e-01 6.85118012e-02
 -2.59655187e-02 2.25981238e-02 1.81813615e-02 -1.26801394e-01
 -4.73385461e-02 -1.07190818e-02 -2.64654928e-02 8.78153760e-02
  1.16679585e-01 1.30529761e-02 9.04272242e-02 -7.30627285e-03
  1.25798682e-01 -5.70091420e-02 8.76330211e-02 -1.13051484e-01
 -1.19068925e-01 6.84649978e-02 -3.13648935e-02 -1.05485446e-02

```
 6.30621897e-02 -1.06194234e-01 -1.59703787e-01 -3.45683432e-02
-7.10815191e-04  1.06275502e-01 -1.95226034e-03 -7.25314807e-02
-4.05456765e-02  1.02716098e-01 -2.64288299e-02  2.79967072e-02
 8.03061765e-02  2.03222787e-02 -9.79105291e-02  1.15234008e-01
 1.26759296e-01 -5.21636625e-02 -3.67425025e-02  1.00228638e-01
-7.29014572e-03  3.82932569e-02  3.70887558e-02  3.01565739e-02
 5.32144422e-02  7.62103437e-03  1.24479243e-01  2.34322498e-02
-1.27395603e-01 -1.93865355e-01  2.41654589e-02  7.16502529e-03
-3.60070849e-02  1.45274528e-01  1.57089586e-01  3.32456822e-02]
```

2. Vectorizing project_title

In [268]:

```
# Initializing tfidf vectorizer
tfIdfTitleTempVectorizer = TfidfVectorizer();
# Vectorizing preprocessed titles using tfidf vectorizer initialized above
tfIdfTitleTempVectorizer.fit(preProcessedProjectTitlesWithoutStopWords);
# Saving dictionary in which each word is key and it's idf is value
tfIdfTitleDictionary = dict(zip(tfIdfTitleTempVectorizer.get_feature_names(),
list(tfIdfTitleTempVectorizer.idf_)));
# Creating set of all unique words used by tfidf vectorizer
tfIdfTitleWords = set(tfIdfTitleTempVectorizer.get_feature_names());
```

In [269]:

```
# Creating list to save tf-idf weighted vectors of project titles
tfIdfWeightedWord2VecTitlesVectors = [];
# Iterating over each title
for title in tqdm(preProcessedProjectTitlesWithoutStopWords):
    # Sum of tf-idf values of all words in a particular project title
    cumulativeSumTfIdfWeightOfTitle = 0;
    # Tf-Idf weighted word2vec vector of a particular project title
    tfIdfWeightedWord2VecTitleVector = np.zeros(300);
    # Splitting title into list of words
    splittedTitle = title.split();
    # Iterating over each word
    for word in splittedTitle:
        # Checking if word is in glove words and set of words used by tfIdf title vectorizer
        if (word in gloveWords) and (word in tfIdfTitleWords):
            # Tf-Idf value of particular word in title
            tfIdfValueWord = tfIdfTitleDictionary[word] * (title.count(word) / len(splittedTitle));
            # Making tf-idf weighted word2vec
            tfIdfWeightedWord2VecTitleVector += tfIdfValueWord * gloveModel[word];
            # Summing tf-idf weight of word to cumulative sum
            cumulativeSumTfIdfWeightOfTitle += tfIdfValueWord;
    if cumulativeSumTfIdfWeightOfTitle != 0:
        # Taking average of sum of vectors with tf-idf cumulative sum
        tfIdfWeightedWord2VecTitleVector = tfIdfWeightedWord2VecTitleVector /
cumulativeSumTfIdfWeightOfTitle;
    # Appending the above calculated tf-idf weighted vector of particular title to list of vectors
of project titles
    tfIdfWeightedWord2VecTitlesVectors.append(tfIdfWeightedWord2VecTitleVector);
```

In [270]:

```
print("Shape of Tf-Idf weighted Word2Vec vectorization matrix of project titles: {}, {}".format(le
n(tfIdfWeightedWord2VecTitlesVectors), len(tfIdfWeightedWord2VecTitlesVectors[0])));
equalsBorder(70);
print("Sample Title: ");
equalsBorder(70);
print(preProcessedProjectTitlesWithoutStopWords[0]);
equalsBorder(70);
print("Tf-Idf Weighted Word2Vec vector of sample title: ");
equalsBorder(70);
print(tfIdfWeightedWord2VecTitlesVectors[0]);
```

```
Shape of Tf-Idf weighted Word2Vec vectorization matrix of project titles: 7000, 300
======================================================================
Sample Title:
======================================================================
```

```
tech fabulous first graders
==========================================================================
Tf-Idf Weighted Word2Vec vector of sample title:
==========================================================================
[ 9.78589882e-02  9.71154845e-02 -7.67262528e-02 -3.43157875e-03
  4.50217140e-02 -1.10826929e-01 -2.14400460e+00 -2.43960543e-01
  4.13823441e-02 -6.65901193e-02  9.32822291e-02 -4.17724166e-02
  6.81919244e-02 -2.62063720e-03 -1.97371808e-01 -3.35290994e-01
 -7.15776492e-03 -2.43866837e-01 -2.01757900e-01  5.47930695e-02
  1.11921838e-02  1.89561351e-01  2.89679151e-02  1.07161779e-02
 -1.46625734e-02 -4.36213237e-03  3.59914215e-01  4.87508433e-02
 -5.07117845e-02  6.75814626e-02 -2.92888932e-01 -3.19656578e-01
  1.11510394e-01  2.41221656e-01 -9.92345449e-02  1.40669131e-01
  9.47748782e-03 -3.36532642e-02  2.96202003e-01 -3.06691712e-02
 -3.50884516e-01  1.15316237e-01  3.76390230e-01 -1.07714262e-01
  8.79559060e-02  1.08976800e-02  1.72932246e-02 -2.51256155e-01
 -5.88449384e-02  1.48464959e-01  9.90564920e-02  1.91786600e-01
  1.68245970e-01  1.38206071e-01 -9.11670736e-03  1.00252140e-01
  9.64599198e-02  1.41400884e-01  6.76177376e-02  1.35631541e-01
 -6.17126330e-02  8.70450021e-02  8.97696477e-02 -6.80325809e-02
  5.14201638e-02 -2.31362676e-02  1.22116414e-01 -1.22296819e-01
  1.67793203e-01 -9.41398638e-02 -2.89250858e-01 -3.08513541e-02
  1.37575163e-01  1.00603255e-01 -3.36943021e-02 -5.36124687e-01
  4.24986203e-02  1.08264149e-03 -6.20610057e-02 -6.12989455e-02
  1.76470831e-01 -1.19940467e-01 -2.30597150e-01  1.04848686e-01
 -3.03875432e-02 -4.96008034e-03  4.01527038e-01 -1.57045599e-01
 -2.17197071e-02  4.83406537e-02  1.92152279e-02 -1.33146501e-01
 -1.84959420e-01  1.12028972e-01 -7.54390169e-02 -2.87282845e-01
 -1.73926004e+00 -1.13588663e-02 -1.32053061e-01  1.08398216e-01
 -2.71096547e-02 -2.62229208e-01  4.42621442e-02 -7.68150602e-02
  1.32059903e-01  6.44074210e-02  3.09414555e-01 -1.84793016e-02
  3.76904909e-03 -1.06306252e-02 -1.40690895e-01 -1.14261653e-01
 -1.89767653e-01  2.74381819e-01 -1.06437595e-01  2.63936202e-01
  1.12010809e-01 -8.54701738e-02 -2.85253207e-02 -6.14417236e-02
 -1.25719366e-01 -6.41734008e-02  1.37251046e-01  1.40102263e-01
  1.05743455e-01  2.96344639e-02 -2.58392919e-02  1.41007299e-01
 -2.11990004e-01  3.47416678e-01  4.83551411e-03 -1.15474158e-01
  6.88011796e-02 -3.99262814e-01 -1.20265125e-02 -3.28347194e-01
  1.40454780e-01 -2.26043118e-01  1.96433396e-01  6.56375049e-01
  4.60352412e-02  9.62211889e-02  2.17994762e-01 -1.22144290e-01
 -3.46829157e-01 -6.68178159e-02  6.31511407e-02  1.13013665e-01
  7.38167659e-02 -9.27130196e-03 -2.72302471e-02  9.28743262e-02
  7.16791516e-02  1.61883813e-01 -9.25715747e-03  5.20901380e-02
  1.75532468e-01 -2.29420020e-01 -4.00584266e-02 -1.52845880e-01
  1.19723119e-01 -3.99084432e-02  2.64061712e-01 -2.71892820e-01
  1.52447871e-01  2.22148469e-01  1.72203653e-01  2.98943175e-01
  1.14230882e-02  3.67383158e-02  3.52103080e-02  1.85181166e-01
  1.45346606e-01 -2.67197851e-01 -6.34252845e-02  5.54609123e-02
 -1.52621378e-01  9.31345811e-02 -2.24929425e-01  6.55023085e-02
  1.35398008e-01  3.56077775e-01  5.91016287e-02 -7.51286405e-02
 -2.90584179e-02 -8.76761953e-02 -3.52558359e-01 -2.46418965e-01
  3.04946094e-02  1.41534116e-01  3.02366907e-01  7.51785706e-02
 -3.18194434e-01  1.91907878e-01  5.36139085e-02  2.27368492e-01
 -2.91627504e-01  6.98633420e-02 -4.81528258e-03  2.84621411e-01
 -1.41261955e-01 -1.16633144e-01 -1.48348113e-01  1.53086678e-01
 -2.00014149e-01  7.50733283e-02  1.23945652e-01 -3.08050097e-01
  3.26190272e-01  1.21643253e-01  2.95166537e-01  8.80848756e-02
 -1.37337875e-01 -6.46626476e-02 -9.08939159e-03  1.58027656e-01
 -2.85578715e-01 -7.41880275e-02 -3.17236823e-01 -8.73954222e-02
 -1.94103274e-01  2.19902856e-02  4.27776045e-02 -1.97574102e-01
 -2.46019807e+00 -2.02619859e-01 -1.62924736e-02  1.17363513e-01
 -2.19329315e-01  1.81913893e-01 -2.17037145e-01 -3.29539323e-01
  1.95620537e-01 -1.25279856e-01  2.30565656e-02  1.67091836e-01
 -1.51984917e-01 -9.49260202e-03 -2.83878341e-02  8.90425947e-02
  1.32382763e-01  1.57540559e-01 -2.14743260e-01  3.57470681e-01
 -3.50015943e-02 -8.74971401e-02 -1.85804878e-02 -6.59620331e-02
 -7.45014752e-02  2.21837610e-02  3.12134283e-01  1.64937473e-01
  1.33602540e-01 -2.01460817e-01  1.88029931e-01  2.36297870e-01
  1.49628881e-01  7.40816038e-02  3.26451706e-02 -4.23978274e-01
 -7.81092579e-02 -1.62195634e-01  4.29628416e-02 -7.83177146e-03
 -1.24689541e-01 -2.24511245e-01 -1.89336219e-01 -1.92194100e-01
  6.74340503e-02  4.23028820e-02 -7.53119756e-02  1.05847238e-01
 -8.50310245e-02  1.37653237e-01 -1.95793442e-01  9.94643581e-02
  9.44960318e-02  7.00958137e-02 -2.17410484e-01  9.20637901e-02
  6.24173030e-02 -2.56659017e-02 -1.37623453e-01  2.19793272e-01
  3.14603741e-01  3.22396363e-01  2.34191730e-01 -4.29883793e-02
 -2.47730500e-01  1.64746652e-02 -4.17495180e-02 -1.85398827e-01
```

```
 -1.23657560e-02 -1.51870151e-01  9.77281668e-02  1.41900575e-01
  7.70887168e-02  1.10211580e-01 -1.69772633e-02 -7.77251713e-02]
```

Vectorizing numerical features

1. Vectorizing price

In [271]:

```
# Standardizing the price data using StandardScaler(Uses mean and std for standardization)
priceScaler = StandardScaler();
priceScaler.fit(trainingData['price'].values.reshape(-1, 1));
priceStandardized = priceScaler.transform(trainingData['price'].values.reshape(-1, 1));
```

In [272]:

```
print("Shape of standardized matrix of prices: ", priceStandardized.shape);
equalsBorder(70);
print("Sample original prices: ");
equalsBorder(70);
print(trainingData['price'].values[0:5]);
print("Sample standardized prices: ");
equalsBorder(70);
print(priceStandardized[0:5]);
```

```
Shape of standardized matrix of prices:  (7000, 1)
======================================================================
Sample original prices:
======================================================================
[157.   251.4  448.45 191.57 104.9 ]
Sample standardized prices:
======================================================================
[[-0.3865048 ]
 [-0.12218254]
 [ 0.42956219]
 [-0.28970797]
 [-0.53238605]]
```

2. Vectorizing quantity

In [273]:

```
# Standardizing the quantity data using StandardScaler(Uses mean and std for standardization)
quantityScaler = StandardScaler();
quantityScaler.fit(trainingData['quantity'].values.reshape(-1, 1));
quantityStandardized = quantityScaler.transform(trainingData['quantity'].values.reshape(-1, 1));
```

In [274]:

```
print("Shape of standardized matrix of quantities: ", quantityStandardized.shape);
equalsBorder(70);
print("Sample original quantities: ");
equalsBorder(70);
print(trainingData['quantity'].values[0:5]);
print("Sample standardized quantities: ");
equalsBorder(70);
print(quantityStandardized[0:5]);
```

```
Shape of standardized matrix of quantities:  (7000, 1)
======================================================================
Sample original quantities:
======================================================================
[10 33  2  5 29]
Sample standardized quantities:
======================================================================
[[-0.26024153]
 [ 0.62611344]
 [-0.56853891]
```

```
 [-0.45292739]
 [ 0.47196475]]
```

3. Vectorizing teacher_number_of_previously_posted_projects

In [275]:

```
# Standardizing the teacher_number_of_previously_posted_projects data using StandardScaler(Uses me
an and std for standardization)
previouslyPostedScaler = StandardScaler();
previouslyPostedScaler.fit(trainingData['teacher_number_of_previously_posted_projects'].values.res
hape(-1, 1));
previouslyPostedStandardized =
previouslyPostedScaler.transform(trainingData['teacher_number_of_previously_posted_projects'].valu
es.reshape(-1, 1));
```

In [276]:

```
print("Shape of standardized matrix of teacher_number_of_previously_posted_projects: ",
previouslyPostedStandardized.shape);
equalsBorder(70);
print("Sample original quantities: ");
equalsBorder(70);
print(trainingData['teacher_number_of_previously_posted_projects'].values[0:5]);
print("Sample standardized teacher_number_of_previously_posted_projects: ");
equalsBorder(70);
print(previouslyPostedStandardized[0:5]);
```

```
Shape of standardized matrix of teacher_number_of_previously_posted_projects:  (7000, 1)
======================================================================
Sample original quantities:
======================================================================
[ 1  2 13  4  8]
Sample standardized teacher_number_of_previously_posted_projects:
======================================================================
[[-0.36304492]
 [-0.32731768]
 [ 0.06568198]
 [-0.2558632 ]
 [-0.11295423]]
```

In [277]:

```
numberOfPoints = 16638;
# Categorical data
categoriesVectorsSub = categoriesVectors[0:numberOfPoints];
subCategoriesVectorsSub = subCategoriesVectors[0:numberOfPoints];
teacherPrefixVectorsSub = teacherPrefixVectors[0:numberOfPoints];
schoolStateVectorsSub = schoolStateVectors[0:numberOfPoints];
projectGradeVectorsSub = projectGradeVectors[0:numberOfPoints];

# Text data
bowEssayModelSub = bowEssayModel[0:numberOfPoints];
bowTitleModelSub = bowTitleModel[0:numberOfPoints];
tfIdfEssayModelSub = tfIdfEssayModel[0:numberOfPoints];
tfIdfTitleModelSub = tfIdfTitleModel[0:numberOfPoints];
word2VecEssaysVectorsSub = word2VecEssaysVectors[0:numberOfPoints];
word2VecTitlesVectorsSub = word2VecTitlesVectors[0:numberOfPoints];
tfIdfWeightedWord2VecEssaysVectorsSub = tfIdfWeightedWord2VecEssaysVectors[0:numberOfPoints];
tfIdfWeightedWord2VecTitlesVectorsSub = tfIdfWeightedWord2VecTitlesVectors[0:numberOfPoints];

# Numerical data
priceStandardizedSub = priceStandardized[0:numberOfPoints];
quantityStandardizedSub = quantityStandardized[0:numberOfPoints];
previouslyPostedStandardizedSub = previouslyPostedStandardized[0:numberOfPoints];
```

In [278]:

```
def getAvgTfIdfEssayVectors(arrayOfTexts):
    # Creating list to save tf-idf weighted vectors of essays
    tfIdfWeightedWord2VecEssaysVectors = [];
```

```
    # Iterating over each essay
    for essay in tqdm(arrayOfTexts):
        # Sum of tf-idf values of all words in a particular essay
        cumulativeSumTfIdfWeightOfEssay = 0;
        # Tf-Idf weighted word2vec vector of a particular essay
        tfIdfWeightedWord2VecEssayVector = np.zeros(300);
        # Splitting essay into list of words
        splittedEssay = essay.split();
        # Iterating over each word
        for word in splittedEssay:
            # Checking if word is in glove words and set of words used by tfIdf essay vectorizer
            if (word in gloveWords) and (word in tfIdfEssayWords):
                # Tf-Idf value of particular word in essay
                tfIdfValueWord = tfIdfEssayDictionary[word] * (essay.count(word) /
len(splittedEssay));
                # Making tf-idf weighted word2vec
                tfIdfWeightedWord2VecEssayVector += tfIdfValueWord * gloveModel[word];
                # Summing tf-idf weight of word to cumulative sum
                cumulativeSumTfIdfWeightOfEssay += tfIdfValueWord;
        if cumulativeSumTfIdfWeightOfEssay != 0:
            # Taking average of sum of vectors with tf-idf cumulative sum
            tfIdfWeightedWord2VecEssayVector = tfIdfWeightedWord2VecEssayVector /
cumulativeSumTfIdfWeightOfEssay;
        # Appending the above calculated tf-idf weighted vector of particular essay to list of
vectors of essays
        tfIdfWeightedWord2VecEssaysVectors.append(tfIdfWeightedWord2VecEssayVector);
    return tfIdfWeightedWord2VecEssaysVectors;
```

In [279]:

```
def getAvgTfIdfTitleVectors(arrayOfTexts):
    # Creating list to save tf-idf weighted vectors of project titles
    tfIdfWeightedWord2VecTitlesVectors = [];
    # Iterating over each title
    for title in tqdm(arrayOfTexts):
        # Sum of tf-idf values of all words in a particular project title
        cumulativeSumTfIdfWeightOfTitle = 0;
        # Tf-Idf weighted word2vec vector of a particular project title
        tfIdfWeightedWord2VecTitleVector = np.zeros(300);
        # Splitting title into list of words
        splittedTitle = title.split();
        # Iterating over each word
        for word in splittedTitle:
            # Checking if word is in glove words and set of words used by tfIdf title vectorizer
            if (word in gloveWords) and (word in tfIdfTitleWords):
                # Tf-Idf value of particular word in title
                tfIdfValueWord = tfIdfTitleDictionary[word] * (title.count(word) /
len(splittedTitle));
                # Making tf-idf weighted word2vec
                tfIdfWeightedWord2VecTitleVector += tfIdfValueWord * gloveModel[word];
                # Summing tf-idf weight of word to cumulative sum
                cumulativeSumTfIdfWeightOfTitle += tfIdfValueWord;
        if cumulativeSumTfIdfWeightOfTitle != 0:
            # Taking average of sum of vectors with tf-idf cumulative sum
            tfIdfWeightedWord2VecTitleVector = tfIdfWeightedWord2VecTitleVector /
cumulativeSumTfIdfWeightOfTitle;
        # Appending the above calculated tf-idf weighted vector of particular title to list of
vectors of project titles
        tfIdfWeightedWord2VecTitlesVectors.append(tfIdfWeightedWord2VecTitleVector);
    return tfIdfWeightedWord2VecTitlesVectors;
```

In [118]:

```
kFoldResultsDataFrame = pd.DataFrame(columns =  ['Vectorizer', 'Model', 'Hyper Parameter - K', 'AUC
']);
kFoldResultsDataFrame
```

Out[118]:

	Vectorizer	Model	Hyper Parameter - K	AUC

Preparing Test data for analysis

In [281]:

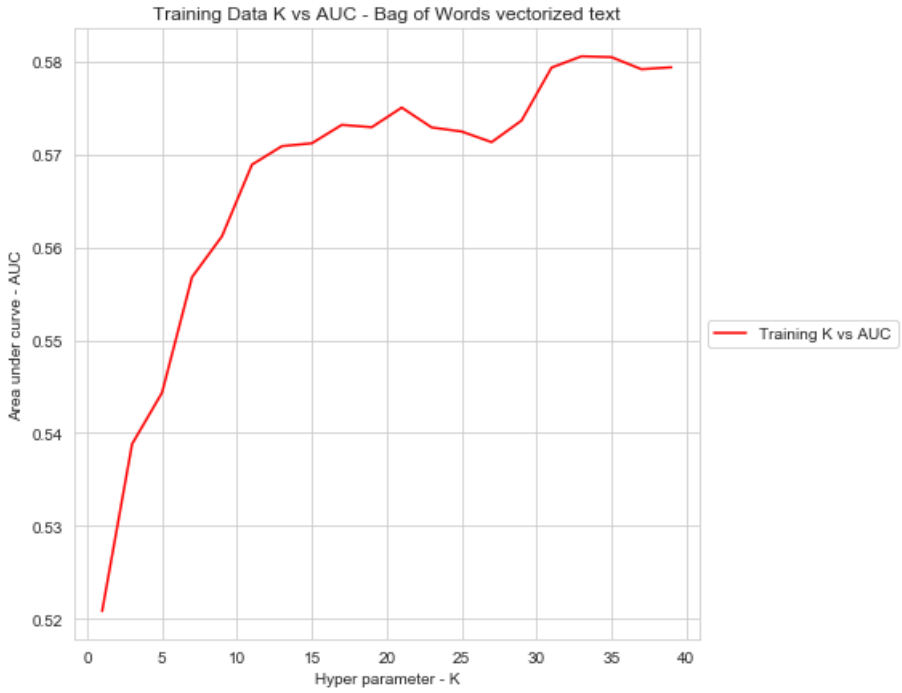
```
# Test data categorical features transformation
categoriesTransformedTestData =
subjectsCategoriesVectorizer.transform(testData['cleaned_categories']);
subCategoriesTransformedTestData =
subjectsSubCategoriesVectorizer.transform(testData['cleaned_sub_categories']);
teacherPrefixTransformedTestData = teacherPrefixVectorizer.transform(testData['teacher_prefix']);
schoolStateTransformedTestData = schoolStateVectorizer.transform(testData['school_state']);
projectGradeTransformedTestData =
projectGradeVectorizer.transform(testData['project_grade_category']);

# Test data text features transformation
preProcessedEssaysTemp = preProcessingWithAndWithoutStopWords(testData['project_essay'])[1];
preProcessedTitlesTemp = preProcessingWithAndWithoutStopWords(testData['project_title'])[1];
bowEssayTransformedTestData = bowEssayVectorizer.transform(preProcessedEssaysTemp);
bowTitleTransformedTestData = bowTitleVectorizer.transform(preProcessedTitlesTemp);
tfIdfEssayTransformedTestData = tfIdfEssayVectorizer.transform(preProcessedEssaysTemp);
tfIdfTitleTransformedTestData = tfIdfTitleVectorizer.transform(preProcessedTitlesTemp);
word2VecEssayTransformedTestData = getWord2VecVectors(preProcessedEssaysTemp);
word2VecTitleTransformedTestData = getWord2VecVectors(preProcessedTitlesTemp);
tfIdfWeightedEssayTransformedTestData = getAvgTfIdfEssayVectors(preProcessedEssaysTemp);
tfIdfWeightedTitleTransformedTestData = getAvgTfIdfTitleVectors(preProcessedTitlesTemp);

# Test data numerical features transformation
priceTransformedTestData = priceScaler.transform(testData['price'].values.reshape(-1, 1));
quantityTransformedTestData = quantityScaler.transform(testData['quantity'].values.reshape(-1, 1));
previouslyPostedTransformedTestData =
previouslyPostedScaler.transform(testData['teacher_number_of_previously_posted_projects'].values.r
eshape(-1, 1));
```

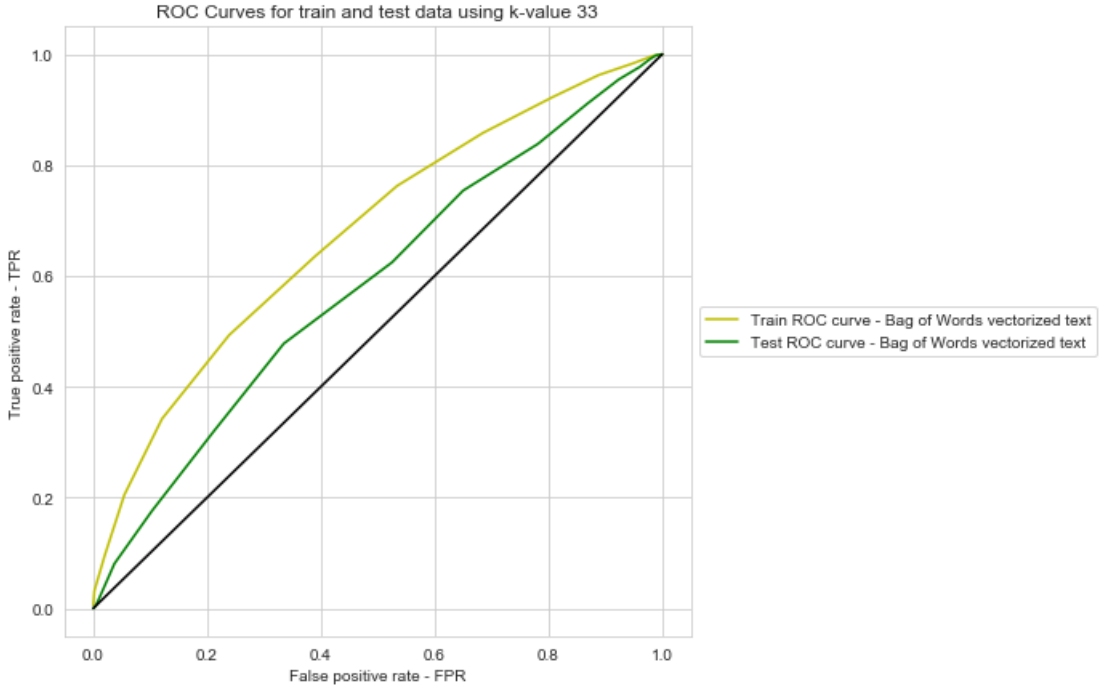
Analysis on imbalanced data using K-NN(k-fold cross validation)

In [124]:

```
testKValues = np.arange(1, 40, 2);
techniques = ['Bag of Words', 'Tf-Idf', 'Average Word2Vec', 'Tf-Idf Weighted Word2Vec'];
for index, technique in enumerate(techniques):
    areaUnderRocValuesTrain = [];
    trainingMergedData = hstack((categoriesVectorsSub,\
                                 subCategoriesVectorsSub,\
                                 teacherPrefixVectorsSub,\
                                 schoolStateVectorsSub,\
                                 projectGradeVectorsSub,\
                                 priceStandardizedSub,\
                                 previouslyPostedStandardizedSub));
    testMergedData = hstack((categoriesTransformedTestData,\
                                     subCategoriesTransformedTestData,\
                                     teacherPrefixTransformedTestData,\
                                     schoolStateTransformedTestData,\
                                     projectGradeTransformedTestData,\
                                     priceTransformedTestData,\
                                     previouslyPostedTransformedTestData));

    if(index == 0):
        trainingMergedData = hstack((trainingMergedData,\
                                     bowTitleModelSub,\
                                     bowEssayModelSub));
        testMergedData = hstack((testMergedData,\
                                 bowTitleTransformedTestData,\
                                 bowEssayTransformedTestData));
    elif(index == 1):
        trainingMergedData = hstack((trainingMergedData,\
                                     tfIdfTitleModelSub,\
                                     tfIdfEssayModelSub));
```

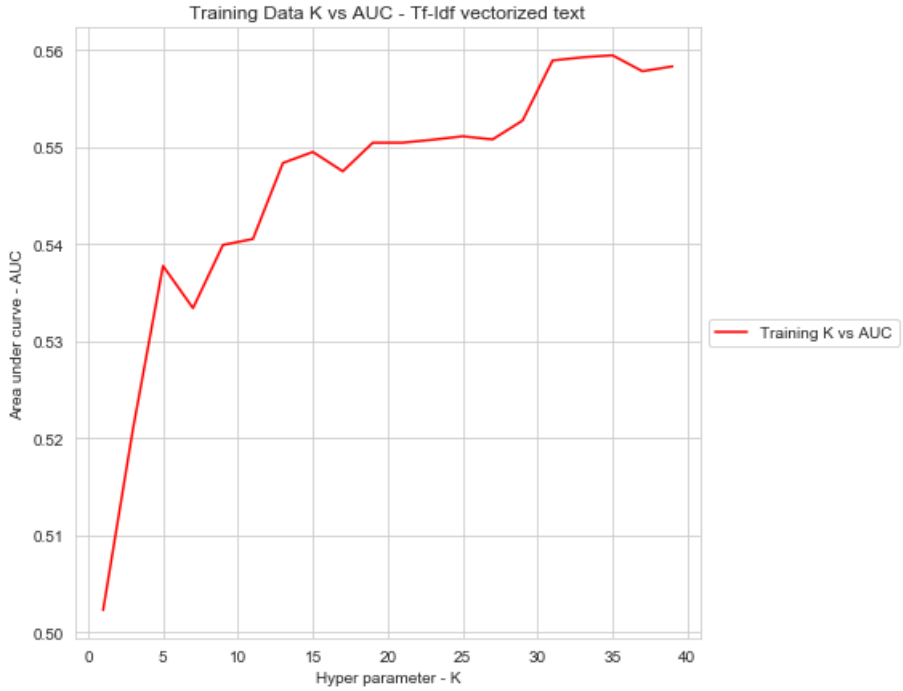
```
                                    tfIdfEssayModelSub));
        testMergedData = hstack((testMergedData,\
                                 tfIdfTitleTransformedTestData,\
                                 tfIdfEssayTransformedTestData));
    elif(index == 2):
        trainingMergedData = hstack((trainingMergedData,\
                                     word2VecTitlesVectorsSub,\
                                     word2VecEssaysVectorsSub));
        testMergedData = hstack((testMergedData,\
                                 word2VecTitleTransformedTestData,\
                                 word2VecEssayTransformedTestData));
    elif(index == 3):
        trainingMergedData = hstack((trainingMergedData,\
                                     tfIdfWeightedWord2VecTitlesVectorsSub,\
                                     tfIdfWeightedWord2VecEssaysVectorsSub));
        testMergedData = hstack((testMergedData,\
                                 tfIdfWeightedTitleTransformedTestData,\
                                 tfIdfWeightedEssayTransformedTestData));
    for testKValue in tqdm(testKValues):
        knnClassifier = KNeighborsClassifier(n_neighbors = testKValue, algorithm = 'brute');
        scores = cross_val_score(knnClassifier, trainingMergedData, classesTraining, cv = 5, scorin
g = 'roc_auc');
        areaUnderRocValuesTrain.append(np.array(scores).mean());

    plt.plot(testKValues, areaUnderRocValuesTrain, 'r', label = "Training K vs AUC");
    plt.title("Training Data K vs AUC - {} vectorized text".format(technique));
    plt.xlabel("Hyper parameter - K");
    plt.ylabel("Area under curve - AUC");
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.show();

    optimalKValue = testKValues[np.argmax(areaUnderRocValuesTrain)];
    knnClassifier = KNeighborsClassifier(n_neighbors = optimalKValue, algorithm = 'brute');
    knnClassifier.fit(trainingMergedData, classesTraining);
    predProbScoresTraining = knnClassifier.predict_proba(trainingMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTraining, predProbScoresTraining[:, 1]);
    predProbScoresTest = knnClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predProbScoresTest[:, 1]);
    areaUnderRocValueTest = auc(fprTest, tprTest);
    plt.plot(fprTrain, tprTrain, 'y', label="Train ROC curve - {} vectorized text".format(technique
));
    plt.plot(fprTest, tprTest, 'g', label="Test ROC curve - {} vectorized text".format(technique));
    plt.plot([0, 1], [0, 1], 'k-');
    plt.title("ROC Curves for train and test data using k-value {}".format(optimalKValue))
    plt.xlabel('False positive rate - FPR');
    plt.ylabel('True positive rate - TPR');
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.show();

    print("Results of analysis using {} vectorized text features merged with other features using
K-NN brute force algorithm:".format(technique));
    equalsBorder(70);
    print("AUC values of train data: ");
    equalsBorder(40);
    print(areaUnderRocValuesTrain);
    equalsBorder(40);
    print("Optimal K-Value: ", optimalKValue);
    equalsBorder(40);
    print("AUC value of test data: ", areaUnderRocValueTest);
    # Predicting classes of test data projects
    predictionClassesTest = knnClassifier.predict(testMergedData);
    equalsBorder(40);
    # Adding results to results dataframe
    kFoldResultsDataFrame = kFoldResultsDataFrame.append({'Vectorizer': technique, 'Model': 'Brute'
, 'Hyper Parameter - K': optimalKValue, 'AUC': areaUnderRocValueTest}, ignore_index = True);
    # Printing confusion matrix
    confusionMatrix = confusion_matrix(classesTest, predictionClassesTest);
    # Creating dataframe for generated confusion matrix
    confusionMatrixDataFrame = pd.DataFrame(data = confusionMatrix, index = ['Actual: NO', 'Actual:
YES'], columns = ['Predicted: NO', 'Predicted: YES']);
    print("Confusion Matrix : ");
    equalsBorder(60);
    print(confusionMatrixDataFrame);
    equalsBorder(110);
    equalsBorder(110);
    equalsBorder(110);
```

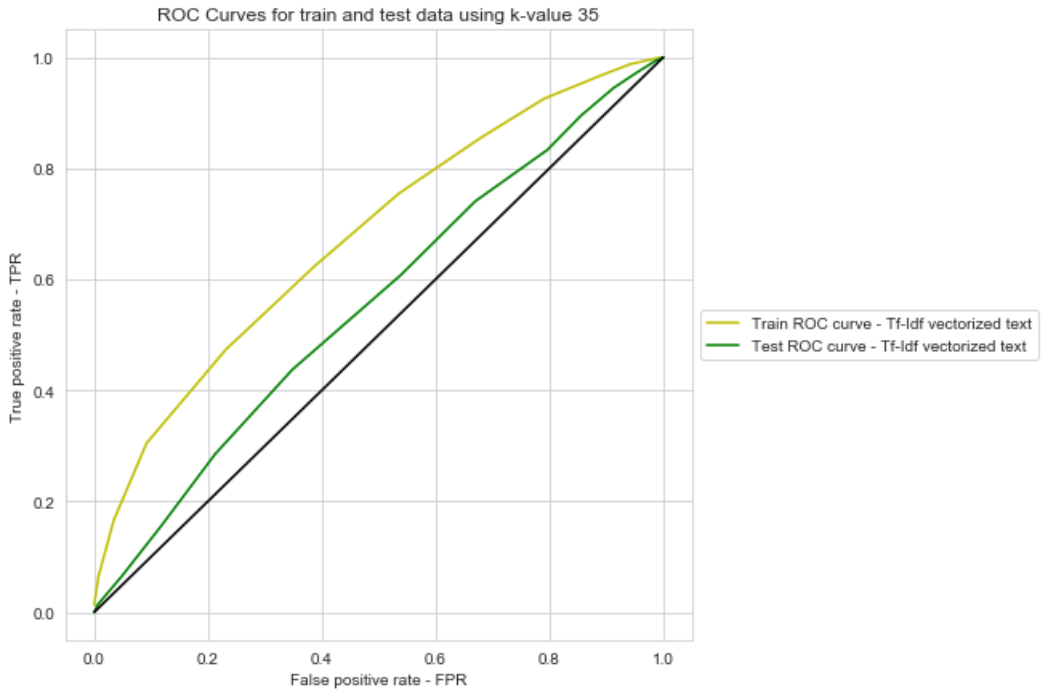





```
Results of analysis using Bag of Words vectorized text features merged with other features using K
-NN brute force algorithm:
========================================================================
AUC values of train data:
========================================
[0.5208231778438155, 0.5388207141106124, 0.5443554856988919, 0.5567904380803824,
0.5611947320020753, 0.5689025654493272, 0.5708951863303491, 0.5712055873582917,
0.5731905803803545, 0.5729348803736909, 0.5750516547049701, 0.5729101695618276,
0.5724710285762991, 0.571328572164729, 0.5736583119248623, 0.5793537015023535, 0.580574158741386,
0.5804823414601682, 0.5791816629509718, 0.5793890388519048]
========================================
Optimal K-Value:  33
========================================
AUC value of test data:  0.58369996669968
========================================
Confusion Matrix :
============================================================
             Predicted: NO  Predicted: YES
Actual: NO               1             456
Actual: YES              0            2543
============================================================================================================
```

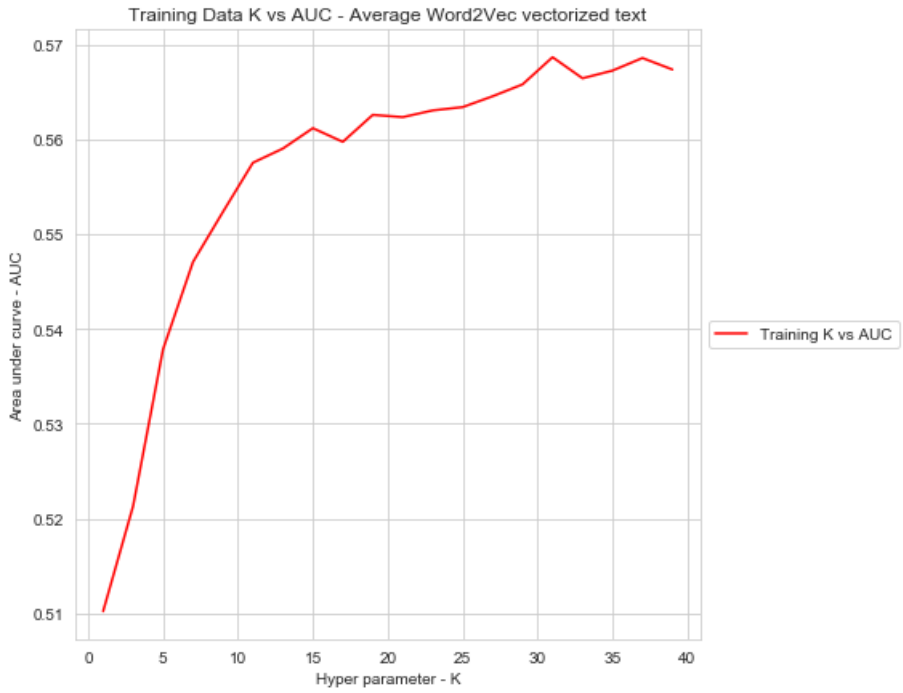
```
=======
==========================================================================================================
=======
==========================================================================================================
=======
```

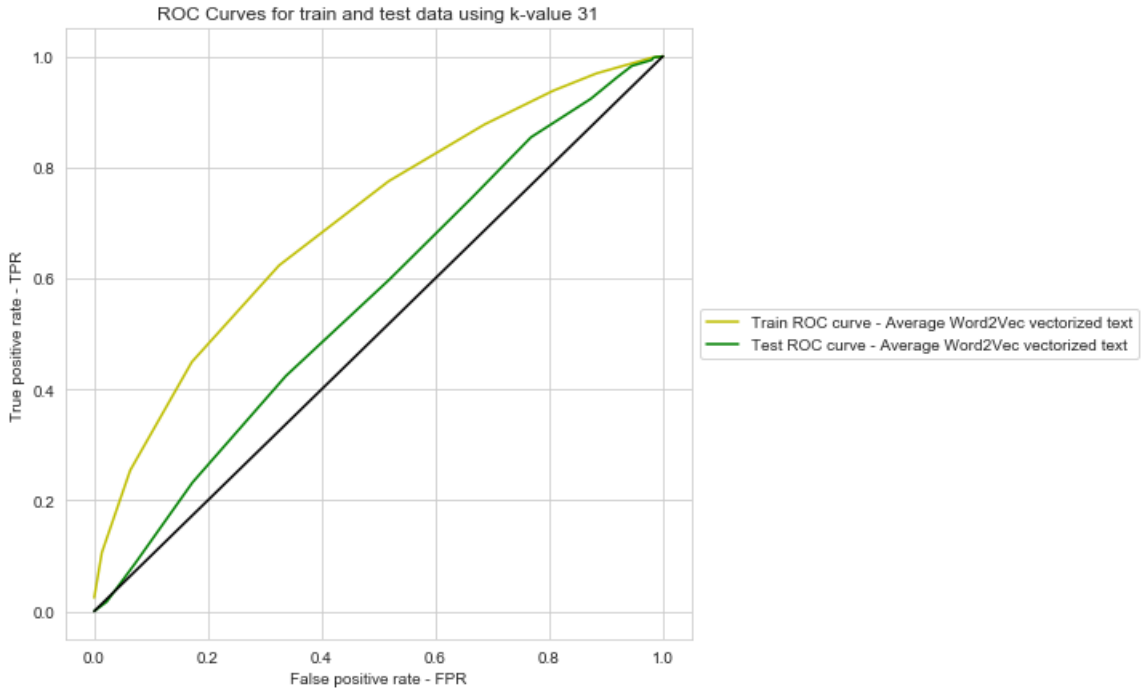





```
Results of analysis using Tf-Idf vectorized text features merged with other features using K-NN br
ute force algorithm:
========================================================================
AUC values of train data:
========================================
[0.502268737913808, 0.5210035911725254, 0.5377502551155098, 0.5333942597761746,
0.5399143246196763, 0.5405209848733368, 0.5483523952589064, 0.5494933934549823,
0.5474954442036795, 0.5504434876457428, 0.5504489280129469, 0.5507520605053919,
0.5511069359522345, 0.5507867210513148, 0.5527310878117206, 0.5589202442517875,
0.5592470560436937, 0.5594561583305314, 0.5578093194591379, 0.5583108324514976]
========================================
Optimal K-Value:  35
========================================
AUC value of test data:  0.5554028693345356
========================================
```

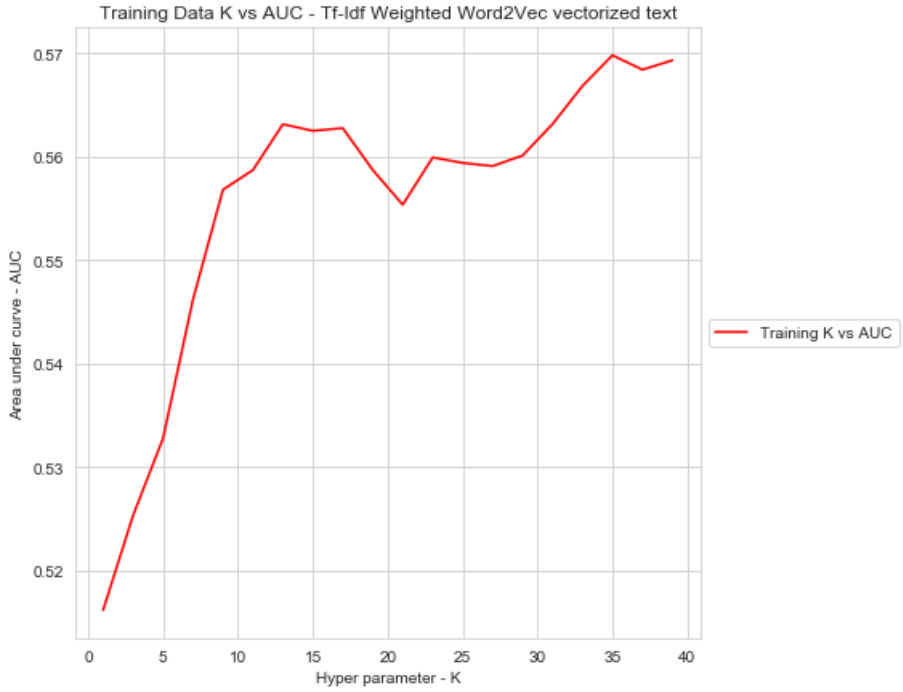
```
Confusion Matrix :
============================================================
              Predicted: NO  Predicted: YES
Actual: NO                0             457
Actual: YES               0            2543
============================================================================================================
=======
============================================================================================================
=======
============================================================================================================
=======
```

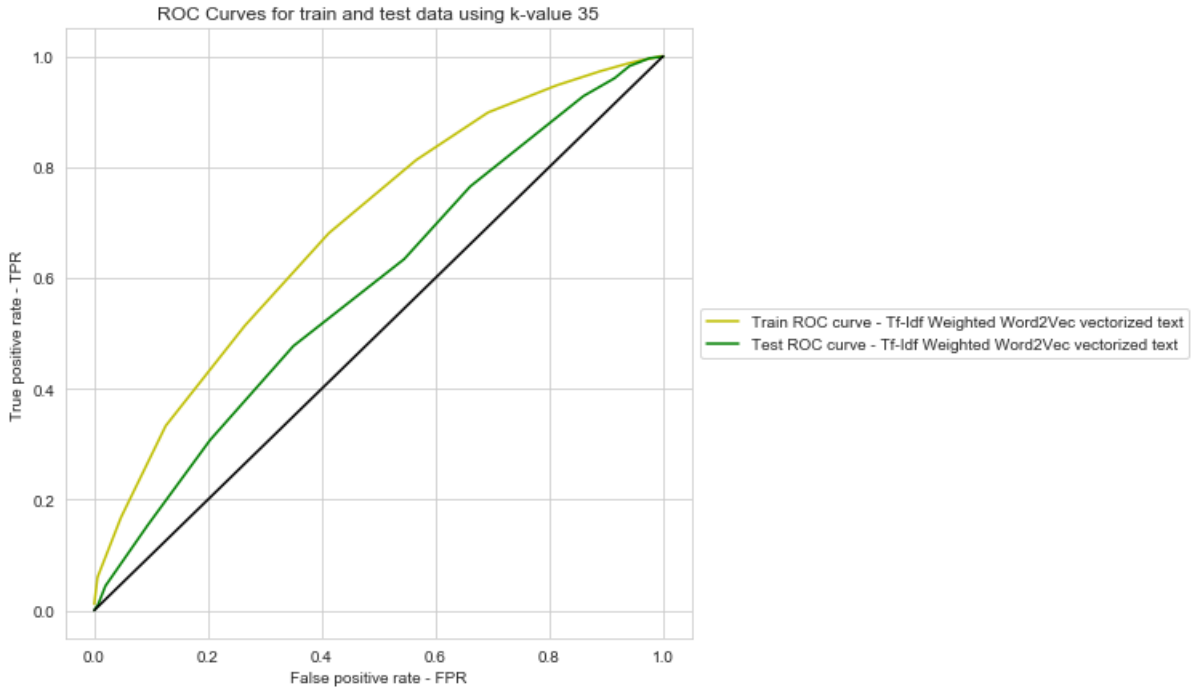





```
Results of analysis using Average Word2Vec vectorized text features merged with other features usi
ng K-NN brute force algorithm:
========================================================================
AUC values of train data:
========================================
[0.5102137153960281, 0.5213048840247699, 0.537847741333578, 0.547059981902272, 0.5523469198548131,
0.5575288556883619, 0.5590231271530748, 0.5611719450355064, 0.5597297396959424,
0.5625774286968841, 0.5623372962884595, 0.5630452459760734, 0.5633931409943183,
0.5645367885916814, 0.5657917551419027, 0.5686589944761107, 0.5664332106229713,
```

```
0.5672437680935047, 0.5685717848701585, 0.567360413524103]
========================================
Optimal K-Value:  31
========================================
AUC value of test data:  0.562602880348595
========================================
Confusion Matrix :
============================================================
             Predicted: NO  Predicted: YES
Actual: NO               1             456
Actual: YES              1            2542
============================================================================================================
=======
============================================================================================================
=======
============================================================================================================
=======
```






```
Results of analysis using Tf-Idf Weighted Word2Vec vectorized text features merged with other feat
ures using K-NN brute force algorithm:
--------------------------------------------------------------------
```

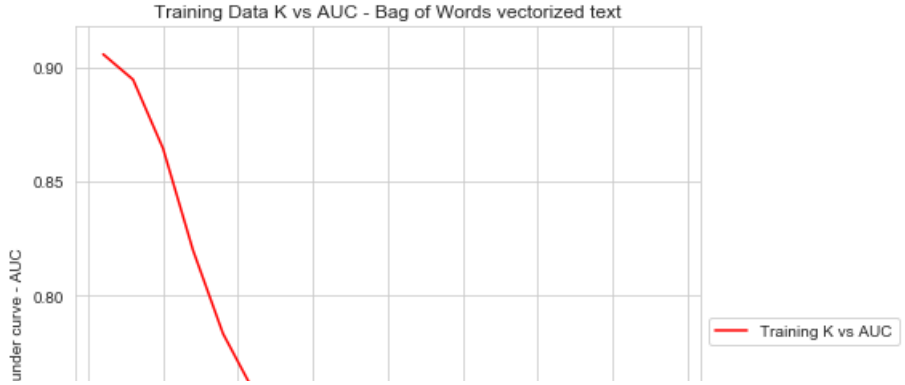
```
========================================================================
AUC values of train data:
========================================
[0.5161888229147853, 0.5253361637812766, 0.5327815047899539, 0.5461971272610854,
0.5568005172331086, 0.5586970538538429, 0.5631244467002008, 0.5624858586903544,
0.5627442960495929, 0.5587017307105413, 0.5553460986679661, 0.559916556306143, 0.5593849935152642,
0.5590823391356098, 0.5600997780095505, 0.5631378829218097, 0.5668231635968736,
0.5697832874755709, 0.5683904920933875, 0.5692970434748391]
========================================
Optimal K-Value:  35
========================================
AUC value of test data:  0.5840665283599119
========================================
Confusion Matrix :
============================================================
             Predicted: NO  Predicted: YES
Actual: NO               0             457
Actual: YES              0            2543
============================================================================================================
=======
============================================================================================================
=======
============================================================================================================
=======
```

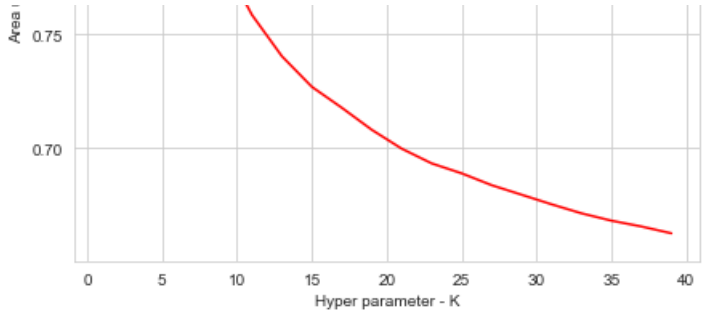
Analysis on balanced data using K-NN(k-fold cross validation)

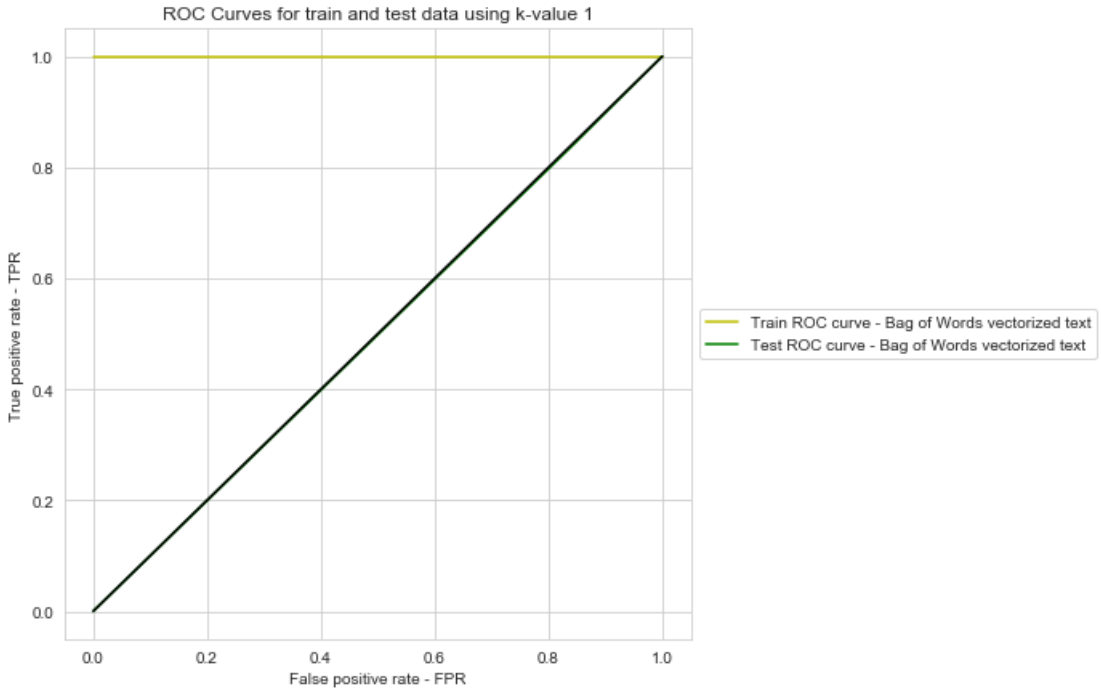
In [194]:

```
testKValues = np.arange(1, 40, 2);
techniques = ['Bag of Words', 'Tf-Idf', 'Average Word2Vec', 'Tf-Idf Weighted Word2Vec'];
for index, technique in enumerate(techniques):
    areaUnderRocValuesTrain = [];
    trainingMergedData = hstack((categoriesVectorsSub,\
                                 subCategoriesVectorsSub,\
                                 teacherPrefixVectorsSub,\
                                 schoolStateVectorsSub,\
                                 projectGradeVectorsSub,\
                                 priceStandardizedSub,\
                                 previouslyPostedStandardizedSub));
    testMergedData = hstack((categoriesTransformedTestData,\
                                      subCategoriesTransformedTestData,\
                                      teacherPrefixTransformedTestData,\
                                      schoolStateTransformedTestData,\
                                      projectGradeTransformedTestData,\
                                      priceTransformedTestData,\
                                      previouslyPostedTransformedTestData));

    if(index == 0):
        trainingMergedData = hstack((trainingMergedData,\
                                     bowTitleModelSub,\
                                     bowEssayModelSub));
        testMergedData = hstack((testMergedData,\
                                 bowTitleTransformedTestData,\
                                 bowEssayTransformedTestData));
    elif(index == 1):
        trainingMergedData = hstack((trainingMergedData,\
                                     tfIdfTitleModelSub,\
                                     tfIdfEssayModelSub));
        testMergedData = hstack((testMergedData,\
                                 tfIdfTitleTransformedTestData,\
                                 tfIdfEssayTransformedTestData));
    elif(index == 2):
        trainingMergedData = hstack((trainingMergedData,\
                                     word2VecTitlesVectorsSub,\
                                     word2VecEssaysVectorsSub));
        testMergedData = hstack((testMergedData,\
                                 word2VecTitleTransformedTestData,\
                                 word2VecEssayTransformedTestData));
    elif(index == 3):
        trainingMergedData = hstack((trainingMergedData,\
                                     tfIdfWeightedWord2VecTitlesVectorsSub,\
                                     tfIdfWeightedWord2VecEssaysVectorsSub));
        testMergedData = hstack((testMergedData,\
```

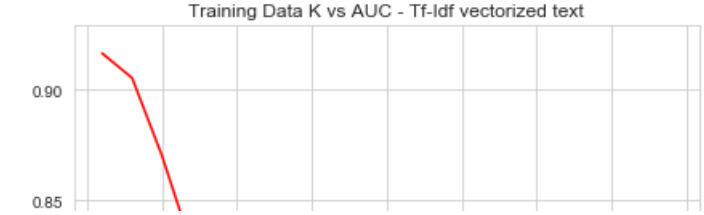
```
                                tfIdfWeightedTitleTransformedTestData,\
                                tfIdfWeightedEssayTransformedTestData));
    for testKValue in tqdm(testKValues):
        knnClassifier = KNeighborsClassifier(n_neighbors = testKValue, algorithm = 'brute');
        scores = cross_val_score(knnClassifier, trainingMergedData, classesTraining, cv = 5, scorin
g = 'roc_auc');
        areaUnderRocValuesTrain.append(np.array(scores).mean());

    plt.plot(testKValues, areaUnderRocValuesTrain, 'r', label = "Training K vs AUC");
    plt.title("Training Data K vs AUC - {} vectorized text".format(technique));
    plt.xlabel("Hyper parameter - K");
    plt.ylabel("Area under curve - AUC");
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.show();

    optimalKValue = testKValues[np.argmax(areaUnderRocValuesTrain)];
    knnClassifier = KNeighborsClassifier(n_neighbors = optimalKValue, algorithm = 'brute');
    knnClassifier.fit(trainingMergedData, classesTraining);
    predProbScoresTraining = knnClassifier.predict_proba(trainingMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTraining, predProbScoresTraining[:, 1]);
    predProbScoresTest = knnClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predProbScoresTest[:, 1]);
    areaUnderRocValueTest = auc(fprTest, tprTest);
    plt.plot(fprTrain, tprTrain, 'y', label="Train ROC curve - {} vectorized text".format(technique
));
    plt.plot(fprTest, tprTest, 'g', label="Test ROC curve - {} vectorized text".format(technique));
    plt.plot([0, 1], [0, 1], 'k-');
    plt.title("ROC Curves for train and test data using k-value {}".format(optimalKValue))
    plt.xlabel('False positive rate - FPR');
    plt.ylabel('True positive rate - TPR');
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.show();

    print("Results of analysis using {} vectorized text features merged with other features using
K-NN brute force algorithm:".format(technique));
    equalsBorder(70);
    print("AUC values of train data: ");
    equalsBorder(40);
    print(areaUnderRocValuesTrain);
    equalsBorder(40);
    print("Optimal K-Value: ", optimalKValue);
    equalsBorder(40);
    print("AUC value of test data: ", areaUnderRocValueTest);
    # Predicting classes of test data projects
    predictionClassesTest = knnClassifier.predict(testMergedData);
    equalsBorder(40);
    # Adding results to results dataframe
    kFoldResultsDataFrame = kFoldResultsDataFrame.append({'Vectorizer': technique, 'Model': 'Brute'
, 'Hyper Parameter - K': optimalKValue, 'AUC': areaUnderRocValueTest}, ignore_index = True);
    # Printing confusion matrix
    confusionMatrix = confusion_matrix(classesTest, predictionClassesTest);
    # Creating dataframe for generated confusion matrix
    confusionMatrixDataFrame = pd.DataFrame(data = confusionMatrix, index = ['Actual: NO', 'Actual:
YES'], columns = ['Predicted: NO', 'Predicted: YES']);
    print("Confusion Matrix : ");
    equalsBorder(60);
    print(confusionMatrixDataFrame);
    equalsBorder(110);
    equalsBorder(110);
    equalsBorder(110);
```

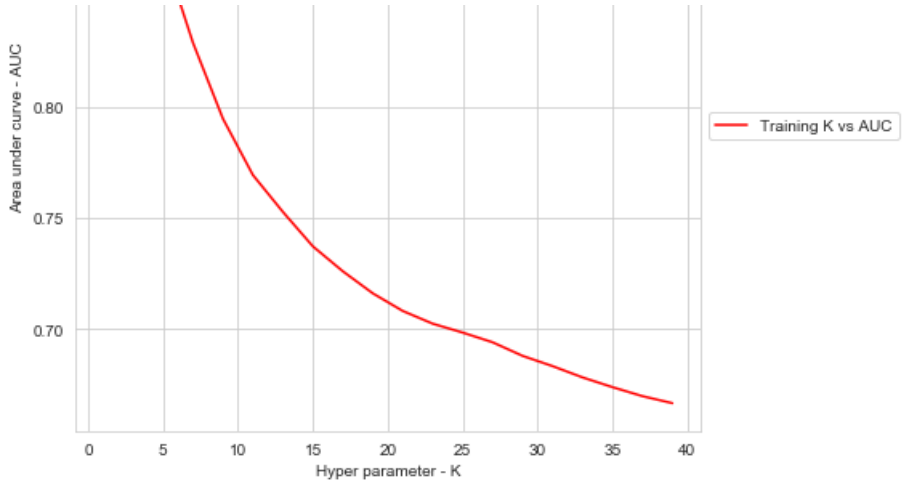


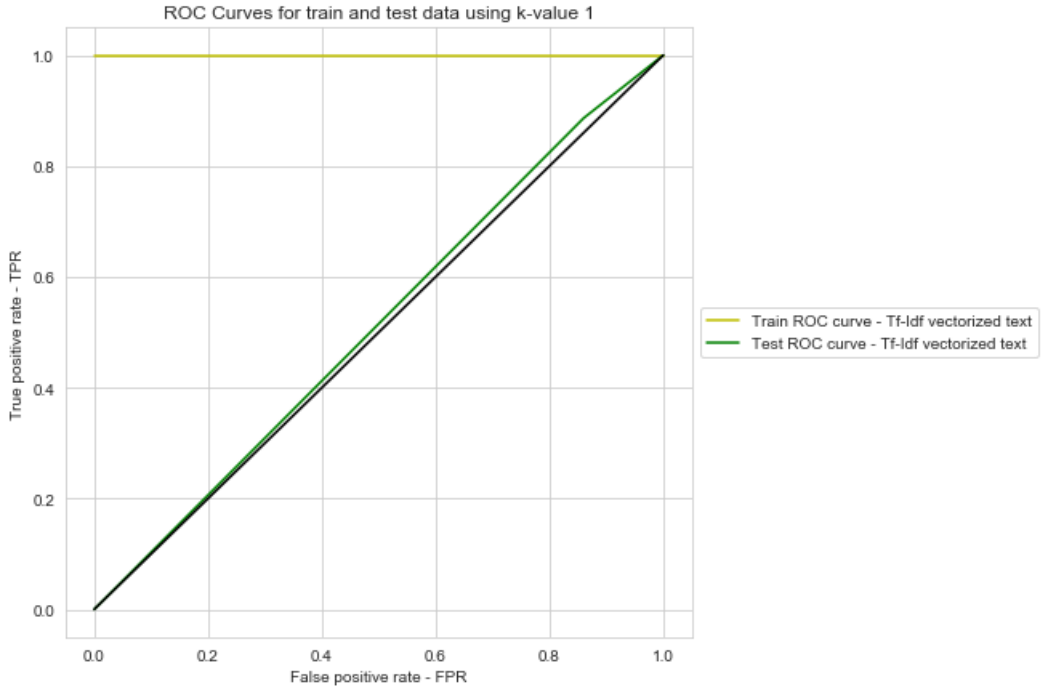




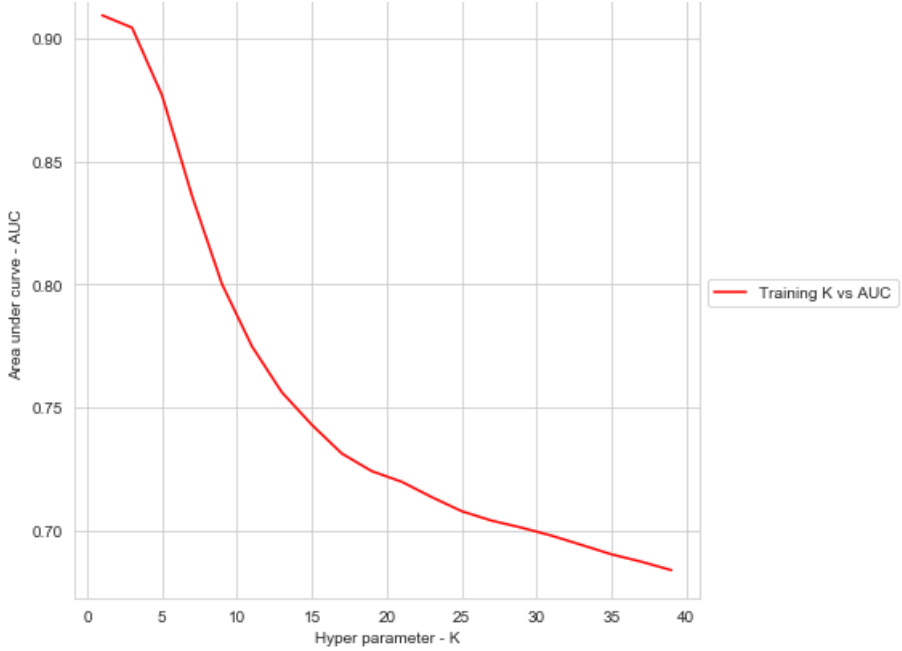

```
Results of analysis using Bag of Words vectorized text features merged with other features using K
-NN brute force algorithm:
========================================================================
AUC values of train data:
========================================
[0.9057420305535363, 0.8945913265518728, 0.8643774256788769, 0.8197352055462597,
0.7831827453591671, 0.758477792194014, 0.7403199541524053, 0.7268958883622098, 0.7177136528989003,
0.7080414033713262, 0.6997465178960134, 0.6932946817988367, 0.6889350859650298,
0.6837457930690102, 0.679552227469989, 0.6753509086931728, 0.6713878555117908, 0.668164493494084,
0.6656132376613825, 0.6626255980811904]
========================================
Optimal K-Value:  1
========================================
AUC value of test data:  0.49852902075547845
========================================
Confusion Matrix :
============================================================
             Predicted: NO  Predicted: YES
Actual: NO              70             387
Actual: YES            397            2146
============================================================================================================
=======
============================================================================================================
=======
============================================================================================================
=======
```

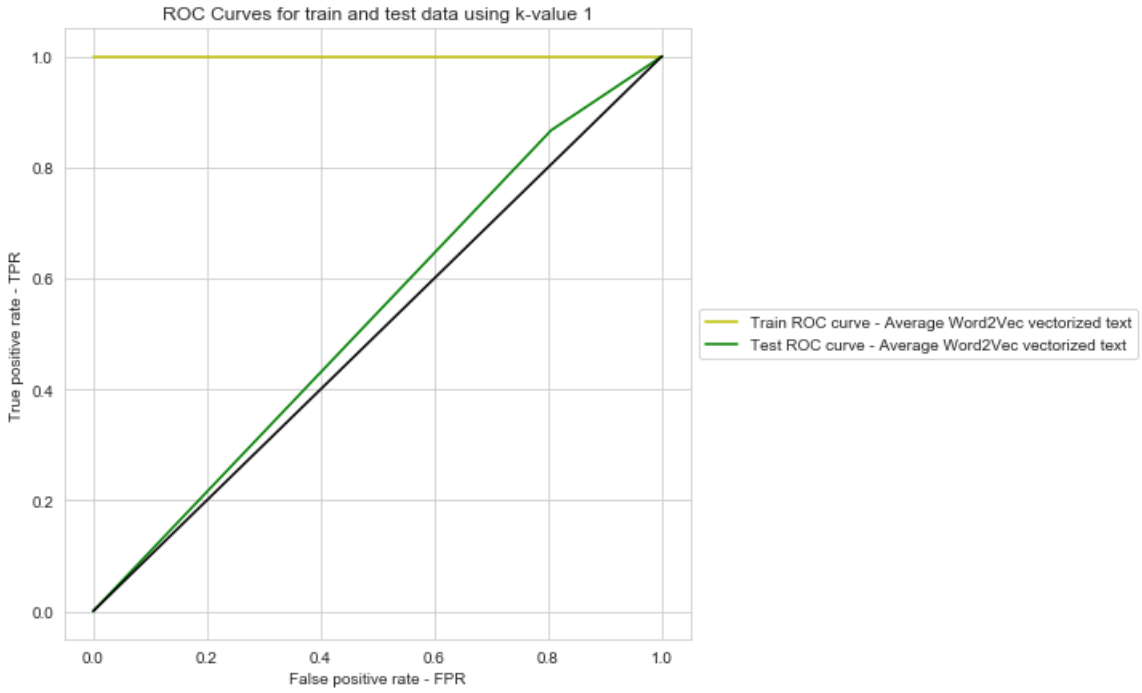






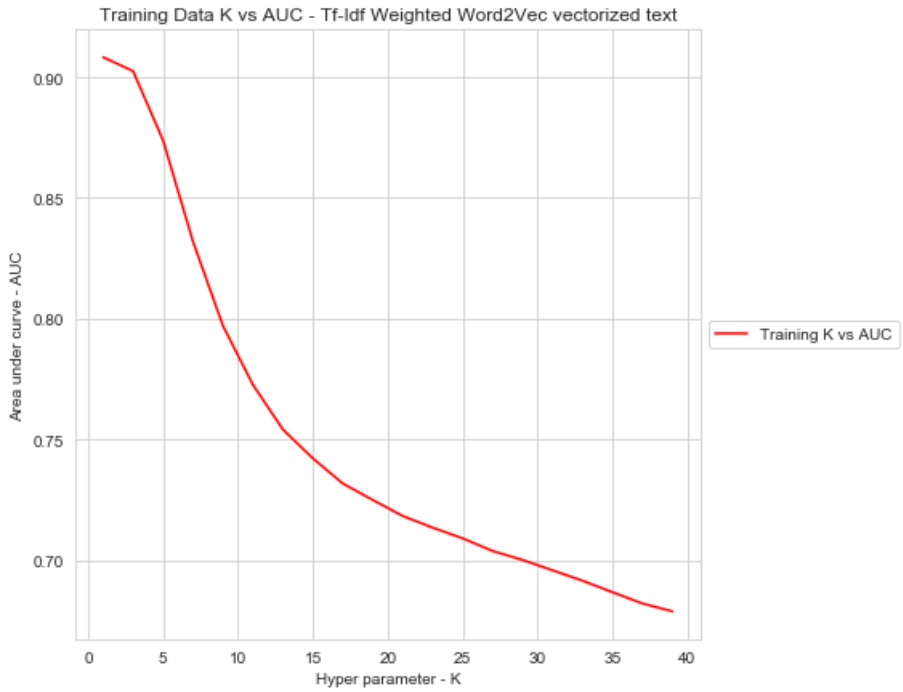

```
Results of analysis using Tf-Idf vectorized text features merged with other features using K-NN br
ute force algorithm:
========================================================================
AUC values of train data:
=========================================
[0.9162314252869678, 0.9051465726945087, 0.8700486476341652, 0.8286864275417989,
0.7945284683543399, 0.7691953017653705, 0.7526217878192457, 0.7371226914526057,
0.7259029424111647, 0.7159810784194411, 0.7080471371772863, 0.7023211488408301,
0.6983180203690182, 0.6939487220959272, 0.6878382399601758, 0.6832277363125655,
0.6781525569231107, 0.6737646368461715, 0.6697149551948767, 0.6665500341435163]
=========================================
Optimal K-Value:  1
=========================================
AUC value of test data:  0.5131992314251763
=========================================
Confusion Matrix :
============================================================
             Predicted: NO  Predicted: YES
Actual: NO              64             393
Actual: YES            289            2254
============================================================================================================
=======
============================================================================================================
=======
============================================================================================================
=======
```

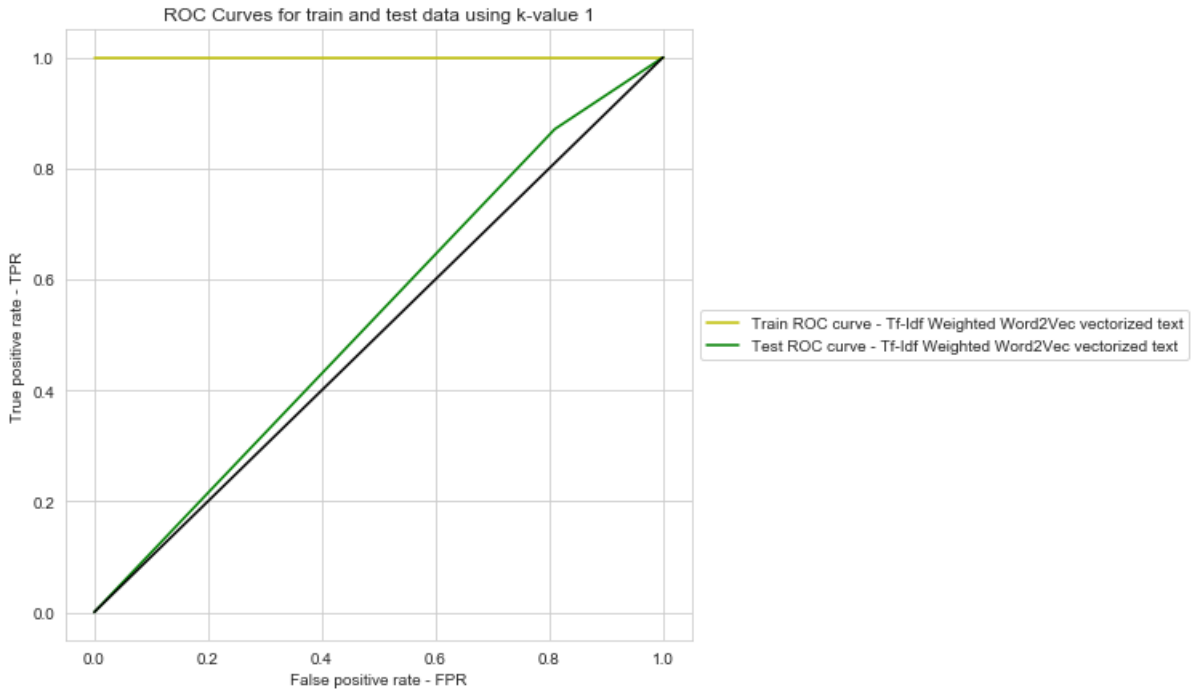





```
Results of analysis using Average Word2Vec vectorized text features merged with other features usi
ng K-NN brute force algorithm:
========================================================================
AUC values of train data:
========================================
[0.9093497652978998, 0.9043068534254385, 0.8767541992552405, 0.8358918803789022,
0.800026322373629, 0.7747237890205636, 0.7560104333549655, 0.7428012794214311, 0.7311305254759695,
0.7239627091724085, 0.7196395662460556, 0.7134192974339 14, 0.7076839620746282, 0.7038409688791923,
0.700988602535348, 0.6977323048810804, 0.6939701123843026, 0.6901406341038436, 0.6871256170518121,
0.6836911609735372]
========================================
Optimal K-Value:  1
========================================
AUC value of test data:  0.5307206206422401
========================================
Confusion Matrix :
============================================================
             Predicted: NO  Predicted: YES
Actual: NO              89             368
Actual: YES            339            2204
============================================================================================================
=======
============================================================================================================
=======
```

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






```
Results of analysis using Tf-Idf Weighted Word2Vec vectorized text features merged with other feat
ures using K-NN brute force algorithm:
========================================================================
AUC values of train data:
========================================
[0.908259020393443, 0.902518511637626, 0.873836129148551, 0.8319427385855086, 0.7970860718982278,
0.772702425590641, 0.7542135496685562, 0.7422664575997266, 0.7318550687138528, 0.7250536979958738,
0.7184275714913131, 0.7135910863600886, 0.7091812336767213, 0.7039484956887567,
0.7002417364824247, 0.6959596335302047, 0.6916482733087009, 0.6868862403662254,
0.6822305545686868, 0.678981570167769]
========================================
Optimal K-Value:  1
========================================
AUC value of test data:  0.5304986185099871
========================================
Confusion Matrix :
============================================================
             Predicted: NO  Predicted: YES
```

```
Actual: NO               87             370
Actual: YES             329            2214
================================================================================================================
=======
================================================================================================================
=======
================================================================================================================
=======
```

Selecting top 2000 important features from data with Tf-Idf vectorized text

In [282]:

```
trainingMergedData = hstack((categoriesVectorsSub,\
                             subCategoriesVectorsSub,\
                             teacherPrefixVectorsSub,\
                             schoolStateVectorsSub,\
                             projectGradeVectorsSub,\
                             priceStandardizedSub,\
                             previouslyPostedStandardizedSub));
testMergedData = hstack((categoriesTransformedTestData,\
                                  subCategoriesTransformedTestData,\
                                  teacherPrefixTransformedTestData,\
                                  schoolStateTransformedTestData,\
                                  projectGradeTransformedTestData,\
                                  priceTransformedTestData,\
                                  previouslyPostedTransformedTestData));
trainingMergedData = hstack((trainingMergedData,\
                            tfIdfTitleModelSub,\
                            tfIdfEssayModelSub));
testMergedData = hstack((testMergedData,\
                         tfIdfTitleTransformedTestData,\
                         tfIdfEssayTransformedTestData));
```

In [283]:

```
print("Training data shape: ", trainingMergedData.shape);
print("Test data shape: ", testMergedData.shape);
print("Classes Training shape: ", classesTraining.shape);
```

```
Training data shape:  (7000, 5671)
Test data shape:  (3000, 5671)
Classes Training shape:  (7000,)
```

In [284]:

```
selectKBest = SelectKBest(f_classif, k = 2000);
filteredFeaturesTrainingMergedData = selectKBest.fit_transform(trainingMergedData, classesTraining
);
filteredFeaturesTrainingMergedData.shape
```

Out[284]:

```
(7000, 2000)
```

In [219]:

```
selectedFeaturesResultsDataFrame = pd.DataFrame(columns =  ['Vectorizer', 'Model', 'Hyper
Parameter - K', 'AUC']);
selectedFeaturesResultsDataFrame
```

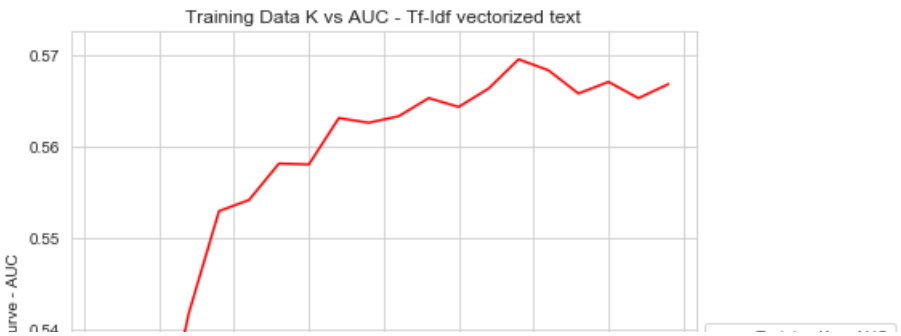
Out[219]:

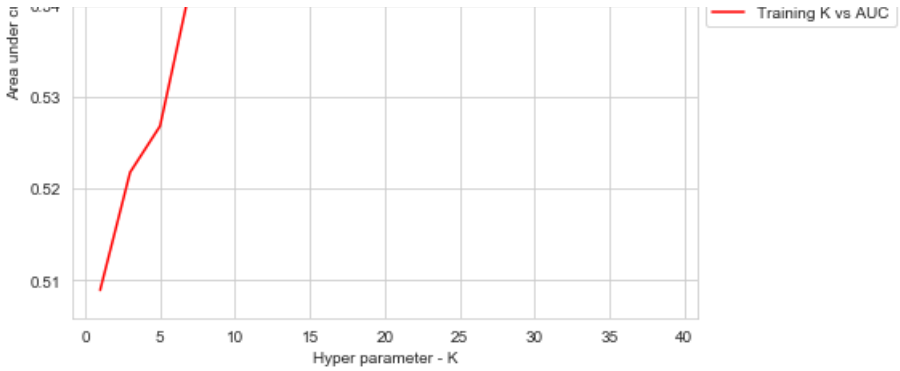
	Vectorizer	Model	Hyper Parameter - K	AUC

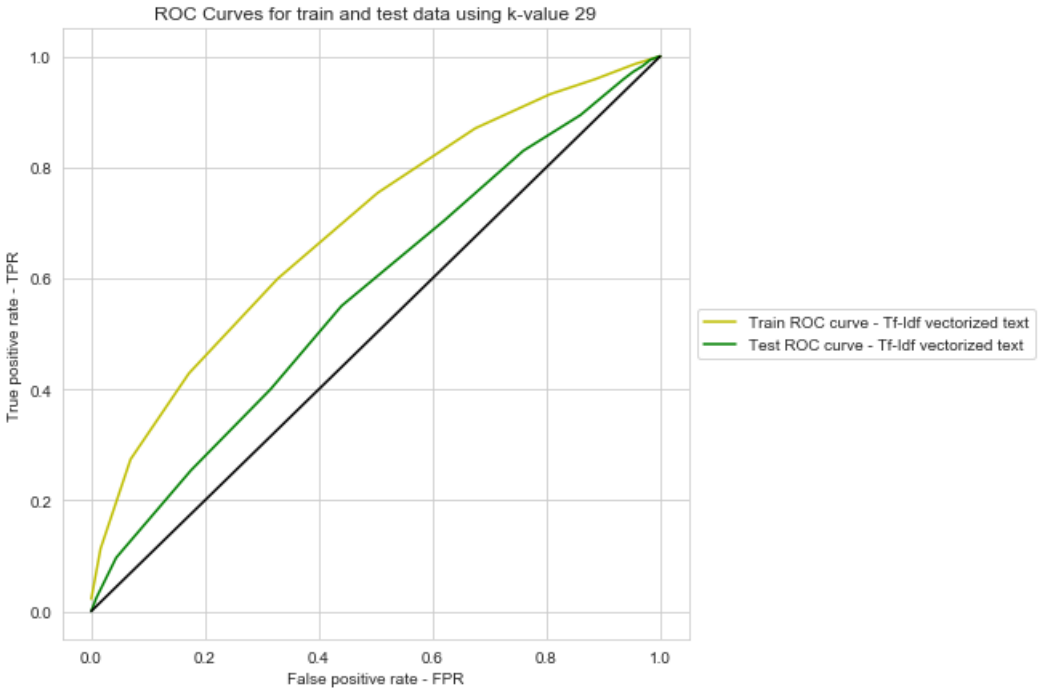
Analysis on imbalanced data using top 2000 features of data & K-NN(k-fold cross validation)

In [285]:

```
testKValues = np.arange(1, 40, 2);
areaUnderRocValuesTrain = [];
for testKValue in tqdm(testKValues):
    knnClassifier = KNeighborsClassifier(n_neighbors = testKValue, algorithm = 'brute');
    scores = cross_val_score(knnClassifier, filteredFeaturesTrainingMergedData, classesTraining, cv
= 10, scoring = 'roc_auc');
    areaUnderRocValuesTrain.append(np.array(scores).mean());

plt.plot(testKValues, areaUnderRocValuesTrain, 'r', label = "Training K vs AUC");
plt.title("Training Data K vs AUC - {} vectorized text".format("Tf-Idf"));
plt.xlabel("Hyper parameter - K");
plt.ylabel("Area under curve - AUC");
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show();

optimalKValue = testKValues[np.argmax(areaUnderRocValuesTrain)];
knnClassifier = KNeighborsClassifier(n_neighbors = optimalKValue, algorithm = 'brute');
knnClassifier.fit(filteredFeaturesTrainingMergedData, classesTraining);
predProbScoresTraining = knnClassifier.predict_proba(filteredFeaturesTrainingMergedData);
fprTrain, tprTrain, thresholdTrain = roc_curve(classesTraining, predProbScoresTraining[:, 1]);
filteredFeaturesTestMergedData = selectKBest.transform(testMergedData);
predProbScoresTest = knnClassifier.predict_proba(filteredFeaturesTestMergedData);
fprTest, tprTest, thresholdTest = roc_curve(classesTest, predProbScoresTest[:, 1]);
areaUnderRocValueTest = auc(fprTest, tprTest);
plt.plot(fprTrain, tprTrain, 'y', label="Train ROC curve - {} vectorized text".format("Tf-Idf"));
plt.plot(fprTest, tprTest, 'g', label="Test ROC curve - {} vectorized text".format("Tf-Idf"));
plt.plot([0, 1], [0, 1], 'k-');
plt.title("ROC Curves for train and test data using k-value {}".format(optimalKValue))
plt.xlabel('False positive rate - FPR');
plt.ylabel('True positive rate - TPR');
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show();

print("Results of analysis using {} vectorized text features merged with other features using K-NN
brute force algorithm:".format("Tf-Idf"));
equalsBorder(70);
print("AUC values of train data: ");
equalsBorder(40);
print(areaUnderRocValuesTrain);
equalsBorder(40);
print("Optimal K-Value: ", optimalKValue);
equalsBorder(40);
print("AUC value of test data: ", areaUnderRocValueTest);
# Predicting classes of test data projects
predictionClassesTest = knnClassifier.predict(filteredFeaturesTestMergedData);
equalsBorder(40);
# Adding results to results dataframe
selectedFeaturesResultsDataFrame = selectedFeaturesResultsDataFrame.append({'Vectorizer': "Tf-Idf"
, 'Model': 'Brute', 'Hyper Parameter - K': optimalKValue, 'AUC': areaUnderRocValueTest}, ignore_ind
ex = True);
# Printing confusion matrix
confusionMatrix = confusion_matrix(classesTest, predictionClassesTest);
# Creating dataframe for generated confusion matrix
confusionMatrixDataFrame = pd.DataFrame(data = confusionMatrix, index = ['Actual: NO', 'Actual: YES
'], columns = ['Predicted: NO', 'Predicted: YES']);
print("Confusion Matrix : ");
equalsBorder(60);
confusionMatrixDataFrame
```








```
Results of analysis using Tf-Idf vectorized text features merged with other features using K-NN br
ute force algorithm:
========================================================================
AUC values of train data:
=========================================
[0.5088720627641867, 0.5217475405236953, 0.5268075645978578, 0.5418624048096264,
0.5529243098246011, 0.5541331755985821, 0.5581202558462723, 0.5580210647113776,
0.5630920439936429, 0.5625807992918386, 0.5632949191550652, 0.5652738508284056,
0.5643100379026158, 0.5663234598175874, 0.5695147297432138, 0.5683041363163748,
0.5657703420165175, 0.5670452133486332, 0.5652631279021676, 0.566803790243511]
=========================================
Optimal K-Value:  29
=========================================
AUC value of test data:  0.5700468355661183
=========================================
Confusion Matrix :
============================================================
```

Out[285]:

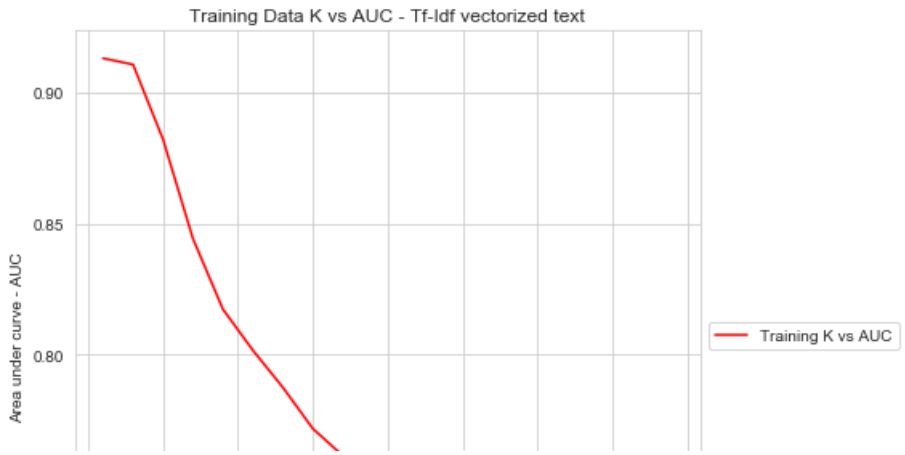
	Predicted: NO	Predicted: YES
Actual: NO	1	456
Actual: YES	0	2543

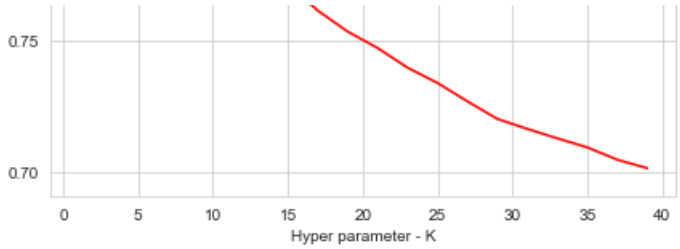
Analysis on balanced data using top 2000 features of data & K-NN(k-fold cross validation)

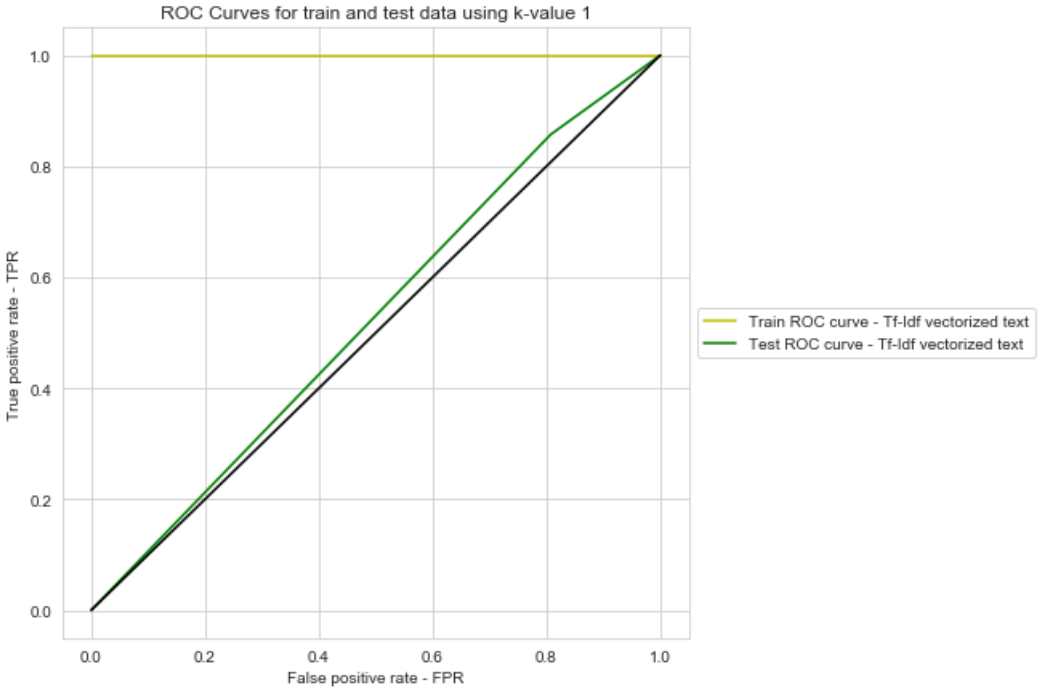
In [223]:

```
testKValues = np.arange(1, 40, 2);
```

```
testKValues = np.arange(1, 40, 2);
areaUnderRocValuesTrain = [];
for testKValue in tqdm(testKValues):
    knnClassifier = KNeighborsClassifier(n_neighbors = testKValue, algorithm = 'brute');
    scores = cross_val_score(knnClassifier, filteredFeaturesTrainingMergedData, classesTraining, cv
= 10, scoring = 'roc_auc');
    areaUnderRocValuesTrain.append(np.array(scores).mean());

plt.plot(testKValues, areaUnderRocValuesTrain, 'r', label = "Training K vs AUC");
plt.title("Training Data K vs AUC - {} vectorized text".format("Tf-Idf"));
plt.xlabel("Hyper parameter - K");
plt.ylabel("Area under curve - AUC");
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show();

optimalKValue = testKValues[np.argmax(areaUnderRocValuesTrain)];
knnClassifier = KNeighborsClassifier(n_neighbors = optimalKValue, algorithm = 'brute');
knnClassifier.fit(filteredFeaturesTrainingMergedData, classesTraining);
predProbScoresTraining = knnClassifier.predict_proba(filteredFeaturesTrainingMergedData);
fprTrain, tprTrain, thresholdTrain = roc_curve(classesTraining, predProbScoresTraining[:, 1]);
filteredFeaturesTestMergedData = selectKBest.transform(testMergedData);
predProbScoresTest = knnClassifier.predict_proba(filteredFeaturesTestMergedData);
fprTest, tprTest, thresholdTest = roc_curve(classesTest, predProbScoresTest[:, 1]);
areaUnderRocValueTest = auc(fprTest, tprTest);
plt.plot(fprTrain, tprTrain, 'y', label="Train ROC curve - {} vectorized text".format("Tf-Idf"));
plt.plot(fprTest, tprTest, 'g', label="Test ROC curve - {} vectorized text".format("Tf-Idf"));
plt.plot([0, 1], [0, 1], 'k-');
plt.title("ROC Curves for train and test data using k-value {}".format(optimalKValue))
plt.xlabel('False positive rate - FPR');
plt.ylabel('True positive rate - TPR');
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show();

print("Results of analysis using {} vectorized text features merged with other features using K-NN
brute force algorithm:".format("Tf-Idf"));
equalsBorder(70);
print("AUC values of train data: ");
equalsBorder(40);
print(areaUnderRocValuesTrain);
equalsBorder(40);
print("Optimal K-Value: ", optimalKValue);
equalsBorder(40);
print("AUC value of test data: ", areaUnderRocValueTest);
# Predicting classes of test data projects
predictionClassesTest = knnClassifier.predict(filteredFeaturesTestMergedData);
equalsBorder(40);
# Adding results to results dataframe
selectedFeaturesResultsDataFrame = selectedFeaturesResultsDataFrame.append({'Vectorizer': "Tf-Idf"
, 'Model': 'Brute', 'Hyper Parameter - K': optimalKValue, 'AUC': areaUnderRocValueTest}, ignore_ind
ex = True);
# Printing confusion matrix
confusionMatrix = confusion_matrix(classesTest, predictionClassesTest);
# Creating dataframe for generated confusion matrix
confusionMatrixDataFrame = pd.DataFrame(data = confusionMatrix, index = ['Actual: NO', 'Actual: YES
'], columns = ['Predicted: NO', 'Predicted: YES']);
print("Confusion Matrix : ");
equalsBorder(60);
confusionMatrixDataFrame
```








```
Results of analysis using Tf-Idf vectorized text features merged with other features using K-NN br
ute force algorithm:
========================================================================
AUC values of train data:
========================================
[0.9129600699340139, 0.9105657467870607, 0.8820615744200531, 0.8441363074351376,
0.8172390430088156, 0.8015838395266643, 0.7874551480982624, 0.7717280210510082, 0.761433674607709,
0.7534328482694689, 0.7471611991430402, 0.7396569254137402, 0.7338309325838084,
0.7267953833586733, 0.7201293359119455, 0.7163098001423787, 0.7127235154083043,
0.7092761731297839, 0.7045749863804595, 0.7013669418724919]
========================================
Optimal K-Value:  1
========================================
AUC value of test data:  0.5249076927180719
========================================
Confusion Matrix :
============================================================
```

Out[223]:

	Predicted: NO	Predicted: YES
Actual: NO	88	369
Actual: YES	363	2180

Summarizing results of above analysis using K-NN

Results of analysis on imbalanced and balanced when K-NN(simple cross validation) is used

In [294]:

```
resultsDataFrame
```

Out[294]:

	Vectorizer	Model	Hyper Parameter - K	AUC
0	Bag of Words	Brute	13	0.553233
1	Tf-Idf	Brute	17	0.549128
2	Average Word2Vec	Brute	31	0.557426
3	Tf-Idf Weighted Word2Vec	Brute	39	0.577138
4	Bag of Words	Brute	23	0.542332
5	Tf-Idf	Brute	19	0.553233
6	Average Word2Vec	Brute	25	0.522696
7	Tf-Idf Weighted Word2Vec	Brute	27	0.535176

Results of analysis on imbalanced and balanced when K-NN(k-fold cross validation) is used

In [195]:

```
kFoldResultsDataFrame
```

Out[195]:

	Vectorizer	Model	Hyper Parameter - K	AUC
0	Bag of Words	Brute	33	0.583700
1	Tf-Idf	Brute	35	0.555403
2	Average Word2Vec	Brute	31	0.562603
3	Tf-Idf Weighted Word2Vec	Brute	35	0.584067
4	Bag of Words	Brute	1	0.498529
5	Tf-Idf	Brute	1	0.513199
6	Average Word2Vec	Brute	1	0.530721
7	Tf-Idf Weighted Word2Vec	Brute	1	0.530499

Results of analysis on imbalanced and balanced when K-NN(k-fold cross validation) with top 200 features

In [287]:

```
selectedFeaturesResultsDataFrame
```

Out[287]:

	Vectorizer	Model	Hyper Parameter - K	AUC
0	Tf-Idf	Brute	1	0.524908
1	Tf-Idf	Brute	29	0.570047

Conclusions of above analysis:

1. The best k-value by considering AUC values will be 33 but the model trained with this k-value and imbalanced data is unable to predict the negative points and so it's like a dumb model.
2. While training with balanced data is able to predict negative and positive points considerably but with accuracy far less than training with imbalanced data.
3. It seems like K-NN cannot be used for solving the above problem if we want good prediction results.

6. It seems like RNN cannot be used for solving the above problem if we want good prediction results.