Donors choose data analysis using Random Forests & Gradient Boosting

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`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:**

Feature	Description
	• 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 [1]:

```
# 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
# For loading natural language processing tool-kit
import nltk
# For calculating mathematical terms
import math
# For plotting 3-D Plot
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
# For generating random numbers
import random

# For storing the model
from joblib import dump, load
# For loading files from google drive
from google.colab import drive
# For working with files in google drive
drive.mount('/content/drive')
# 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 converting dataframes into sparse matrix
from scipy.sparse import csr_matrix
# 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 model_selection
# For using decison tree classifier
from sklearn import tree
# For generating word cloud
from wordcloud import WordCloud
# For using random forest classifier
from sklearn.ensemble import RandomForestClassifier
# For using gradient boosting classifier
import xgboost as xgb
# For predicting probability values
from sklearn.calibration import CalibratedClassifierCV
# For plotting decision tree
import graphviz
# For using svm classifer - hinge loss function of sgd
from sklearn import linear_model
# 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 using gridsearch cv to find best parameter
from sklearn.model_selection import GridSearchCV
# For using randomized search cross validation
from sklearn.model_selection import RandomizedSearchCV
# For performing min-max standardization to features
from sklearn.preprocessing import MinMaxScaler
# For calculating sentiment score of the text
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')

warnings.filterwarnings('ignore')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

```
/usr/local/lib/python3.6/dist-packages/nltk/twitter/__init__.py:20: UserWarning:

The twython library has not been installed. Some functionality from the twitter package will not b
e available.
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

Reading and Storing Data

In [0]:

```
projectsData = pd.read_csv('drive/My Drive/train_data.csv');
resourcesData = pd.read_csv('drive/My Drive/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]:

	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 [0]:

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

In [0]:

```
# 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 [0]:

```
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
```

In [0]:

```
# 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 [12]:

```
# 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[12]:

	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

	Unnamed:	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro

In [13]:

```
# 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[13]:

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

In [14]:

```
# 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, orient='index');
sortedCategoriesData.columns = ['subject_categories'];
printStyle("Number of projects by Subject Categories: ", color.BOLD);
equalsBorder(60);
sortedCategoriesData
```

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

Out[14]:

	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 [15]:

```
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 [16]:

```python
# 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[16]:

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

In [17]:

```python
# 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, orient = 'index');
sortedSubCategoriesData.columns = ['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[17]:

	subject_sub_categories

	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

In [18]:

```
projectsData['project_essay'] = projectsData['project_essay_1'].map(str) + projectsData['project_essay_2'].map(str) + \
                                projectsData['project_essay_3'].map(str) + projectsData['project_essay_4'].map(str);
projectsData.head(5)
```

Out[18]:

	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

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
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 [19]:

```
priceAndQuantityData = resourcesData.groupby('id').agg({'price': 'sum', 'quantity':
'sum'}).reset_index();
priceAndQuantityData.head(5)
```

Out[19]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21
2	p000003	298.97	4
3	p000004	1113.69	98
4	p000005	485.99	8

In [20]:

```
projectsData.shape
```

Out[20]:

```
(109248, 20)
```

In [21]:

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

```
(109248, 22)
```

Out[21]:

	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

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Gra

In [22]:

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

Out[22]:

	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

In [23]:

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

Out[23]:

	id	price	quantity
253736	p253737	154.6	23

Preprocessing data

In [0]:

```
# 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', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]);
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 [25]:

```
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 [26]:

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

In [27]:

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

Out[27]:

```
['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 [28]:

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

Out[28]:

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

In [29]:

```
projectsData['preprocessed_titles'] = preProcessedProjectTitlesWithoutStopWords;
projectsData['preprocessed_essays'] = preProcessedEssaysWithoutStopWords;
projectsData.shape
```

Out[29]:

```
(109248, 24)
```

Preparing data for classification and modelling

In [30]:

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

Out[30]:

	All features in projects data
0	Unnamed: 0
1	id
2	teacher_id
3	teacher_prefix
4	school_state

All features in projects data

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
22	preprocessed_titles
23	preprocessed_essays

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

Classification & Modelling using random forests & Gradient Boosting

Splitting Data(Only training and test)

In [31]:

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

Out[31]:

```
(109245, 24)
```

In [32]:

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

```
(109245,)
```

In [0]:

```
trainingData, testData, classesTraining, classesTest = model_selection.train_test_split(projectsDat
a[0:60000], classesData[0:60000], test_size =  0.3, random_state = 44, stratify = classesData[0:600
00]);
trainingData, crossValidateData, classesTraining, classesCrossValidate =
model_selection.train_test_split(trainingData, classesTraining, test_size = 0.3, random_state = 0,
stratify = classesTraining);
```

In [34]:

```
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:  (18000, 24)
classesTest:  (18000,)
trainingData shape:  (29400, 24)
classesTraining shape:  (29400,)
```

In [35]:

```
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:  (4481, 24)
Number of positive points:  (24919, 24)
```

In [0]:

```
vectorizedFeatureNames = [];
```

Vectorizing categorical data

1. Vectorizing cleaned_categories(project_subject_categories cleaned) - Response Encoding

In [37]:

```
# Categorizing subjects categories feature using response encoding
subjectsCategoriesResponseData = [dict(), dict()];
for index, dataPoint in trainingData.iterrows():
    subjectCategory = dataPoint['cleaned_categories'];
    classValue = dataPoint['project_is_approved'];
    if(subjectCategory in subjectsCategoriesResponseData[classValue]):
      subjectsCategoriesResponseData[classValue][subjectCategory] += 1;
    else:
      subjectsCategoriesResponseData[classValue][subjectCategory] = 1;
allSubjectCategories = set(list(subjectsCategoriesResponseData[0].keys()) +
list(subjectsCategoriesResponseData[1].keys()));
for subjectCategory in allSubjectCategories:
  if(subjectCategory not in subjectsCategoriesResponseData[0]):
    subjectsCategoriesResponseData[0][subjectCategory] = 0;
  if(subjectCategory not in subjectsCategoriesResponseData[1]):
    subjectsCategoriesResponseData[1][subjectCategory] = 0;
def subjectsCategoriesTransform(subjectCategories):
  transformedData = pd.DataFrame(columns = ['SubjectsCategories0', 'SubjectsCategories1']);
  numRows = len(subjectCategories);
  for index, subjectCategory in enumerate(tqdm(subjectCategories)):
    if subjectCategory in allSubjectCategories:
      class0Value = subjectsCategoriesResponseData[0][subjectCategory];
      class1Value = subjectsCategoriesResponseData[1][subjectCategory];
      totalValue = class0Value + class1Value;
      class0Value = class0Value / totalValue;
      class1Value = class1Value / totalValue;
      transformedData.loc[index] = [class0Value, class1Value];
    else:
      transformedData.loc[index] = [0.5, 0.5];
  return csr_matrix(transformedData);
categoriesVector = subjectsCategoriesTransform(trainingData['cleaned_categories'].values);
```

In [38]:

```
print("Features used in vectorizing subject categories: ");
equalsBorder(70);
print(list(allSubjectCategories));
equalsBorder(70);
print("Shape of cleaned_categories matrix after vectorization(response encoding): ",
categoriesVector.shape);
equalsBorder(70);
print("Sample vectors of categories: ");
equalsBorder(70);
print(categoriesVector[0:4])
```

```
Features used in vectorizing subject categories:
======================================================================
['AppliedLearning History_Civics ', 'Literacy_Language Health_Sports ', 'Music_Arts SpecialNeeds '
, 'Health_Sports History_Civics ', 'Health_Sports Music_Arts ', 'Literacy_Language Math_Science ',
'Music_Arts History_Civics ', 'Math_Science History_Civics ', 'Math_Science Literacy_Language ',
'Literacy_Language Music_Arts ', 'Math_Science AppliedLearning ', 'Health_Sports Math_Science ',
'History_Civics AppliedLearning ', 'History_Civics Music_Arts ', 'Math_Science ', 'History_Civics
Health_Sports ', 'Music_Arts AppliedLearning ', 'Literacy_Language AppliedLearning ',
'SpecialNeeds Warmth Care_Hunger ', 'Math_Science Warmth Care_Hunger ', 'AppliedLearning
Music_Arts ', 'History_Civics Math_Science ', 'History_Civics Literacy_Language ', 'History_Civics
SpecialNeeds ', 'Health_Sports AppliedLearning ', 'History_Civics ', 'Warmth Care_Hunger ',
'Health_Sports Warmth Care_Hunger ', 'AppliedLearning Warmth Care_Hunger ', 'Literacy_Language His
tory_Civics ', 'Literacy_Language ', 'Health_Sports SpecialNeeds ', 'AppliedLearning SpecialNeeds
', 'Literacy_Language SpecialNeeds ', 'Literacy_Language Warmth Care_Hunger ', 'Math_Science
Music_Arts ', 'AppliedLearning Literacy_Language ', 'Music_Arts Health_Sports ', 'AppliedLearning
Health_Sports ', 'Math_Science Health_Sports ', 'SpecialNeeds Health_Sports ', 'SpecialNeeds ', 'Mu
sic_Arts ', 'AppliedLearning Math_Science ', 'AppliedLearning ', 'SpecialNeeds Music_Arts ',
'Health_Sports ', 'Math_Science SpecialNeeds ', 'Health_Sports Literacy_Language ']
======================================================================
Shape of cleaned_categories matrix after vectorization(response encoding):  (29400, 2)
======================================================================
Sample vectors of categories:
======================================================================
  (0, 0)	0.13858144972720188
  (0, 1)	0.8614185502727981
  (1, 0)	0.13858144972720188
  (1, 1)	0.8614185502727981
  (2, 0)	0.16129032258064516
  (2, 1)	0.8387096774193549
  (3, 0)	0.17974071632608218
  (3, 1)	0.8202592836739179
```

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

In [39]:

```
# Categorizing subjects sub categories feature using response encoding
subjectsSubCategoriesResponseData = [dict(), dict()];
for index, dataPoint in trainingData.iterrows():
    subjectSubCategory = dataPoint['cleaned_sub_categories'];
    classValue = dataPoint['project_is_approved'];
    if(subjectSubCategory in subjectsSubCategoriesResponseData[classValue]):
      subjectsSubCategoriesResponseData[classValue][subjectSubCategory] += 1;
    else:
      subjectsSubCategoriesResponseData[classValue][subjectSubCategory] = 1;
allSubjectSubCategories = set(list(subjectsSubCategoriesResponseData[0].keys()) +
list(subjectsSubCategoriesResponseData[1].keys()));
for subjectSubCategory in allSubjectSubCategories:
  if(subjectSubCategory not in subjectsSubCategoriesResponseData[0]):
    subjectsSubCategoriesResponseData[0][subjectSubCategory] = 0;
  if(subjectSubCategory not in subjectsSubCategoriesResponseData[1]):
    subjectsSubCategoriesResponseData[1][subjectSubCategory] = 0;
def subjectsSubCategoriesTransform(subjectSubCategories):
  transformedData = pd.DataFrame(columns = ['SubjectsSubCategories0', 'SubjectsSubCategories1']);
  numRows = len(subjectSubCategories);
  for index, subjectSubCategory in enumerate(tqdm(subjectSubCategories)):
    if subjectSubCategory in allSubjectSubCategories:
      class0Value = subjectsSubCategoriesResponseData[0][subjectSubCategory];
      class1Value = subjectsSubCategoriesResponseData[1][subjectSubCategory];
      totalValue = class0Value + class1Value;
      class0Value = class0Value / totalValue;
      class1Value = class1Value / totalValue;
      transformedData.loc[index] = [class0Value, class1Value];
    else:
      transformedData.loc[index] = [0.5, 0.5];
  return csr_matrix(transformedData);
subCategoriesVectors =
subjectsSubCategoriesTransform(trainingData['cleaned_sub_categories'].values);
```

In [40]:

```
print("Features used in vectorizing subject sub categories: ");
equalsBorder(70);
print(list(allSubjectSubCategories));
equalsBorder(70);
print("Shape of cleaned_sub_categories matrix after vectorization(response encoding): ",
subCategoriesVectors.shape);
equalsBorder(70);
print("Sample vectors of sub categories: ");
equalsBorder(70);
print(subCategoriesVectors[0:4])
```

```
Features used in vectorizing subject sub categories:
======================================================================
['Economics History_Geography ', 'AppliedSciences ESL ', 'ESL Extracurricular ', 'EarlyDevelopment
Mathematics ', 'Extracurricular Health_LifeScience ', 'Other ParentInvolvement ',
'College_CareerPrep ESL ', 'Music SocialSciences ', 'Mathematics ParentInvolvement ', 'Economics M
usic ', 'ParentInvolvement PerformingArts ', 'AppliedSciences PerformingArts ', 'CommunityService
NutritionEducation ', 'Health_LifeScience Mathematics ', 'Health_Wellness Literacy ',
'PerformingArts TeamSports ', 'Health_LifeScience Health_Wellness ', 'ParentInvolvement
SpecialNeeds ', 'EarlyDevelopment VisualArts ', 'Other TeamSports ', 'Extracurricular
Literature_Writing ', 'Civics_Government History_Geography ', 'College_CareerPrep Literacy ',
'EnvironmentalScience Literacy ', 'ESL EnvironmentalScience ', 'Civics_Government
EnvironmentalScience ', 'CommunityService ESL ', 'Civics_Government SocialSciences ',
'AppliedSciences SpecialNeeds ', 'FinancialLiteracy ', 'Music ', 'Literature_Writing TeamSports ',
'AppliedSciences ForeignLanguages ', 'Gym_Fitness Literature_Writing ', 'Mathematics ',
'CharacterEducation Warmth Care_Hunger ', 'AppliedSciences Health_Wellness ', 'AppliedSciences Hea
lth_LifeScience ', 'EnvironmentalScience SocialSciences ', 'CharacterEducation SpecialNeeds ', 'Ch
aracterEducation Gym_Fitness ', 'Literacy NutritionEducation ', 'Health_LifeScience
ParentInvolvement ', 'History_Geography Music ', 'ESL Literacy ', 'Literature_Writing SpecialNeeds
', 'ESL EarlyDevelopment ', 'Extracurricular History_Geography ', 'College_CareerPrep
PerformingArts ', 'Economics EnvironmentalScience ', 'CharacterEducation CommunityService ',
'Health_Wellness Literature_Writing ', 'CharacterEducation PerformingArts ', 'SpecialNeeds ',
'CommunityService ParentInvolvement ', 'FinancialLiteracy SpecialNeeds ', 'Health_LifeScience
Literacy ', 'Other SocialSciences ', 'ESL FinancialLiteracy ', 'Mathematics VisualArts ',
'Mathematics TeamSports ', 'Gym_Fitness Other ', 'Literature_Writing ', 'History_Geography Other
', 'Health_Wellness NutritionEducation ', 'Economics SocialSciences ', 'Health_LifeScience
SocialSciences ', 'NutritionEducation SpecialNeeds ', 'NutritionEducation ', 'EarlyDevelopment
SocialSciences ', 'FinancialLiteracy Health_Wellness ', 'Literature_Writing SocialSciences ',
'Gym_Fitness Mathematics ', 'Civics_Government Literature_Writing ', 'CommunityService Literacy ',
'Economics ', 'FinancialLiteracy Literacy ', 'AppliedSciences CommunityService ', 'Gym_Fitness Vis
ualArts ', 'EnvironmentalScience SpecialNeeds ', 'FinancialLiteracy SocialSciences ',
'EarlyDevelopment Warmth Care_Hunger ', 'EarlyDevelopment Gym_Fitness ', 'College_CareerPrep Music
', 'History_Geography Literature_Writing ', 'AppliedSciences Literacy ', 'Gym_Fitness
History_Geography ', 'Literacy TeamSports ', 'Gym_Fitness Music ', 'AppliedSciences VisualArts ',
'EnvironmentalScience ', 'Extracurricular VisualArts ', 'History_Geography SocialSciences ',
'Gym_Fitness SpecialNeeds ', 'CharacterEducation Literature_Writing ', 'Extracurricular
ForeignLanguages ', 'Literacy SocialSciences ', 'CommunityService Extracurricular ',
'College_CareerPrep NutritionEducation ', 'NutritionEducation TeamSports ', 'CharacterEducation
Mathematics ', 'Mathematics PerformingArts ', 'Civics_Government NutritionEducation ', 'ESL
SpecialNeeds ', 'ForeignLanguages Other ', 'Other SpecialNeeds ', 'Extracurricular ',
'Health_Wellness Other ', 'EnvironmentalScience VisualArts ', 'Civics_Government FinancialLiteracy
', 'Economics Mathematics ', 'Gym_Fitness TeamSports ', 'ESL ', 'College_CareerPrep
Health_Wellness ', 'Health_LifeScience Warmth Care_Hunger ', 'Mathematics Other ',
'CommunityService VisualArts ', 'ESL Mathematics ', 'Health_LifeScience NutritionEducation ',
'AppliedSciences NutritionEducation ', 'Mathematics Music ', 'EnvironmentalScience Health_Wellness
', 'Health_Wellness VisualArts ', 'Civics_Government ', 'EnvironmentalScience TeamSports ',
'Health_Wellness History_Geography ', 'College_CareerPrep Economics ', 'EarlyDevelopment Economics
', 'CommunityService EnvironmentalScience ', 'College_CareerPrep Health_LifeScience ',
'College_CareerPrep EarlyDevelopment ', 'History_Geography Literacy ', 'AppliedSciences
EnvironmentalScience ', 'Health_Wellness PerformingArts ', 'ParentInvolvement ', 'Economics Other
', 'Literacy Music ', 'Health_Wellness SpecialNeeds ', 'Civics_Government Extracurricular ',
'AppliedSciences Civics_Government ', 'Extracurricular Mathematics ', 'PerformingArts
SocialSciences ', 'College_CareerPrep ForeignLanguages ', 'CharacterEducation Literacy ',
'College_CareerPrep SpecialNeeds ', 'EnvironmentalScience NutritionEducation ',
'EnvironmentalScience Other ', 'ESL Literature_Writing ', 'FinancialLiteracy Mathematics ',
'FinancialLiteracy ForeignLanguages ', 'College_CareerPrep Gym_Fitness ', 'Health_Wellness
TeamSports ', 'CommunityService Literature_Writing ', 'Gym_Fitness PerformingArts ',
'PerformingArts SpecialNeeds ', 'Economics Literacy ', 'EnvironmentalScience Literature_Writing ',
'CharacterEducation Civics_Government ', 'EarlyDevelopment ForeignLanguages ', 'Health_Wellness ',
'CharacterEducation Economics ', 'Civics_Government TeamSports ', 'Extracurricular Other ',
'History_Geography ParentInvolvement ', 'Health_LifeScience Literature_Writing ',
'EarlyDevelopment ParentInvolvement ', 'Literacy PerformingArts ', 'Other ', 'PerformingArts ', 'C
haracterEducation VisualArts ', 'Civics_Government ESL ', 'Literacy ParentInvolvement ',
'EarlyDevelopment NutritionEducation ', 'Health_LifeScience Music ', 'ForeignLanguages Music ',
'FinancialLiteracy VisualArts ', 'ESL Music ', 'SocialSciences SpecialNeeds ', 'EarlyDevelopment
Literature_Writing ', 'ParentInvolvement VisualArts ', 'SocialSciences VisualArts ',
'ForeignLanguages Literacy ', 'SpecialNeeds Warmth Care_Hunger ', 'EarlyDevelopment
FinancialLiteracy ', 'College_CareerPrep FinancialLiteracy ', 'SocialSciences ',
'EnvironmentalScience PerformingArts ', 'CharacterEducation Music ', 'Health_Wellness Mathematics
', 'Health_LifeScience History_Geography ', 'Gym_Fitness ParentInvolvement ', 'Health_LifeScience
', 'EarlyDevelopment Extracurricular ', 'NutritionEducation Other ', 'EnvironmentalScience
ForeignLanguages ', 'CharacterEducation EarlyDevelopment ', 'Mathematics NutritionEducation ',
'ForeignLanguages History_Geography ', 'Literacy Warmth Care_Hunger ', 'Literacy Mathematics ', 'A
ppliedSciences SocialSciences ', 'Economics Literature_Writing ', 'College_CareerPrep
Extracurricular ', 'College_CareerPrep VisualArts ', 'AppliedSciences CharacterEducation ',
'FinancialLiteracy Health_LifeScience ', 'CommunityService ', 'EarlyDevelopment Literacy ',
'Extracurricular Literacy ', 'Music PerformingArts ', 'FinancialLiteracy Other ',
'College_CareerPrep History_Geography ', 'AppliedSciences EarlyDevelopment ', 'Health_LifeScience
TeamSports ', 'AppliedSciences Mathematics ', 'ESL History_Geography ', 'CommunityService
Health_LifeScience ', 'Civics_Government Mathematics ', 'CharacterEducation College_CareerPrep ',
'AppliedSciences ', 'Literacy SpecialNeeds ', 'EarlyDevelopment Other ', 'ESL VisualArts ',
'Literacy Other ', 'Civics_Government VisualArts ', 'AppliedSciences History_Geography ',
'ForeignLanguages Mathematics ', 'Extracurricular TeamSports ', 'Extracurricular Music ',
'History_Geography ', 'AppliedSciences Literature_Writing ', 'CommunityService FinancialLiteracy '
, 'College_CareerPrep Literature_Writing ', 'Music VisualArts ', 'Civics_Government
CommunityService ', 'ESL ForeignLanguages ', 'CharacterEducation FinancialLiteracy ',
'College_CareerPrep Warmth Care_Hunger ', 'College_CareerPrep SocialSciences ', 'EarlyDevelopment
Music ', 'Literature_Writing Other ', 'CharacterEducation ESL ', 'College_CareerPrep
EnvironmentalScience ', 'SpecialNeeds TeamSports ', 'FinancialLiteracy Literature_Writing ',
'CharacterEducation EnvironmentalScience ', 'CharacterEducation Health_Wellness ',
'EarlyDevelopment PerformingArts ', 'CommunityService PerformingArts ', 'Health_Wellness
SocialSciences ', 'Music TeamSports ', 'College_CareerPrep CommunityService ', 'Literature_Writing
PerformingArts ', 'Health_Wellness Music ', 'EnvironmentalScience ParentInvolvement ',
'EarlyDevelopment Health_Wellness ', 'Civics_Government SpecialNeeds ', 'Music SpecialNeeds ',
'Health_LifeScience SpecialNeeds ', 'College_CareerPrep Mathematics ', 'Literature_Writing
Mathematics ', 'ESL SocialSciences ', 'NutritionEducation SocialSciences ', 'Extracurricular
Gym_Fitness ', 'EnvironmentalScience Health_LifeScience ', 'Mathematics Warmth Care_Hunger ',
'History_Geography SpecialNeeds ', 'AppliedSciences ParentInvolvement ', 'ESL PerformingArts ', 'C
ollege_CareerPrep Other ', 'EnvironmentalScience History_Geography ', 'CharacterEducation
SocialSciences ', 'Literature_Writing VisualArts ', 'ParentInvolvement SocialSciences ',
'CharacterEducation TeamSports ', 'Warmth Care_Hunger ', 'Other PerformingArts ',
'Health_LifeScience VisualArts ', 'ForeignLanguages SocialSciences ', 'Literature_Writing
ParentInvolvement ', 'Health_Wellness Warmth Care_Hunger ', 'ForeignLanguages Literature_Writing '
, 'History_Geography PerformingArts ', 'ESL Other ', 'PerformingArts VisualArts ',
'EnvironmentalScience Music ', 'CharacterEducation ForeignLanguages ', 'FinancialLiteracy
History_Geography ', 'Literacy VisualArts ', 'ForeignLanguages ', 'Extracurricular Health_Wellness
', 'AppliedSciences Music ', 'Extracurricular PerformingArts ', 'Mathematics SpecialNeeds ',
'AppliedSciences Other ', 'College_CareerPrep ParentInvolvement ', 'ForeignLanguages SpecialNeeds
', 'TeamSports ', 'ESL Health_LifeScience ', 'Gym_Fitness Literacy ', 'CommunityService
Mathematics ', 'History_Geography VisualArts ', 'Gym_Fitness Health_Wellness ',
'Health_LifeScience Other ', 'CharacterEducation Extracurricular ', 'CommunityService Economics ',
'Music ParentInvolvement ', 'AppliedSciences Extracurricular ', 'CharacterEducation
Health_LifeScience ', 'Literacy ', 'Gym_Fitness ', 'Health_Wellness ParentInvolvement ',
'ForeignLanguages Health_Wellness ', 'CharacterEducation ', 'ESL ParentInvolvement ',
'AppliedSciences TeamSports ', 'EnvironmentalScience Extracurricular ', 'ParentInvolvement Warmth
Care_Hunger ', 'AppliedSciences Gym_Fitness ', 'CharacterEducation ParentInvolvement ',
'SpecialNeeds VisualArts ', 'EarlyDevelopment History_Geography ', 'EnvironmentalScience
Mathematics ', 'Literature_Writing Warmth Care_Hunger ', 'Civics_Government College_CareerPrep ',
'AppliedSciences College_CareerPrep ', 'CharacterEducation History_Geography ',
'EnvironmentalScience Gym_Fitness ', 'Other VisualArts ', 'EarlyDevelopment Health_LifeScience ',
'ForeignLanguages VisualArts ', 'EarlyDevelopment ', 'ForeignLanguages Health_LifeScience ',
'CharacterEducation Other ', 'Literature_Writing Music ', 'ESL Health_Wellness ', 'Extracurricular
SpecialNeeds ', 'CommunityService Other ', 'Civics_Government Economics ', 'EarlyDevelopment
EnvironmentalScience ', 'Gym_Fitness NutritionEducation ', 'CommunityService SpecialNeeds ',
'EarlyDevelopment SpecialNeeds ', 'CommunityService SocialSciences ', 'Extracurricular
NutritionEducation ', 'CommunityService Health_Wellness ', 'CommunityService History_Geography ',
'EarlyDevelopment TeamSports ', 'History_Geography Mathematics ', 'VisualArts ', 'Mathematics
SocialSciences ', 'College_CareerPrep ', 'Literacy Literature_Writing ', 'Extracurricular
ParentInvolvement ', 'CharacterEducation NutritionEducation ', 'Civics_Government Literacy ',
'Civics_Government Health_LifeScience ', 'Economics FinancialLiteracy ']
======================================================================
Shape of cleaned_sub_categories matrix after vectorization(response encoding):  (29400, 2)
======================================================================
Sample vectors of sub categories:
======================================================================
  (0, 0) 0.12631975867269984
  (0, 1) 0.8736802413273002
  (1, 0) 0.14383561643835616
  (1, 1) 0.8561643835616438
  (2, 0) 0.23255813953488372
  (2, 1) 0.7674418604651163
  (3, 0) 0.16629955947136563
  (3, 1) 0.8337004405286343
```

3. Vectorizing teacher_prefix - One Hot Encoding

In [0]:

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

In [42]:

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

Out[42]:

```
Counter({'Dr.': 2, 'Mr.': 2841, 'Mrs.': 15338, 'Ms.': 10577, 'Teacher': 642})
```

In [43]:

```
# Categorizing teacher prefixes feature using response encoding
teacherPrefixResponseData = [dict(), dict()];
for index, dataPoint in trainingData.iterrows():
    teacherPrefix = dataPoint['teacher_prefix'];
    classValue = dataPoint['project_is_approved'];
    if(teacherPrefix in teacherPrefixResponseData[classValue]):
      teacherPrefixResponseData[classValue][teacherPrefix] += 1;
    else:
      teacherPrefixResponseData[classValue][teacherPrefix] = 1;

allTeacherPrefixes = set(list(teacherPrefixResponseData[0].keys()) +
list(teacherPrefixResponseData[1].keys()));
```

```
for teacherPrefix in allTeacherPrefixes:
  if(teacherPrefix not in teacherPrefixResponseData[0]):
    teacherPrefixResponseData[0][teacherPrefix] = 0;
  if(teacherPrefix not in teacherPrefixResponseData[1]):
    teacherPrefixResponseData[1][teacherPrefix] = 0;

def teacherPrefixTransform(teacherPrefixes):
  transformedData = pd.DataFrame(columns = ['teacherPrefixes0', 'teacherPrefixes1']);
  numRows = len(teacherPrefixes);
  for index, teacherPrefix in enumerate(tqdm(teacherPrefixes)):
    if teacherPrefix in allTeacherPrefixes:
      class0Value = teacherPrefixResponseData[0][teacherPrefix];
      class1Value = teacherPrefixResponseData[1][teacherPrefix];
      totalValue = class0Value + class1Value;
      class0Value = class0Value / totalValue;
      class1Value = class1Value / totalValue;
      transformedData.loc[index] = [class0Value, class1Value];
    else:
      transformedData.loc[index] = [0.5, 0.5];
  return csr_matrix(transformedData);
teacherPrefixVectors = teacherPrefixTransform(trainingData['teacher_prefix'].values);
```

In [44]:

```
print("Features used in vectorizing teacher prefixes: ");
equalsBorder(70);
print(list(allTeacherPrefixes));
equalsBorder(70);
print("Shape of teacher prefixes matrix after vectorization(response encoding): ",
teacherPrefixVectors.shape);
equalsBorder(70);
print("Sample vectors of teacher prefixes: ");
equalsBorder(70);
print(teacherPrefixVectors[0:4]);
```

```
Features used in vectorizing teacher prefixes: 
======================================================================
['Dr.', 'Mr.', 'Ms.', 'Mrs.', 'Teacher']
======================================================================
Shape of teacher prefixes matrix after vectorization(response encoding):  (29400, 2)
======================================================================
Sample vectors of teacher prefixes: 
======================================================================
  (0, 0)	0.15770067126784532
  (0, 1)	0.8422993287321546
  (1, 0)	0.1451949406702308
  (1, 1)	0.8548050593297692
  (2, 0)	0.15770067126784532
  (2, 1)	0.8422993287321546
  (3, 0)	0.1451949406702308
  (3, 1)	0.8548050593297692
```

4. Vectorizing school_state - One Hot Encoding

In [45]:

```
# Categorizing school state feature using response encoding
schoolStateResponseData = [dict(), dict()];
for index, dataPoint in trainingData.iterrows():
    schoolState = dataPoint['school_state'];
    classValue = dataPoint['project_is_approved'];
    if(schoolState in schoolStateResponseData[classValue]):
      schoolStateResponseData[classValue][schoolState] += 1;
    else:
      schoolStateResponseData[classValue][schoolState] = 1;

allSchoolStates = set(list(schoolStateResponseData[0].keys()) + list(schoolStateResponseData[1].key
s()));

for schoolState in allSchoolStates:
  if(schoolState not in schoolStateResponseData[0]):
    schoolStateResponseData[0][schoolState] = 0;
    schoolStateResponseData[0][schoolState] = 0;
  if(schoolState not in schoolStateResponseData[1]):
    schoolStateResponseData[1][schoolState] = 0;

def schoolStateTransform(schoolStates):
  transformedData = pd.DataFrame(columns = ['SchoolStates0', 'SchoolStates1']);
  numRows = len(schoolStates);
  for index, schoolState in enumerate(tqdm(schoolStates)):
    if schoolState in allSchoolStates:
      class0Value = schoolStateResponseData[0][schoolState];
      class1Value = schoolStateResponseData[1][schoolState];
      totalValue = class0Value + class1Value;
      class0Value = class0Value / totalValue;
      class1Value = class1Value / totalValue;
      transformedData.loc[index] = [class0Value, class1Value];
    else:
      transformedData.loc[index] = [0.5, 0.5];
  return csr_matrix(transformedData);
schoolStateVectors = schoolStateTransform(trainingData['school_state'].values);
```

In [46]:

```
print("Features used in vectorizing school states: ");
equalsBorder(70);
print(list(allSchoolStates));
equalsBorder(70);
print("Shape of school states matrix after vectorization(response encoding): ", schoolStateVectors
.shape);
equalsBorder(70);
print("Sample vectors of school states: ");
equalsBorder(70);
print(schoolStateVectors[0:4]);
```

```
Features used in vectorizing school states:
======================================================================
['MI', 'AZ', 'WI', 'LA', 'IN', 'AK', 'CT', 'SD', 'MS', 'VT', 'RI', 'TN', 'CO', 'NY', 'NC', 'SC', 'C
K', 'NM', 'MD', 'WA', 'DE', 'MA', 'VA', 'MT', 'TX', 'ND', 'OR', 'NJ', 'MO', 'ME', 'PA', 'AL', 'HI',
'NV', 'AR', 'IL', 'WV', 'MN', 'NE', 'UT', 'WY', 'FL', 'NH', 'KS', 'CA', 'IA', 'ID', 'OH', 'GA', 'DC
', 'KY']
======================================================================
Shape of school states matrix after vectorization(response encoding):  (29400, 2)
======================================================================
Sample vectors of school states:
======================================================================
  (0, 0) 0.14498510427010924
  (0, 1) 0.8550148957298908
  (1, 0) 0.14498510427010924
  (1, 1) 0.8550148957298908
  (2, 0) 0.1440162271805274
  (2, 1) 0.8559837728194726
  (3, 0) 0.1649269311064718
  (3, 1) 0.8350730688935282
```

5. Vectorizing project_grade_category - One Hot Encoding

In [47]:

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

Out[47]:

```
Counter({'3-5': 10099,
         '6-8': 4574,
         '9-12': 2939,
         'Grades': 29400,
         'PreK-2': 11788})
```

In [48]:

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

Out[48]:

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

In [49]:

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

Out[49]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime
3781	116868	p019379	4c7282348b7ca01f9bb31f1195d344c0	Ms.	CA	2016-06-23 14:19:39
35253	61230	p031780	8b9c3f2053b086652a103e06bc3c1624	Mrs.	CA	2016-10-09 18:25:00
55224	8376	p125024	8705467872beedbb9a032f35d95bf236	Ms.	WI	2016-05-10 12:40:36
84	50274	p074147	1075b1b1e7882bc33e36c8e6b5a131d8	Mrs.	UT	2016-09-20 21:36:30

In [50]:

```
# Categorizing project grade feature using response encoding
projectGradeResponseData = [dict(), dict()];
for index, dataPoint in trainingData.iterrows():
    projectGrade = dataPoint['project_grade_category'];
    classValue = dataPoint['project_is_approved'];
    if(schoolState in projectGradeResponseData[classValue]):
      projectGradeResponseData[classValue][projectGrade] += 1;
    else:
      projectGradeResponseData[classValue][projectGrade] = 1;

allProjectGrades = set(list(projectGradeResponseData[0].keys()) + list(projectGradeResponseData[1].
keys()));

for projectGrade in allProjectGrades:
  if(projectGrade not in projectGradeResponseData[0]):
    projectGradeResponseData[0][projectGrade] = 0;
  if(projectGrade not in projectGradeResponseData[1]):
    projectGradeResponseData[1][projectGrade] = 0;

def projectGradeTransform(projectGrades):
  transformedData = pd.DataFrame(columns = ['ProjectGrades0', 'ProjectGrades1']);
  numRows = len(projectGrades);
  for index, projectGrade in enumerate(tqdm(projectGrades)):
    if projectGrade in allProjectGrades:
      class0Value = projectGradeResponseData[0][projectGrade];
      class1Value = projectGradeResponseData[1][projectGrade];
      totalValue = class0Value + class1Value;
      class0Value = class0Value / totalValue;
      class1Value = class1Value / totalValue;
      transformedData.loc[index] = [class0Value, class1Value];
    else:
      transformedData.loc[index] = [0.5, 0.5];
  return csr_matrix(transformedData);
projectGradeVectors = projectGradeTransform(trainingData['project_grade_category'].values);
```

In [51]:

```
print("Features used in vectorizing project grades: ");
equalsBorder(70);
print(list(allProjectGrades));
equalsBorder(70);
print("Shape of project grades matrix after vectorization(response encoding): ",
projectGradeVectors.shape);
equalsBorder(70);
print("Sample vectors of project grades: ");
equalsBorder(70);
print(projectGradeVectors[0:4]);
```

```
Features used in vectorizing project grades: 
======================================================================
['Grades6to8', 'GradesPreKto2', 'Grades9to12', 'Grades3to5']
======================================================================
Shape of project grades matrix after vectorization(response encoding):  (29400, 2)
======================================================================
Sample vectors of project grades: 
======================================================================
  (0, 0)	0.5
  (0, 1)	0.5
  (1, 0)	0.5
  (1, 1)	0.5
  (2, 0)	0.5
  (2, 1)	0.5
  (3, 0)	0.5
  (3, 1)	0.5
```

Vectorizing Text Data

In [52]:

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

In [0]:

```
bagOfWordsVectorizedFeatures = [];
```

Bag of Words

1. Vectorizing project_essay

In [0]:

```
# Initializing countvectorizer for bag of words vectorization of preprocessed project essays
bowEssayVectorizer = CountVectorizer(min_df = 10, max_features = 5000);
# Transforming the preprocessed essays to bag of words vectors
bowEssayModel = bowEssayVectorizer.fit_transform(preProcessedEssaysWithoutStopWords);
```

In [55]:

```
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:
======================================================================
['worries', 'worry', 'worrying', 'worth', 'worthy', 'would', 'wow', 'wrap', 'write', 'writer', 'wr
iters', 'writing', 'writings', 'written', 'wrong', 'wrote', 'xylophone', 'xylophones', 'yard', 'ye
ar', 'yearbook', 'yearly', 'yearn', 'yearning', 'years', 'yes', 'yesterday', 'yet', 'yoga', 'york'
, 'young', 'younger', 'youngest', 'youth', 'youtube', 'zest', 'zip', 'zone', 'zones', 'zoo']
======================================================================
Shape of preprocessed essay matrix after vectorization:  (29400, 5000)
======================================================================
Sample bag-of-words vector of preprocessed essay:
======================================================================
  (0, 2836)	1
  (0, 1902)	2
  (0, 3922)	1
  (0, 3059)	1
  (0, 2761)	2
  (0, 1787)	5
  (0, 3158)	1
  (0, 1777)	1
  (0, 3632)	1
  (0, 1654)	1
  (0, 823)	6
  (0, 1600)	2
  (0, 4842)	2
  (0, 4366)	4
  (0, 2677)	2
  (0, 3954)	5
  (0, 2226)	4
  (0, 419)	4
  (0, 4586)	1
  (0, 3047)	1
  (0, 3926)	1
  (0, 1071)	1
  (0, 3424)	1
  (0, 73)	1
  (0, 1565)	1
  :	:
  (0, 3992)	1
  (0, 1155)	3
  (0, 4836)	1
  (0, 312)	1
  (0, 2187)	2
  (0, 3576)	1
  (0, 816)	1
  (0, 2857)	1
  (0, 2965)	1
  (0, 4953)	1
  (0, 4591)	1
  (0, 2379)	1
  (0, 4226)	1
  (0, 4254)	1
  (0, 2038)	1
  (0, 4879)	2
  (0, 1788)	1
  (0, 2048)	1
  (0, 1625)	1
  (0, 3261)	1
  (0, 3095)	1
  (0, 4037)	1
  (0, 3903)	1
  (0, 3863)	1
  (0, 3015)	1
```

2. Vectorizing project_title

In [0]:

```
# 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 [57]:

```
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:
======================================================================
['wild', 'win', 'window', 'winning', 'winter', 'wireless', 'within', 'without', 'wizards', 'wo', '
wobble', 'wobbles', 'wobbling', 'wobbly', 'wonder', 'wonderful', 'wonders', 'word', 'words',
'work', 'working', 'works', 'workshop', 'world', 'worlds', 'worms', 'worth', 'would', 'write', 'wr
iters', 'writing', 'ye', 'year', 'yearbook', 'yes', 'yoga', 'young', 'youngest', 'youth', 'zone']
======================================================================
Shape of preprocessed title matrix after vectorization:  (29400, 1396)
======================================================================
Sample bag-of-words vector of preprocessed title:
======================================================================
  (0, 182)	1
  (0, 156)	1
  (0, 1313)	1
  (0, 1275)	1
```

Tf-Idf Vectorization

1. Vectorizing project_essay

In [0]:

```
# Intializing tfidf vectorizer for tf-idf vectorization of preprocessed project essays
tfIdfEssayVectorizer = TfidfVectorizer(min_df = 10, max_features = 5000);
# Transforming the preprocessed project essays to tf-idf vectors
tfIdfEssayModel = tfIdfEssayVectorizer.fit_transform(preProcessedEssaysWithoutStopWords);
```

In [59]:

```
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:
======================================================================
['worries', 'worry', 'worrying', 'worth', 'worthy', 'would', 'wow', 'wrap', 'write', 'writer', 'wr
iters', 'writing', 'writings', 'written', 'wrong', 'wrote', 'xylophone', 'xylophones', 'yard', 'ye
ar', 'yearbook', 'yearly', 'yearn', 'yearning', 'years', 'yes', 'yesterday', 'yet', 'yoga', 'york'
, 'young', 'younger', 'youngest', 'youth', 'youtube', 'zest', 'zip', 'zone', 'zones', 'zoo']
======================================================================
Shape of preprocessed title matrix after tf-idf vectorization:  (29400, 5000)
======================================================================
Sample Tf-Idf vector of preprocessed essay:
======================================================================
  (0, 3015)	0.01926912271987573
  (0, 3863)	0.06581027280531167
  (0, 3903)	0.10936621160060597
  (0, 4037)	0.05830509724792171
  (0, 3095)	0.026662468987984696
  (0, 3261)	0.055298810369397856
  (0, 1625)	0.07340458360370782
  (0, 2048)	0.05588218858932122
  (0, 1788)	0.09106747374963492
  (0, 4879)	0.19214559526835698
  (0, 2038)	0.08056711449863806
  (0, 4254)	0.1170161314929748
  (0, 4226)	0.077686152376333
  (0, 2379)	0.09885140727906383
  (0, 4591)	0.054011116589029545
  (0, 4953)	0.09641421546986415
  (0, 2965)	0.075552866332162
  (0, 2857)	0.05939996209635262
  (0, 816)	0.03481464612436699
  (0, 3576)	0.04037782458044481
  (0, 2187)	0.05590553523314935
  (0, 312)	0.08427073947843199
  (0, 4836)	0.0692585803155017
  (0, 1155)	0.10367184680045069
  (0, 3992)	0.04803322299914389
  :	:
  (0, 1565)	0.05118214937719337
  (0, 73)	0.09325504852678364
  (0, 3424)	0.06115883953510785
  (0, 1071)	0.09223154161229592
  (0, 3926)	0.11258888096135646
  (0, 3047)	0.07069561523758365
  (0, 4586)	0.047955321944185905
  (0, 419)	0.3125125332663674
  (0, 2226)	0.18047668022470958
  (0, 3954)	0.10698745392548661
  (0, 2677)	0.08501669989972188
  (0, 4366)	0.07425002816152836
  (0, 4842)	0.07628789153978258
  (0, 1600)	0.09659481636464504
  (0, 823)	0.15398544013401358
  (0, 1654)	0.09970671103765077
  (0, 3632)	0.13424109044851282
  (0, 1777)	0.08073196838011223
  (0, 3158)	0.03739069255991366
  (0, 1787)	0.26658067144983943
  (0, 2761)	0.13731259171226137
  (0, 3059)	0.06469789535385433
  (0, 3922)	0.08418178376309329
  (0, 1902)	0.19890201203969
  (0, 2836)	0.05801305653971179
```

2. Vectorizing project_title

In [0]:

```
# 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 [61]:

```
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: 
======================================================================
----------------------------------------------------------------------
['wild', 'win', 'window', 'winning', 'winter', 'wireless', 'within', 'without', 'wizards', 'wo', '
wobble', 'wobbles', 'wobbling', 'wobbly', 'wonder', 'wonderful', 'wonders', 'word', 'words',
'work', 'working', 'works', 'workshop', 'world', 'worlds', 'worms', 'worth', 'would', 'write', 'wr
iters', 'writing', 'ye', 'year', 'yearbook', 'yes', 'yoga', 'young', 'youngest', 'youth', 'zone']
======================================================================
Shape of preprocessed title matrix after tf-idf vectorization:  (29400, 1396)
======================================================================
Sample Tf-Idf vector of preprocessed title:
======================================================================
  (0, 1275) 0.5276309900818379
  (0, 1313) 0.38197788907501773
  (0, 156) 0.537852015087327
  (0, 182) 0.535176270428309
```

Average Word2Vector Vectorization

In [0]:

```
# 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('drive/My Drive/glove_vectors', 'rb') as f:
    gloveModel = pickle.load(f)
    gloveWords =  set(gloveModel.keys())
```

In [63]:

```
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 [0]:

```
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 [65]:

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

In [66]:

```
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: 29400,300
======================================================================
Sample essay:
======================================================================
may forget said never forget made feel one favorite maya angelou quotes exactly feel classroom env
ironment want students feel like school home away home title 1 school tucked away quaint
neighborhood san mateo county population made 80 english language learners dedicated providing equ
itable learning opportunity every student school classroom new school year classroom no large carp
et important first grade classroom place gather eager readers learners circle time mini lessons re
ading time partners individual also important designated spot know would love colorful rug
students see first day walk appreciate help classroom rug provide class place meet morning circle
time day reading workshop time reading together individually rug specific spots gives students wel
come feeling know go want students feel like classroom home away home carpet essential part enviro
nment would love place first day school not help feel welcome also set routines place right away n
annan
======================================================================
Word2Vec vector of sample essay:
======================================================================
[ 1.12471119e-01  4.71350130e-02 -1.26847804e-02 -1.07180031e-01
  4.20351608e-03 -6.19551245e-02 -2.97473769e+00 -2.30111392e-02
 -2.63128671e-02 -1.38888446e-01 -7.29432168e-03  2.35975910e-02
```

```
  8.99517993e-02 -8.41387909e-02 -7.55696720e-02 -9.76498224e-02
  1.78618171e-02 -5.12567699e-02  4.96212308e-03 -3.05456497e-02
  3.62935524e-02  3.04891497e-02 -1.33299860e-02 -4.26457420e-02
 -2.96593238e-02 -3.33273336e-02  4.93409441e-02  3.85871062e-02
  5.36207413e-03 -8.79827455e-02 -2.77544929e-01 -6.52610399e-02
  5.16278000e-02  1.64793664e-01 -3.78686224e-02 -3.99580930e-02
  3.15189811e-02 -2.45339832e-02 -4.67430818e-02 -6.76411832e-02
 -9.18983252e-02  5.13699601e-02 -2.39750042e-02 -1.17837648e-01
  4.62138601e-02 -1.50146685e-02  9.55656042e-02 -2.26131699e-02
  1.19367294e-02 -9.24030510e-02 -5.62439860e-04 -7.90948951e-03
  7.37042664e-02 -1.85386783e-02  2.71571049e-02 -9.86180364e-02
  5.08288531e-02  4.48803531e-02 -4.06459930e-02  1.09280694e-01
 -1.62181308e-02 -1.13938168e-01  9.31155152e-02 -1.28720497e-01
 -5.58955664e-02  4.47420211e-02  2.17511902e-02  4.75687916e-02
  1.33006776e-01 -4.49542011e-02 -1.64763770e-01  5.45135503e-02
 -6.09920490e-04 -1.18703438e-01 -8.20260783e-02 -1.84868527e-01
  3.36470378e-02  6.51100483e-02  6.79856614e-02 -7.30723420e-02
  2.31481119e-02 -2.84183164e-01 -2.02424434e-02  1.46299713e-02
 -2.76042902e-02  4.22624131e-02  1.26226664e-01 -9.94025035e-02
  1.48827469e-01  7.90762517e-03 -1.64119954e-02 -2.72477748e-02
 -1.05408615e-02  3.79997717e-02 -7.31182343e-02 -2.70963569e-01
 -2.13394839e+00  5.92876887e-02  1.72385655e-01  6.24967371e-02
 -8.92853063e-02 -4.08150266e-02  9.34932014e-02 -5.45428601e-02
  4.44140266e-02  5.39602098e-03 -2.35229362e-02 -1.30042724e-01
 -5.68859959e-02 -1.22936503e-03  4.71819580e-03 -3.62626224e-02
  6.48123510e-02  2.41494224e-01 -3.56930441e-02  1.52567338e-01
 -2.03174845e-01  3.08524497e-02  1.42671768e-01  1.18601399e-02
 -3.70199741e-02  4.62670308e-02  7.17045006e-02 -8.98027357e-02
  6.10790168e-02 -6.85447860e-02  1.09333443e-01 -3.89047769e-02
 -5.22813098e-02  1.21731071e-01  5.25312643e-02  2.50860699e-02
 -8.88798718e-02 -1.29561190e-01  7.21123427e-02 -9.79481301e-02
  1.04294232e-01 -5.92040763e-02  6.79383811e-02  4.11124336e-01
  1.23634321e-01  1.94895524e-02  6.03386650e-02  1.71661280e-02
 -5.63125303e-02 -4.84666280e-02  1.27135534e-02 -4.83062434e-02
  2.08251273e-01 -5.86346503e-03 -3.10670329e-02  1.52439287e-02
  1.04379975e-01 -2.31800902e-02  5.33976790e-02  1.41736364e-02
 -3.01182832e-02 -5.18702832e-02 -1.15060669e-01  6.16389161e-02
  8.82487294e-02 -1.37814895e-02 -4.89597937e-02 -1.13351201e-01
 -1.25162888e-01  8.92162881e-02 -4.69626692e-02  7.27701818e-02
  7.74103958e-02 -6.93435189e-02 -2.84860692e-02 -3.17532245e-02
 -3.43973098e-02 -1.50006232e-01 -1.61509027e-02 -2.26282168e-03
 -1.50525168e-02  6.16252559e-02 -1.12366755e-01 -8.16371119e-02
  2.84450070e-02  1.94623881e-01 -5.37379091e-02  3.11797343e-02
 -9.59922154e-02 -1.17003012e-01 -5.73356809e-02 -7.38555682e-02
  3.07314203e-02  4.39414685e-02 -7.66869045e-02 -8.09380559e-02
 -5.30492559e-02  1.90851399e-02  7.02124566e-02 -1.13717476e-01
  3.89847552e-02  1.73475035e-02  2.91997126e-02  3.78226149e-02
  1.25022590e-01 -2.42426657e-02 -9.43883217e-04  4.79770238e-02
 -6.08388727e-02  2.83808811e-02  8.72236667e-02 -7.74159832e-02
  2.42222468e-01  1.98133531e-02  1.38319310e-01 -3.00962657e-02
 -7.28444629e-02 -1.44365085e-01 -7.65295974e-02 -2.94600559e-03
 -1.07845313e-01 -4.61546469e-02 -9.46822245e-02 -4.22423615e-02
 -8.92158650e-02 -1.36251608e-03 -1.29234304e-01 -1.97542608e-01
 -2.44230308e+00  5.50849273e-02 -6.32100371e-02  2.79472517e-02
 -1.03341455e-01 -1.69413514e-01  1.79229119e-02  3.17661993e-02
 -3.79497916e-02 -8.25943077e-03 -1.28231311e-01  4.13816434e-02
  2.40297958e-02 -1.94647636e-02  2.33071413e-02  1.36391640e-01
 -5.34641175e-02 -7.91534755e-03 -2.72232176e-01  1.33006747e-01
 -8.29253203e-03  2.76799441e-02  1.44412685e-02 -9.65847294e-02
  2.79939823e-03 -3.28682098e-02  1.43899084e-02  9.14554519e-02
  9.93858804e-02 -3.68847413e-03  1.09918552e-01  1.58395406e-02
  1.01393655e-01  2.87143105e-02  3.18551189e-03 -1.44515687e-01
 -4.98306333e-02  1.13466154e-03 -1.87192622e-02 -1.40051727e-02
  7.47962469e-02 -3.03833916e-02 -1.18997968e-01 -3.53159959e-02
  7.61833860e-02  6.97373413e-02 -9.56159371e-03 -1.12915880e-01
 -2.36966415e-01  7.18602077e-02 -9.13090615e-02 -4.21152448e-04
  1.00062503e-01 -3.10420797e-02 -2.24921832e-02  5.75461922e-02
  2.50783133e-01 -9.47922685e-02 -7.10422937e-02  5.33298294e-02
 -1.28643175e-02  1.23950145e-01 -4.69837098e-02  4.34865531e-02
  2.53839217e-02 -5.48542245e-02 -1.22408993e-02  3.17757901e-02
 -9.17242930e-02 -1.02616085e-01  9.10432189e-02  4.45634028e-02
  2.56314140e-02  1.02408752e-01  8.56158212e-02 -6.32249657e-02]
```

2. Vectorizing project_title

In [67]:

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

In [68]:

```
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: 29400, 300
======================================================================
Sample title: 
======================================================================
carpet bring us together
======================================================================
Word2Vec vector of sample title: 
======================================================================
[ 0.01138825  0.03329575  0.07418175 -0.056228    0.15311845 -0.1071075
 -3.215625   -0.13671     0.0156175  -0.1627785  -0.25882375  0.01003425
  0.142081    0.0308865  -0.23050425  0.10522875 -0.106915   -0.08107882
  0.1189625   0.157595   -0.204674   -0.0191775  -0.14706325 -0.0493075
 -0.16155237  0.13297125  0.1061275   0.28609575  0.08378    -0.01867213
 -0.3579275  -0.19281    -0.1625465   0.01142    -0.11253825 -0.06970325
 -0.0087325  -0.34278    -0.213785   -0.2867365  -0.049391    0.1014475
 -0.119817   -0.140374    0.0381605   0.13426025  0.0770675  -0.2894615
  0.0328835  -0.15579925 -0.05002372 -0.192693   -0.0569935  -0.165171
 -0.05599925  0.128005   -0.1129975  -0.00818     0.062005     0.3587125
  0.053496   -0.0260725  -0.04713075 -0.012835    0.028754     0.1836095
  0.07381675  0.03364375  0.01001    -0.01533425 -0.2017375  -0.01981875
 -0.040057   -0.0522145  -0.2260625  -0.059355    0.23364725 -0.03804825
  0.03582655  0.0362325   0.17473275  0.122265   -0.19825875  0.008615
 -0.02749625 -0.22691425  0.07589275  0.1118775  -0.18560575  0.006455
 -0.08760425 -0.04248127 -0.236097    0.0397975  -0.08000525 -0.5945
 -2.110325   -0.1576275   0.4686875  -0.18604875  0.0521575  -0.07513025
  0.547405    0.1037935  -0.230985    0.1583855   0.0526005  -0.04743425
  0.0559655  -0.22033     0.0600125  -0.09355    -0.06314125 -0.067832
 -0.12674325 -0.072584   -0.290295   -0.028777    0.0938175   0.03288425
 -0.185264   -0.14696154  0.03498175 -0.139081   -0.1182145   0.00373125
 -0.16387725 -0.04168875  0.109441    0.0556915  -0.2542205  -0.13697
 -0.265435   -0.2871      0.3578425  -0.36731    -0.172635    0.06421025
  0.11912675  0.622585   -0.01297133  0.0118835   0.10994      0.16913
  0.03563425  0.0691135  -0.048685   -0.1410225   0.171525     0.076652
  0.10776375 -0.00637575 -0.05607025 -0.03554725  0.15348325 -0.22543575
  0.0042045   0.13903925 -0.0786445   0.09520525 -0.24312    -0.2288764
 -0.0285425  -0.050312   -0.03518075 -0.003716   -0.20302     0.0936375
  0.17715625 -0.35311    -0.10355325 -0.05693475 -0.25117545 -0.14200575
 -0.0281845   0.05719375 -0.0409775   0.3476     -0.004897   -0.060199
  0.090975   -0.2044525  -0.123666    0.2555325   0.0596025  -0.0182392
 -0.1420225   0.013418   -0.01354675  0.15792375 -0.0235625  -0.023775
 -0.08998575  0.03770732  0.072388   -0.2053525  -0.09501825  0.05546925
 -0.02747525  0.1128055  -0.0494175   0.045682   -0.2153115  -0.094872
  0.13554425  0.08358725  0.1031465  -0.13788775 -0.0866425  -0.2312575
  0.1232595   0.03951225 -0.149474   -0.08032625  0.13168575 -0.182765
 -0.2346825  -0.267063    0.018695   -0.13495025  0.28624    -0.13641945
 -0.01387625  0.0813125  -2.519475   -0.00507288 -0.165826    0.189605
  0.12097575 -0.21542075  0.163035   -0.07467525 -0.13038975  0.262395
 -0.162155    0.278985   -0.19788675  0.16445025 -0.1767775  -0.014365
  0.0053455   0.15700225 -0.13485     0.1937125   0.20771475  0.1314255
 -0.13007525 -0.07616125  0.1459965  -0.19568723  0.037662   -0.07145
  0.09571775 -0.13868227  0.040812    0.1209025  -0.2608215   0.158657
 -0.1228505  -0.1892725   0.1473525  -0.03123575 -0.0342785  -0.06028925
  0.22026     0.05392    -0.113052    0.02372375  0.24039525  0.1041815
 -0.1854855  -0.329232   -0.5017925  -0.15935    -0.15118675 -0.08824475
  0.1410656  -0.19407075 -0.323277   -0.1012725   0.4034825   0.13312975
  0.0534825   0.0595625   0.1629485   0.183905   -0.04107725 -0.1064415
 -0.1186785  -0.3138945  -0.28081175  0.199084   -0.045365    0.10960325
```

```
  0.24399425 -0.116697   -0.09865     0.09828525  0.1727685   0.0609875 ]
```

Tf-Idf Weighted Word2Vec Vectorization

1. Vectorizing project_essay

In [0]:

```python
# 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 [70]:

```python
# 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 [71]:

```python
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: 29400, 300
======================================================================
Sample Essay:
======================================================================
may forget said never forget made feel one favorite maya angelou quotes exactly feel classroom env
ironment want students feel like school home away home title 1 school tucked away quaint
neighborhood san mateo county population made 80 english language learners dedicated providing equ
itable learning opportunity every student school classroom new school year classroom no large carp
```

```
et important first grade classroom place gather eager readers learners circle time mini lessons re
ading time partners individual also important designated spot know would love colorful rug
students see first day walk appreciate help classroom rug provide class place meet morning circle
time day reading workshop time reading together individually rug specific spots gives students wel
come feeling know go want students feel like classroom home away home carpet essential part enviro
nment would love place first day school not help feel welcome also set routines place right away n
annan
======================================================================
Tf-Idf Weighted Word2Vec vector of sample essay:
======================================================================
[ 1.24853477e-01  5.26377839e-02 -2.85066789e-03 -1.12770421e-01
  3.36052391e-02 -6.35116045e-02 -2.92673618e+00 -8.42212703e-02
 -2.49990575e-02 -2.07446202e-01 -1.08429356e-02 -1.32800640e-02
  6.51197216e-02 -6.76028667e-02 -9.87902070e-02 -4.72800252e-02
 -5.23184692e-04 -1.41161219e-02 -2.79493917e-02 -7.73137779e-02
  2.22452269e-02  1.06039028e-02 -3.45131962e-02 -2.46358525e-02
 -5.48460381e-02  1.21558958e-02  2.95925391e-02  8.17506555e-02
  5.79181946e-02 -1.22439098e-01 -2.96359605e-01 -7.55089538e-02
  4.23530703e-02  2.00250397e-01 -2.83525321e-02 -7.32525273e-02
  4.62795928e-02 -8.81821724e-03 -3.58511788e-02 -1.06292349e-01
 -7.50706011e-02  1.19909162e-02 -5.49649575e-02 -1.06914756e-01
  8.56391786e-02  2.52848593e-02  1.41117647e-01 -4.77422657e-02
 -3.10764963e-03 -6.71818576e-02  3.49382549e-02 -4.50192979e-02
  8.88745015e-02 -5.20711238e-02  9.13277429e-03 -1.11919971e-01
  2.86661921e-02  3.90981929e-02  2.30179374e-02  1.24791284e-01
  2.39879429e-02 -1.30818726e-01  7.68496554e-02 -1.79863568e-01
 -2.00016555e-02  1.38226680e-02  8.42114176e-03  8.54362429e-02
  1.27035489e-01  7.21698648e-03 -1.58087425e-01 -4.49115346e-03
 -6.73186745e-03 -1.22052438e-01 -9.78433487e-02 -1.86891817e-01
  1.01651901e-01  1.07110657e-01  8.88148851e-02 -7.01831894e-02
  3.42529521e-03 -2.08424085e-01 -5.12014806e-02  6.18946308e-02
 -2.69668293e-03  6.98520161e-02  1.28781765e-01 -9.36417472e-02
  1.13995153e-01  7.35242593e-03 -1.55444623e-02 -8.00538917e-03
 -2.04438315e-02  3.15584290e-02 -9.36927809e-02 -3.27298551e-01
 -2.03516348e+00  5.95399061e-02  1.89077778e-01  2.07791558e-02
 -9.21465952e-02 -2.12589860e-03  6.78396938e-02 -2.90873800e-02
  5.26849469e-03 -2.84666863e-02 -1.08486647e-01 -1.24435887e-01
 -7.27453289e-02 -1.74157694e-02 -7.00312759e-03 -2.28447683e-02
  8.58710444e-02  2.37890576e-01 -7.69864058e-02  1.50831490e-01
 -2.04331466e-01  2.68337467e-02  1.55645544e-01 -2.01868708e-02
 -4.81311217e-02  2.15853365e-02  7.63674680e-02 -9.67348661e-02
  4.81168487e-02 -1.01741003e-01  1.16106257e-01 -6.57671059e-02
 -4.28831955e-02  1.03071886e-01  6.67581332e-02  2.16545391e-02
 -1.12795508e-01 -1.50346407e-01  7.32309308e-02 -8.30940069e-02
  9.53621365e-02 -5.71094320e-02  4.20716969e-02  4.23429077e-01
  1.33816016e-01  8.68988159e-03  6.84605796e-02  4.58070653e-02
 -7.96362646e-03 -1.60209544e-02 -1.56542741e-02 -4.16002483e-02
  1.64853682e-01 -1.71049931e-02 -5.47778867e-02  4.75837481e-02
  7.94015766e-02 -2.97446037e-02  9.54482938e-02  6.34993305e-02
 -7.93899307e-02 -5.76185404e-02 -1.52748511e-01  9.15408007e-02
  9.84275882e-02 -3.65947133e-03 -6.54266488e-02 -8.66342548e-02
 -1.52810378e-01  1.30453964e-01  1.36023520e-04  8.43441349e-02
  1.09816299e-01 -2.31050777e-02 -5.20714144e-02 -5.60464764e-02
 -4.02799746e-02 -1.49210013e-01 -2.81398143e-02 -5.21439761e-02
  1.06259886e-02  8.94192557e-02 -1.20620182e-01 -8.83047932e-02
  1.86881624e-02  1.37420960e-01 -5.21388580e-02  8.07981512e-02
 -1.06726808e-01 -1.13072014e-01 -5.22537085e-02 -6.71443644e-02
 -2.97827084e-02  3.59658564e-02 -7.59357428e-02 -8.29976833e-02
 -7.21681316e-02  2.08504685e-02  8.50050755e-02 -1.53215827e-01
  9.38194394e-02  1.79471531e-03  3.25892231e-03  4.67837637e-02
  1.29124799e-01 -9.27090779e-03  1.41737983e-02  1.71977853e-02
 -3.42810484e-02 -1.68312417e-02  1.08423518e-01 -8.55497821e-02
  2.43492526e-01  3.38394010e-02  1.64340419e-01 -4.98756012e-02
 -1.10969137e-01 -1.31011845e-01 -4.78937789e-02 -3.40566657e-02
 -1.07823854e-01 -4.36112180e-02 -8.00487176e-02 -1.08461223e-01
 -4.60207259e-02  3.30214358e-02 -1.04605806e-01 -2.70074919e-01
 -2.48436057e+00  1.13013249e-02 -8.27253317e-02  1.57752958e-02
 -1.10657629e-01 -1.90991149e-01  3.91769465e-02  4.99799297e-02
 -6.89864275e-02 -6.22676920e-04 -1.50748801e-01  6.15907834e-02
  1.99588137e-02 -8.24806593e-03  5.59510048e-03  1.31232900e-01
 -2.96500945e-02  3.27422263e-03 -2.53419225e-01  1.48081958e-01
  5.95583619e-02  1.08159791e-02  4.01645956e-02 -9.27587163e-02
  7.52540463e-03 -6.65669121e-02  5.15594295e-02  1.17771585e-01
  1.23088279e-01  9.46259146e-03  1.18398481e-01  7.63250872e-03
  1.15626804e-01  2.22879151e-02 -4.38362203e-02 -1.68250511e-01
 -5.27769074e-02  1.10654646e-02 -1.96079971e-02 -3.23488636e-02
  6.84017309e-02 -1.84952402e-02 -1.19386677e-01 -4.56792448e-02
 6.79929529e-02  5.97962731e-02 -4.40236180e-02 -1.42240203e-01
-3.02925935e-01  3.24254635e-02 -1.05686916e-01 -5.21908855e-02
 9.46989275e-02  2.70231195e-05 -3.94890112e-02  2.32384106e-02
 3.79000342e-01 -1.21541590e-01 -7.48300740e-02  3.76947402e-02
-4.07513881e-02  9.92543890e-02 -4.51761651e-02  5.87749996e-02
 2.92957415e-02 -9.42505558e-02 -2.96339237e-02  6.05931080e-02
-1.12810762e-01 -1.34183640e-01  1.29661564e-01  9.72513962e-02
 3.60369855e-02  6.49761702e-02  7.67317565e-02 -1.37648043e-01]
```

2. Vectorizing project_title

In [0]:

```
# 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 [73]:

```
# 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 [74]:

```
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: 29400, 300
======================================================================
Sample Title:
======================================================================
carpet bring us together
```

```
================================================================================
Tf-Idf Weighted Word2Vec vector of sample title:
================================================================================
[ 1.80074200e-02  5.65714661e-02  7.30731668e-02 -8.00615069e-02
  1.62778303e-01 -1.23045171e-01 -3.15239794e+00 -1.32714199e-01
  8.23130368e-03 -1.69862866e-01 -2.57348254e-01  6.61027279e-03
  1.41719789e-01  3.47332370e-02 -1.92306422e-01  9.68833505e-02
 -8.81305446e-02 -8.72195526e-02  1.08314450e-01  1.58294170e-01
 -2.16909625e-01 -2.91147021e-03 -1.55005319e-01 -5.61942277e-02
 -1.51335597e-01  1.22172647e-01  1.02240857e-01  3.03921196e-01
  7.92454373e-02 -1.97878653e-02 -3.49206311e-01 -2.25271901e-01
 -1.83705237e-01  1.06777540e-03 -1.12381540e-01 -5.27883650e-02
  1.24386643e-02 -3.34289280e-01 -2.00933849e-01 -3.12629290e-01
 -3.09424093e-02  1.19366157e-01 -8.90614225e-02 -1.48885716e-01
  5.82783003e-02  1.40216593e-01  6.97722719e-02 -2.97494494e-01
  2.15702913e-02 -1.50910836e-01 -5.09171289e-02 -1.94084040e-01
 -5.35268749e-02 -2.02901569e-01 -6.17592488e-02  1.60618518e-01
 -1.45207532e-01 -1.81850767e-02  8.97267626e-02  3.77800112e-01
  2.94070462e-02 -3.74178063e-02 -3.19166130e-02 -4.81329021e-02
  1.28551972e-02  1.46233658e-01  8.12359266e-02  2.04074200e-02
 -1.38257237e-02 -1.65225104e-02 -1.89545544e-01 -2.68373757e-02
 -5.26010049e-02 -6.63075551e-02 -2.11865130e-01 -9.27347796e-02
  2.48852513e-01 -4.51056710e-02  3.75122111e-02  1.86885031e-02
  1.60417060e-01  1.54147021e-01 -2.03286575e-01  5.63632398e-02
 -2.75297477e-02 -2.51300898e-01  4.40542214e-02  1.22540428e-01
 -1.85236366e-01 -3.39899721e-02 -7.25989928e-02 -5.94161402e-02
 -2.38659149e-01  2.73775852e-02 -1.09154173e-01 -5.68312952e-01
 -2.09450801e+00 -1.60528102e-01  4.68820979e-01 -1.93185606e-01
  6.89707810e-02 -1.18112450e-01  5.52787821e-01  8.89787404e-02
 -2.18854128e-01  1.61895243e-01 -6.66738405e-03 -2.57878454e-02
  3.63339197e-02 -2.06007884e-01  5.14652161e-02 -7.26800800e-02
 -6.54059240e-02 -6.43910738e-02 -1.14646237e-01 -8.55475038e-02
 -3.00189179e-01 -6.31222006e-02  1.29440885e-01  4.50317709e-02
 -1.96254574e-01 -1.23242550e-01  6.70403262e-02 -1.08858701e-01
 -1.11591967e-01  6.57503254e-03 -1.31318538e-01  1.14286360e-02
  6.84582583e-02  6.43620642e-02 -2.50772818e-01 -1.21549059e-01
 -2.57616095e-01 -3.30508828e-01  3.53944873e-01 -3.51413080e-01
 -1.64881167e-01  7.76834098e-02  1.23988658e-01  6.18864001e-01
 -1.14731066e-02  1.86232889e-02  1.06241538e-01  1.98013015e-01
  4.29153792e-02  1.05039821e-01 -1.81349134e-02 -1.35909239e-01
  1.73625492e-01  8.13016668e-02  1.12354045e-01  8.11626729e-04
 -6.56697732e-02 -2.06408298e-02  1.44540989e-01 -2.21602687e-01
  1.66672039e-02  1.49951118e-01 -8.28636282e-02  9.04899611e-02
 -2.52042276e-01 -2.12021441e-01 -1.07961831e-02 -4.57357734e-02
 -5.15653590e-02  6.94103889e-03 -2.04843034e-01  6.59200618e-02
  1.35071270e-01 -3.39230916e-01 -8.76428309e-02 -4.63251328e-02
 -2.70463898e-01 -1.32918819e-01 -2.38283126e-02  3.35589891e-02
 -3.97313010e-02  3.53751360e-01 -5.09347747e-02 -3.81136267e-02
  1.18371372e-01 -2.28177958e-01 -1.07202327e-01  2.34960820e-01
  7.52890865e-02 -3.28116316e-02 -1.43332403e-01 -7.90969610e-04
 -2.22565124e-02  1.57404609e-01 -4.84339594e-02 -5.56017112e-02
 -8.33981183e-02  4.06278351e-02  7.43701395e-02 -2.06796730e-01
 -8.82451124e-02  6.36814348e-02 -1.82125960e-02  1.42573589e-01
 -2.65368909e-02  3.78909384e-02 -2.21980683e-01 -6.67539369e-02
  1.42214311e-01  8.89560146e-02  1.05709292e-01 -1.42995235e-01
 -9.96963226e-02 -2.74667327e-01  1.16228717e-01  5.59525228e-02
 -1.37840410e-01 -8.80111318e-02  1.30927449e-01 -1.73552494e-01
 -2.35960627e-01 -2.46962028e-01  2.48936280e-02 -1.34398252e-01
  3.31914220e-01 -1.46276387e-01 -1.19434666e-03  5.77730244e-02
 -2.48273147e+00 -4.35998899e-03 -1.44742664e-01  1.74014294e-01
  1.06954521e-01 -2.20661126e-01  1.86634348e-01 -8.77357439e-02
 -1.28933737e-01  2.30102527e-01 -1.93923648e-01  2.73909317e-01
 -2.18936348e-01  1.67255169e-01 -1.77552885e-01  4.83479381e-03
  2.17529824e-02  1.43310946e-01 -1.25427795e-01  1.85610297e-01
  2.00798616e-01  1.37743176e-01 -1.01809577e-01 -5.09941688e-02
  1.28319438e-01 -2.12751248e-01  1.74749448e-02 -6.23675770e-02
  1.09185776e-01 -1.41392929e-01  4.65072439e-02  1.01756467e-01
 -2.73782229e-01  1.65534292e-01 -1.25343980e-01 -1.86509913e-01
  1.22581792e-01 -2.66707759e-02 -2.92267574e-02 -4.23745080e-02
  2.37633310e-01  4.88159577e-02 -1.22355319e-01  6.29464285e-02
  2.24022259e-01  7.07338835e-02 -2.01751580e-01 -3.60815601e-01
 -5.06214014e-01 -1.48752651e-01 -1.43785212e-01 -7.77949946e-02
  1.29774799e-01 -2.12211947e-01 -3.41729575e-01 -7.90092422e-02
  4.44015466e-01  1.41073128e-01  4.48183745e-02  4.28113602e-02
  1.52496665e-01  2.05931435e-01 -1.06144155e-02 -1.05914802e-01
 -9.08945292e-02 -3.16062784e-01 -2.83664656e-01  1.91864337e-01
 -6.83981995e-02  1.09869785e-01  2.58375972e-01 -1.50421561e-01
 -1.15691168e-01  7.46054100e-02  1.67477014e-01  4.83436783e-02]
```

Method for vectorizing unknown essays using our training data tf-idf weighted model

In [0]:

```
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;
```

Method for vectorizing unknown titles using our training data tf-idf weighted model

In [0]:

```
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;
```

Vectorizing numerical features

1. Vectorizing price

In [0]:

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

In [78]:

```
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:  (29400, 1)
======================================================================
Sample original prices:
======================================================================
[ 479.    1246.72  368.48  484.14   10.95]
Sample standardized prices:
======================================================================
[[0.04784194]
 [0.12462669]
 [0.03678811]
 [0.04835603]
 [0.00102917]]
```

2. Vectorizing quantity

In [0]:

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

In [80]:

```
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:  (29400, 1)
======================================================================
Sample original quantities:
======================================================================
[ 1  4  6 11 10]
Sample standardized quantities:
======================================================================
[[0.        ]
 [0.00322928]
 [0.00538213]
 [0.01076426]
 [0.00968784]]
```

3. Vectorizing teacher_number_of_previously_posted_projects

In [0]:

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

In [82]:

```
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:  (29400, 1)
======================================================================
Sample original quantities:
======================================================================
[0 0 4 3 8]
Sample standardized teacher_number_of_previously_posted_projects:
======================================================================
[[0.        ]
 [0.        ]
 [0.00941176]
 [0.00705882]
 [0.01882353]]
```

In [0]:

```
numberOfPoints = previouslyPostedStandardized.shape[0];
# Categorical data
categoriesVectorsSub = categoriesVector[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];

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

In [84]:

```
randomForestsAndGbdtResultsDataFrame = pd.DataFrame(columns =  ['Vectorizer', 'Model', 'Max Depth', 'N Estimators', 'AUC']);
randomForestsAndGbdtResultsDataFrame
```

Out[84]:

	Vectorizer	Model	Max Depth	N Estimators	AUC

Preparing cross validate data for analysis

In [85]:

```
# Test data categorical features transformation
categoriesTransformedCrossValidateData =
subjectsCategoriesTransform(crossValidateData['cleaned_categories']);
subCategoriesTransformedCrossValidateData = subjectsSubCategoriesTransform(crossValidateData['clea
ned_sub_categories']);
teacherPrefixTransformedCrossValidateData =
teacherPrefixTransform(crossValidateData['teacher_prefix']);
schoolStateTransformedCrossValidateData = schoolStateTransform(crossValidateData['school_state']);
projectGradeTransformedCrossValidateData =
projectGradeTransform(crossValidateData['project_grade_category']);

# Test 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);
avgWord2VecEssayTransformedCrossValidateData = getWord2VecVectors(preProcessedEssaysTemp);
avgWord2VecTitleTransformedCrossValidateData = getWord2VecVectors(preProcessedTitlesTemp);
tfIdfWeightedWord2VecEssayTransformedCrossValidateData =
getAvgTfIdfEssayVectors(preProcessedEssaysTemp);
tfIdfWeightedWord2VecTitleTransformedCrossValidateData =
getAvgTfIdfTitleVectors(preProcessedTitlesTemp);


# Test data 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 for analysis

In [86]:

```
# Test data categorical features transformation
categoriesTransformedTestData = subjectsCategoriesTransform(testData['cleaned_categories']);
subCategoriesTransformedTestData =
subjectsSubCategoriesTransform(testData['cleaned_sub_categories']);
teacherPrefixTransformedTestData = teacherPrefixTransform(testData['teacher_prefix']);
schoolStateTransformedTestData = schoolStateTransform(testData['school_state']);
projectGradeTransformedTestData = projectGradeTransform(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);
avgWord2VecEssayTransformedTestData = getWord2VecVectors(preProcessedEssaysTemp);
avgWord2VecTitleTransformedTestData = getWord2VecVectors(preProcessedTitlesTemp);
```

```
avgWord2VecTitleTransformedTestData = getWord2VecVectors(preProcessedTitlesTemp);
tfIdfWeightedWord2VecEssayTransformedTestData = getAvgTfIdfEssayVectors(preProcessedEssaysTemp);
tfIdfWeightedWord2VecTitleTransformedTestData = 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));
```

In [0]:

```
def configure_plotly_browser_state():
  import IPython
  display(IPython.core.display.HTML('''
        <script src="/static/components/requirejs/require.js"></script>
        <script>
          requirejs.config({
            paths: {
              base: '/static/base',
              plotly: 'https://cdn.plot.ly/plotly-latest.min.js?noext',
            },
          });
        </script>
        '''))
```

Classification using data vectorized by various models by random forests

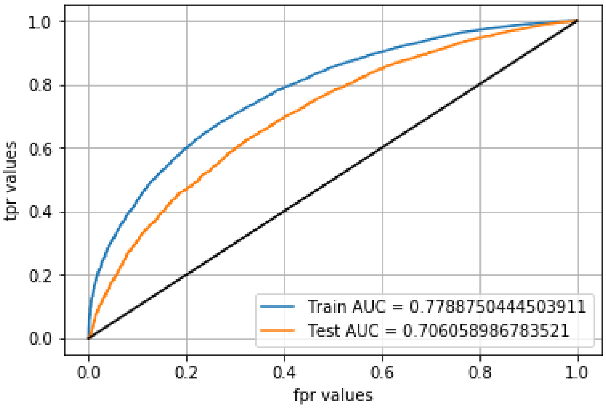
Classification using bag of words vectorized data by random forests

In [0]:

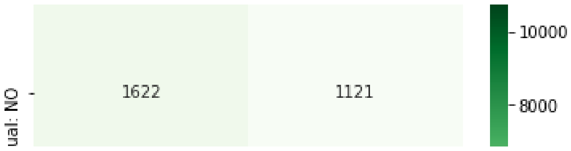
```
techniques = ['Bag of words'];
for index, technique in enumerate(techniques):
    trainingMergedData = hstack((categoriesVectorsSub,\
                                     subCategoriesVectorsSub,\
                                     teacherPrefixVectorsSub,\
                                     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));
    trainingLength = trainingMergedData.shape[0];
    crossValidationLength = crossValidateMergedData.shape[0];
    totalLength = trainingLength+crossValidationLength;
    classesTrainingSub = np.concatenate((classesTraining.values, classesCrossValidate.values));

    trainingAndCrossValidateMergedData = vstack((trainingMergedData, crossValidateMergedData));
    rfClassifier = RandomForestClassifier(class_weight="balanced", n_jobs = -1, min_samples_split =
500);
    tunedParameters = {'n_estimators': [100, 250, 400, 500, 700], 'max_depth': [5, 8, 10, 12]};



    classifier = GridSearchCV(rfClassifier, tunedParameters, cv = 3, scoring = 'roc_auc', n_jobs =
-1);
    classifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);

    crossValidateAucMeanValues = classifier.cv_results_['mean_test_score'];
    crossValidateAucStdValues = classifier.cv_results_['std_test_score'];

    trace1 = go.Scatter3d(x = tunedParameters['n_estimators'], y = tunedParameters['max_depth'], z
= crossValidateAucMeanValues, name = 'Cross-Validate');
    data = [trace1];

    layout = go.Layout(scene = dict(
            xaxis = dict(title='n_estimators'),
            yaxis = dict(title='max_depth'),
            zaxis = dict(title='AUC'),))

    fig = go.Figure(data=data, layout=layout)
    configure_plotly_browser_state()
    offline.iplot(fig, filename='3d-scatter-colorscale')

    optimalHypParamValue = classifier.best_params_['n_estimators'];
    optimalHypParam2Value = classifier.best_params_['max_depth'];
    rfClassifier = RandomForestClassifier(class_weight = 'balanced', n_estimators = optimalHypParam
Value, max_depth = optimalHypParam2Value, n_jobs = -1,  min_samples_split = 500);
    rfClassifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);
    predScoresTraining = rfClassifier.predict_proba(trainingAndCrossValidateMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTrainingSub, predScoresTraining[:, 1]);
    predScoresTest = rfClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predScoresTest[:, 1]);
    predictionClassesTest = rfClassifier.predict(testMergedData);

    equalsBorder(70);
    plt.plot(fprTrain, tprTrain, label = "Train AUC = " + str(auc(fprTrain, tprTrain)));
    plt.plot(fprTest, tprTest, label = "Test AUC = " + str(auc(fprTest, tprTest)));
    plt.plot([0, 1], [0, 1], 'k-');
    plt.xlabel("fpr values");
    plt.ylabel("tpr values");
    plt.grid();
    plt.legend();
    plt.show();

    areaUnderRocValueTest = auc(fprTest, tprTest);

    print("Results of analysis using {} vectorized text features merged with other features using
random forest classifier: ".format(technique));
    equalsBorder(70);
    print("Optimal n_estimators Value: ", optimalHypParamValue);
    equalsBorder(40);
    print("Optimal max_depth Value: ", optimalHypParam2Value);
    equalsBorder(40);
    print("AUC value of test data: ", str(areaUnderRocValueTest));
    # Predicting classes of test data projects
    predictionClassesTest = rfClassifier.predict(testMergedData);
    equalsBorder(40);
    # Printing confusion matrix
    confusionMatrix = confusion_matrix(classesTest, predictionClassesTest);
    # Creating dataframe for generated confusion matrix
    confusionMatrixDataFrame = pd.DataFrame(data = confusionMatrix, index = ['Actual: NO', 'Actual:
```

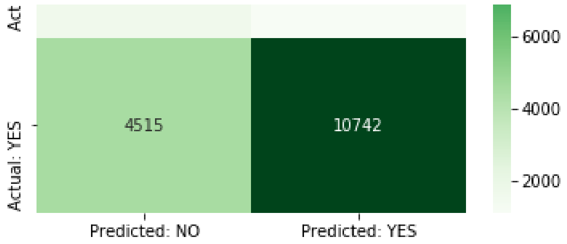
```
YES'], columns = ['Predicted: NO', 'Predicted: YES']);
    print("Confusion Matrix : ");
    equalsBorder(60);
    sbrn.heatmap(confusionMatrixDataFrame, annot = True, fmt = 'd', cmap="Greens");
    plt.show();
```

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



```
Results of analysis using Bag of words vectorized text features merged with other features using r
andom forest classifier:
======================================================================
Optimal n_estimators Value:  700
========================================
Optimal max_depth Value:  12
========================================
AUC value of test data:  0.706058986783521
========================================
Confusion Matrix :
============================================================
```

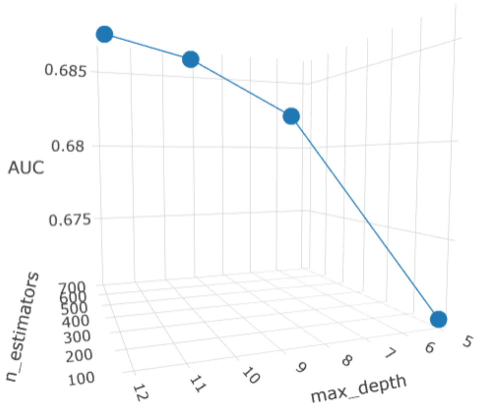




In [17]:

```
print("Cross-validation curve: ");
print("="*40);
display(Image(filename = "Random Forests - Bag of words.png", unconfined = True, width = '450px'));
```

```
Cross-validation curve: 
========================================
```

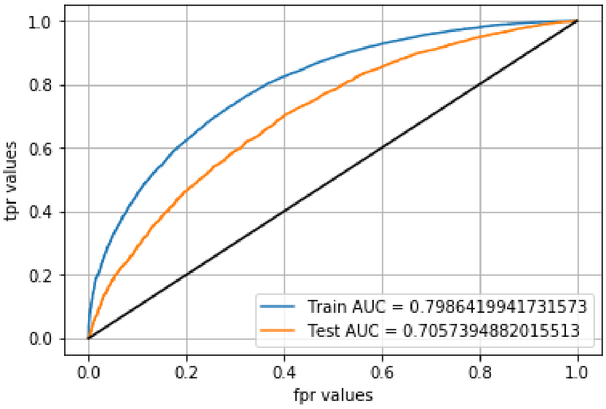


Classification using tf-idf vectorized data by random forests

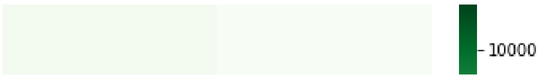
In [0]:

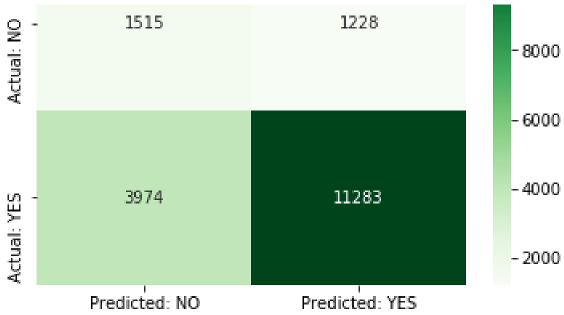
```
techniques = ['Tf-Idf'];
for index, technique in enumerate(techniques):
    trainingMergedData = hstack((categoriesVectorsSub,\
                                 subCategoriesVectorsSub,\
                                 teacherPrefixVectorsSub,\
                                 schoolStateVectorsSub,\
                                 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,\
                                     tfIdfTitleModelSub,\
                                     tfIdfEssayModelSub));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                 tfIdfTitleTransformedCrossValidateData,\
                                 tfIdfEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                 tfIdfTitleTransformedTestData,\
                                 tfIdfEssayTransformedTestData));

    trainingLength = trainingMergedData.shape[0];
    crossValidationLength = crossValidateMergedData.shape[0];
    totalLength = trainingLength+crossValidationLength;
    classesTrainingSub = np.concatenate((classesTraining.values, classesCrossValidate.values));

    trainingAndCrossValidateMergedData = vstack((trainingMergedData, crossValidateMergedData));
    rfClassifier = RandomForestClassifier(class_weight="balanced", n_jobs = -1, min_samples_split =
500);
    tunedParameters = {'n_estimators': [100, 250, 400, 500, 700], 'max_depth': [5, 8, 10, 12]};

    customCVIterator = []
    for i in range(3):
      trainIndices = np.arange(trainingLength);
      testIndices = random.sample(range(trainingLength, totalLength), 5000);
      customCVIterator.append( (trainIndices, testIndices) )


    classifier = GridSearchCV(rfClassifier, tunedParameters, cv = 3, scoring = 'roc_auc', n_jobs =
-1);
    classifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);

    crossValidateAucMeanValues = classifier.cv_results_['mean_test_score'];
    crossValidateAucStdValues = classifier.cv_results_['std_test_score'];

    trace1 = go.Scatter3d(x = tunedParameters['n_estimators'], y = tunedParameters['max_depth'], z
= crossValidateAucMeanValues, name = 'Cross-Validate');
    data = [trace1];

    layout = go.Layout(scene = dict(
            xaxis = dict(title='n_estimators'),
            yaxis = dict(title='max_depth'),
            zaxis = dict(title='AUC'),))

    fig = go.Figure(data=data, layout=layout)
    configure_plotly_browser_state()
    offline.iplot(fig, filename='3d-scatter-colorscale')

    optimalHypParamValue = classifier.best_params_['n_estimators'];
    optimalHypParam2Value = classifier.best_params_['max_depth'];
    rfClassifier = RandomForestClassifier(class_weight = 'balanced', n_estimators = optimalHypParam
Value, max_depth = optimalHypParam2Value, n_jobs = -1,  min_samples_split = 500);
    rfClassifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);
    predScoresTraining = rfClassifier.predict_proba(trainingAndCrossValidateMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTrainingSub, predScoresTraining[:, 1]);
    predScoresTest = rfClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predScoresTest[:, 1]);
    predictionClassesTest = rfClassifier.predict(testMergedData);

    equalsBorder(70);
    plt.plot(fprTrain, tprTrain, label = "Train AUC = " + str(auc(fprTrain, tprTrain)));
    plt.plot(fprTest, tprTest, label = "Test AUC = " + str(auc(fprTest, tprTest)));
    plt.plot([0, 1], [0, 1], 'k-');
    plt.xlabel("fpr values");
    plt.ylabel("tpr values");
    plt.grid();
    plt.legend();
    plt.show();

    areaUnderRocValueTest = auc(fprTest, tprTest);

    print("Results of analysis using {} vectorized text features merged with other features using
random forest classifier: ".format(technique));
    equalsBorder(70);
    print("Optimal n_estimators Value: ", optimalHypParamValue);
    equalsBorder(40);
    print("Optimal max_depth Value: ", optimalHypParam2Value);
    equalsBorder(40);
    print("AUC value of test data: ", str(areaUnderRocValueTest));
    # Predicting classes of test data projects
    predictionClassesTest = rfClassifier.predict(testMergedData);
    equalsBorder(40);
    # 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);
    sbrn.heatmap(confusionMatrixDataFrame, annot = True, fmt = 'd', cmap="Greens");
    plt.show();
```

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



```
Results of analysis using Tf-Idf vectorized text features merged with other features using random
forest classifier:
======================================================================
Optimal n_estimators Value:  250
========================================
Optimal max_depth Value:  12
========================================
AUC value of test data:  0.7057394882015513
========================================
Confusion Matrix :
============================================================
```

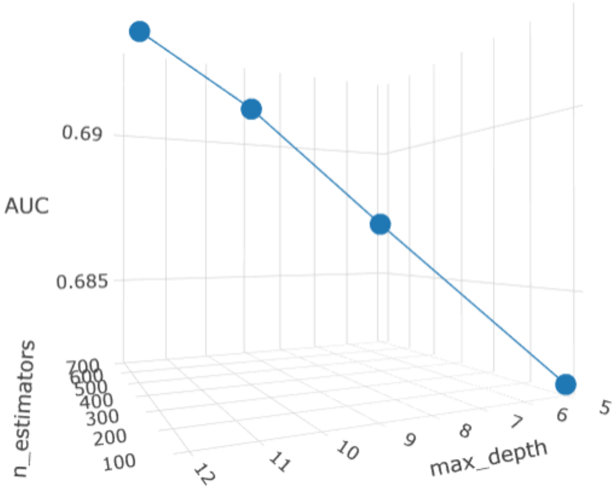




In [9]:

```
print("Cross-validation curve: ");
print("="*40);
display(Image(filename = "Random Forests - Tf-Idf.png", unconfined = True, width = '450px'));
```

```
Cross-validation curve: 
========================================
```



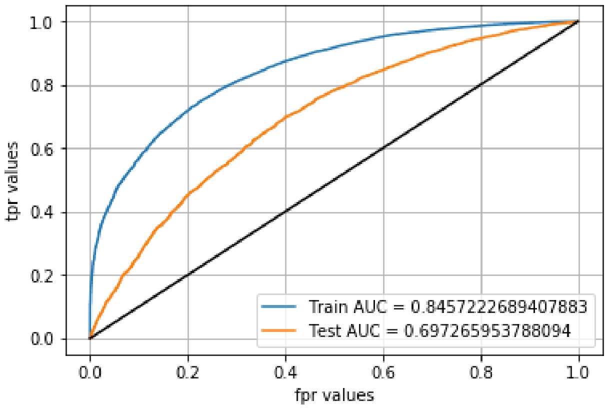
Classification using word2vec vectorized data by random forests

In [0]:

```
techniques = ['Average Word2Vec'];
for index, technique in enumerate(techniques):
    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):
```

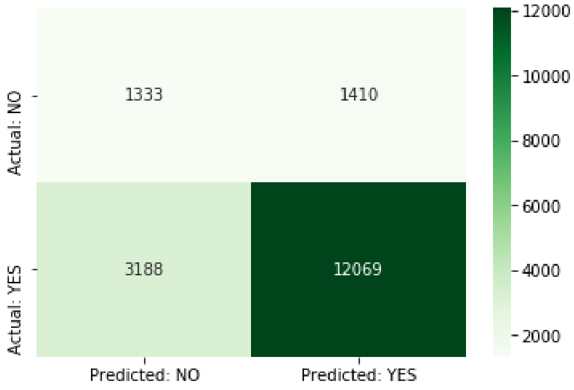
```
    if(index == 0):
        trainingMergedData = hstack((trainingMergedData,\
                                     word2VecTitlesVectors,\
                                     word2VecEssaysVectors));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                 avgWord2VecTitleTransformedCrossValidateData,\
                                 avgWord2VecEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                 avgWord2VecTitleTransformedTestData,\
                                 avgWord2VecEssayTransformedTestData));

    trainingLength = trainingMergedData.shape[0];
    crossValidationLength = crossValidateMergedData.shape[0];
    totalLength = trainingLength+crossValidationLength;
    classesTrainingSub = np.concatenate((classesTraining.values, classesCrossValidate.values));

    trainingAndCrossValidateMergedData = vstack((trainingMergedData, crossValidateMergedData));
    rfClassifier = RandomForestClassifier(class_weight="balanced", n_jobs = -1, min_samples_split = 500);
    tunedParameters = {'n_estimators': [100, 250, 400, 500, 700], 'max_depth': [5, 8, 10, 12, 15]};

    customCVIterator = []
    for i in range(3):
      trainIndices = np.arange(trainingLength);
      testIndices = random.sample(range(trainingLength, totalLength), 5000);
      customCVIterator.append( (trainIndices, testIndices) )


    classifier = GridSearchCV(rfClassifier, tunedParameters, cv = 3, scoring = 'roc_auc', n_jobs = -1);
    classifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);

    crossValidateAucMeanValues = classifier.cv_results_['mean_test_score'];
    crossValidateAucStdValues = classifier.cv_results_['std_test_score'];

    trace1 = go.Scatter3d(x = tunedParameters['n_estimators'], y = tunedParameters['max_depth'], z = crossValidateAucMeanValues, name = 'Cross-Validate');
    data = [trace1];

    layout = go.Layout(scene = dict(
            xaxis = dict(title='n_estimators'),
            yaxis = dict(title='max_depth'),
            zaxis = dict(title='AUC'),))

    fig = go.Figure(data=data, layout=layout)
    configure_plotly_browser_state()
    offline.iplot(fig, filename='3d-scatter-colorscale')

    optimalHypParamValue = classifier.best_params_['n_estimators'];
    optimalHypParam2Value = classifier.best_params_['max_depth'];
    rfClassifier = RandomForestClassifier(class_weight = 'balanced', n_estimators = optimalHypParamValue, max_depth = optimalHypParam2Value, n_jobs = -1,  min_samples_split = 500);
    rfClassifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);
    predScoresTraining = rfClassifier.predict_proba(trainingAndCrossValidateMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTrainingSub, predScoresTraining[:, 1]);
    predScoresTest = rfClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predScoresTest[:, 1]);
    predictionClassesTest = rfClassifier.predict(testMergedData);

    equalsBorder(70);
    plt.plot(fprTrain, tprTrain, label = "Train AUC = " + str(auc(fprTrain, tprTrain)));
    plt.plot(fprTest, tprTest, label = "Test AUC = " + str(auc(fprTest, tprTest)));
    plt.plot([0, 1], [0, 1], 'k-');
    plt.xlabel("fpr values");
    plt.ylabel("tpr values");
    plt.grid();
    plt.legend();
    plt.show();

    areaUnderRocValueTest = auc(fprTest, tprTest);

    print("Results of analysis using {} vectorized text features merged with other features using random forest classifier: ".format(technique));
    equalsBorder(70);
    print("Optimal n_estimators Value: ", optimalHypParamValue);
    equalsBorder(40);
    print("Optimal max_depth Value: ", optimalHypParam2Value);
    equalsBorder(40);
    print("AUC value of test data: ", str(areaUnderRocValueTest));
    # Predicting classes of test data projects
    predictionClassesTest = rfClassifier.predict(testMergedData);
    equalsBorder(40);
    # 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);
    sbrn.heatmap(confusionMatrixDataFrame, annot = True, fmt = 'd', cmap="Greens");
    plt.show();
```

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



```
Results of analysis using Average Word2Vec vectorized text features merged with other features usi
ng random forest classifier:
======================================================================
Optimal n_estimators Value:  700
========================================
Optimal max_depth Value:  12
========================================
AUC value of test data:  0.697265953788094
========================================
Confusion Matrix :
```

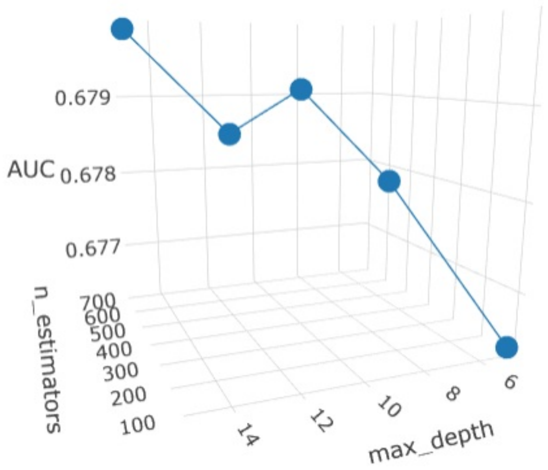
```
============================================================
```



In [11]:

```
print("Cross-validation curve: ");
print("="*40);
display(Image(filename = "Random Forests - Average word2Vec.png", unconfined = True, width = '450px'));
```

```
Cross-validation curve: 
========================================
```

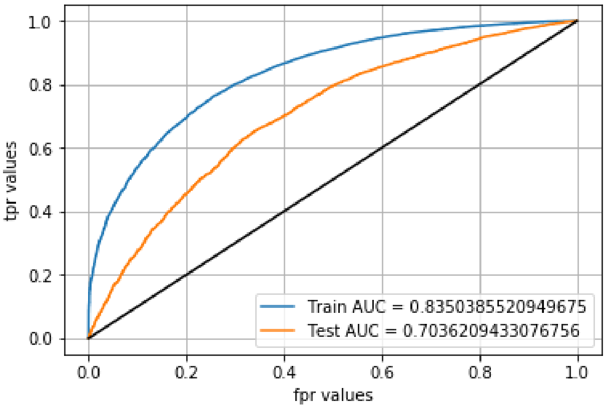


Classification using tf-idf weighted word2vec vectorized data by random forests

In [0]:

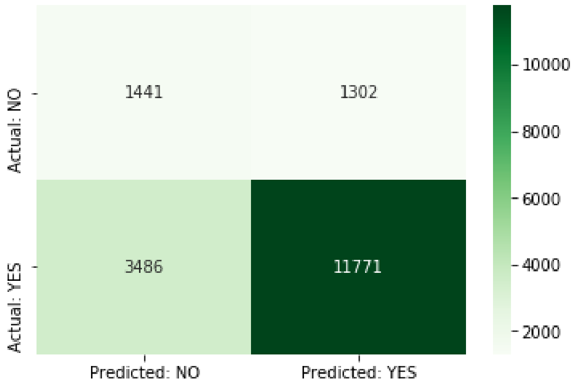
```
techniques = ['Tf-Idf Weighted Word2Vec'];
for index, technique in enumerate(techniques):
    trainingMergedData = hstack((categoriesVectorsSub,\
                                  subCategoriesVectorsSub,\
                                  teacherPrefixVectorsSub,\
                                  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,\
                                     tfIdfWeightedWord2VecTitlesVectors,\
                                     tfIdfWeightedWord2VecEssaysVectors));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                 tfIdfWeightedWord2VecTitleTransformedCrossValidateData,\
                                 tfIdfWeightedWord2VecEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                 tfIdfWeightedWord2VecTitleTransformedTestData,\
                                 tfIdfWeightedWord2VecEssayTransformedTestData));

    trainingLength = trainingMergedData.shape[0];
    crossValidationLength = crossValidateMergedData.shape[0];
    totalLength = trainingLength+crossValidationLength;
    classesTrainingSub = np.concatenate((classesTraining.values, classesCrossValidate.values));

    trainingAndCrossValidateMergedData = vstack((trainingMergedData, crossValidateMergedData));
    rfClassifier = RandomForestClassifier(class_weight="balanced", n_jobs = -1, min_samples_split =
500);
    tunedParameters = {'n_estimators': [100, 250, 400, 500, 700], 'max_depth': [5, 8, 10, 12]};


    classifier = GridSearchCV(rfClassifier, tunedParameters, cv = 3, scoring = 'roc_auc', n_jobs =
-1);
    classifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);

    crossValidateAucMeanValues = classifier.cv_results_['mean_test_score'];
    crossValidateAucStdValues = classifier.cv_results_['std_test_score'];

    trace1 = go.Scatter3d(x = tunedParameters['n_estimators'], y = tunedParameters['max_depth'], z
= crossValidateAucMeanValues, name = 'Cross-Validate');
    data = [trace1];

    layout = go.Layout(scene = dict(
            xaxis = dict(title='n_estimators'),
            yaxis = dict(title='max_depth'),
            zaxis = dict(title='AUC'),))

    fig = go.Figure(data=data, layout=layout)
    configure_plotly_browser_state()
    offline.iplot(fig, filename='3d-scatter-colorscale')

    optimalHypParamValue = classifier.best_params_['n_estimators'];
    optimalHypParam2Value = classifier.best_params_['max_depth'];
    rfClassifier = RandomForestClassifier(class_weight = 'balanced', n_estimators = optimalHypParam
Value, max_depth = optimalHypParam2Value, n_jobs = -1,  min_samples_split = 500);
    rfClassifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);
    predScoresTraining = rfClassifier.predict_proba(trainingAndCrossValidateMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTrainingSub, predScoresTraining[:, 1]);
    predScoresTest = rfClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predScoresTest[:, 1]);
    predictionClassesTest = rfClassifier.predict(testMergedData);

    equalsBorder(70);
    plt.plot(fprTrain, tprTrain, label = "Train AUC = " + str(auc(fprTrain, tprTrain)));
    plt.plot(fprTest, tprTest, label = "Test AUC = " + str(auc(fprTest, tprTest)));
    plt.plot([0, 1], [0, 1], 'k-');
    plt.xlabel("fpr values");
    plt.ylabel("tpr values");
    plt.grid();
    plt.legend();
    plt.show();

    areaUnderRocValueTest = auc(fprTest, tprTest);

    print("Results of analysis using {} vectorized text features merged with other features using
random forest classifier: ".format(technique));
    equalsBorder(70);
    print("Optimal n_estimators Value: ", optimalHypParamValue);
    equalsBorder(40);
    print("Optimal max_depth Value: ", optimalHypParam2Value);
    equalsBorder(40);
    print("AUC value of test data: ", str(areaUnderRocValueTest));
    # Predicting classes of test data projects
    predictionClassesTest = rfClassifier.predict(testMergedData);
    equalsBorder(40);
    # 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);
    sbrn.heatmap(confusionMatrixDataFrame, annot = True, fmt = 'd', cmap="Greens");
    plt.show();
```

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



```
Results of analysis using Tf-Idf Weighted Word2Vec vectorized text features merged with other feat
ures using random forest classifier:
======================================================================
Optimal n_estimators Value:  700
========================================
Optimal max_depth Value:  12
========================================
AUC value of test data:  0.7036209433076756
```

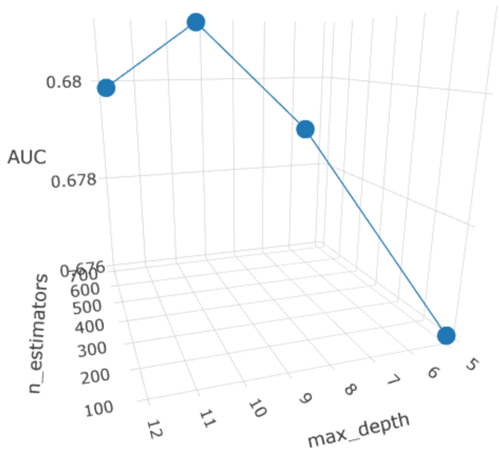
```
========================================
Confusion Matrix :
============================================================
```



In [12]:

```python
print("Cross-validation curve: ");
print("="*40);
display(Image(filename = "Random Forests - Tf-Idf Weighted Word2Vec.png", unconfined = True, width
= '450px'));
```

```
Cross-validation curve:
========================================
```



Classification using data vectorized by various models by gradient boosting classifier

In [0]:

```python
numNegative = 0;
numPositive = 0;
for classType in classesTrainingSub:
 if(classType == 0):
  numNegative += 1;
 else:
  numPositive += 1;
print(numNegative/numPositive);
```

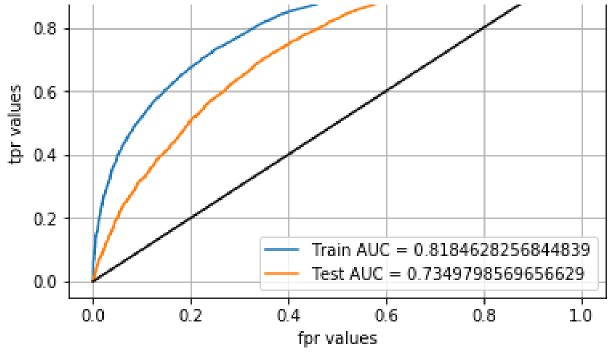
```
0.17980842158487598
```

Classification using bag of words vectorized data by gradient boosting classifier

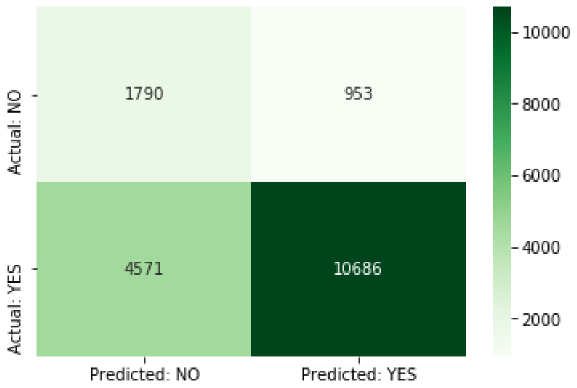
In [0]:

```
techniques = ['Bag of words'];
for index, technique in enumerate(techniques):
    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));
    trainingLength = trainingMergedData.shape[0];
    crossValidationLength = crossValidateMergedData.shape[0];
    totalLength = trainingLength+crossValidationLength;
    classesTrainingSub = np.concatenate((classesTraining.values, classesCrossValidate.values));

    trainingAndCrossValidateMergedData = vstack((trainingMergedData, crossValidateMergedData));
    gbdtClassifier = xgb.XGBClassifier(n_jobs = -1, min_samples_split = 500, reg_alpha = 1,
reg_lambda = 0, subsample = 0.5, colsample_bytree = 0.5, scale_pos_weight = 0.18);
    tunedParameters = {'n_estimators': [100, 250, 400, 500, 700], 'max_depth': [1, 3, 4, 5, 7]};
    classifier = GridSearchCV(gbdtClassifier, tunedParameters, cv = 5, scoring = 'roc_auc', n_jobs=
-1);
    classifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);

    testScoresDataFrame = pd.DataFrame(data =
np.hstack((classifier.cv_results_['param_n_estimators'].data[:, None],
classifier.cv_results_['param_max_depth'].data[:, None], classifier.cv_results_['mean_test_score']
[:, None], classifier.cv_results_['std_test_score'][:, None])), columns = ['n_estimators', 'max_dep
th', 'mts', 'stdts']);
    testScoresDataFrame = testScoresDataFrame.astype(float);

    crossValidateAucMeanValues = classifier.cv_results_['mean_test_score'];
    crossValidateAucStdValues = classifier.cv_results_['std_test_score'];

    trace1 = go.Scatter3d(x = tunedParameters['n_estimators'], y = tunedParameters['max_depth'], z
= crossValidateAucMeanValues, name = 'Cross-Validate');
    data = [trace1];

    layout = go.Layout(scene = dict(
            xaxis = dict(title='n_estimators'),
            yaxis = dict(title='max_depth'),
            zaxis = dict(title='AUC'),))

    fig = go.Figure(data=data, layout=layout)
    configure_plotly_browser_state()
    offline.iplot(fig, filename='3d-scatter-colorscale')

    optimalHypParamValue = classifier.best_params_['n_estimators'];
    optimalHypParam2Value = classifier.best_params_['max_depth'];
    gbdtClassifier = xgb.XGBClassifier(n_estimators = optimalHypParamValue, max_depth = optimalHypP
aram2Value, n_jobs = -1, min_samples_split = 500, reg_alpha = 1, reg_lambda = 0, subsample = 0.5, c
olsample_bytree = 0.5, scale_pos_weight = 0.18);
    gbdtClassifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);
    predScoresTraining = gbdtClassifier.predict_proba(trainingAndCrossValidateMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTrainingSub, predScoresTraining[:, 1]);
    predScoresTest = gbdtClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predScoresTest[:, 1]);
    predictionClassesTest = gbdtClassifier.predict(testMergedData);

    equalsBorder(70);
    plt.plot(fprTrain, tprTrain, label = "Train AUC = " + str(auc(fprTrain, tprTrain)));
    plt.plot(fprTest, tprTest, label = "Test AUC = " + str(auc(fprTest, tprTest)));
    plt.plot([0, 1], [0, 1], 'k-');
    plt.xlabel("fpr values");
    plt.ylabel("tpr values");
    plt.grid();
    plt.legend();
    plt.show();

    areaUnderRocValueTest = auc(fprTest, tprTest);

    print("Results of analysis using {} vectorized text features merged with other features using
gradient boosting classifier: ".format(technique));
    equalsBorder(70);
    print("Optimal n_estimators Value: ", optimalHypParamValue);
    equalsBorder(40);
    print("Optimal max_depth Value: ", optimalHypParam2Value);
    equalsBorder(40);
    print("AUC value of test data: ", str(areaUnderRocValueTest));
    # Predicting classes of test data projects
    predictionClassesTest = gbdtClassifier.predict(testMergedData);
    equalsBorder(40);
    # 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);
    sbrn.heatmap(confusionMatrixDataFrame, annot = True, fmt = 'd', cmap="Greens");
    plt.show();
```

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



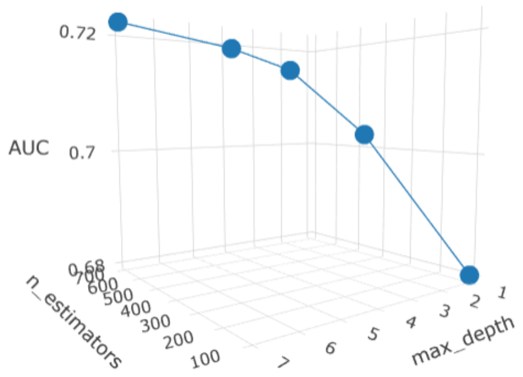
```
Results of analysis using Bag of words vectorized text features merged with other features using g
radient boosting classifier:
======================================================================
Optimal n_estimators Value:  400
========================================
Optimal max_depth Value:  3
========================================
AUC value of test data:  0.7349798569656629
========================================
Confusion Matrix :
============================================================
```



In [13]:

```
print("Cross-validation curve: ");
print("="*40);
display(Image(filename = "GBDT - Bag of words.png", unconfined = True, width = '450px'));
```

```
Cross-validation curve:
========================================
```

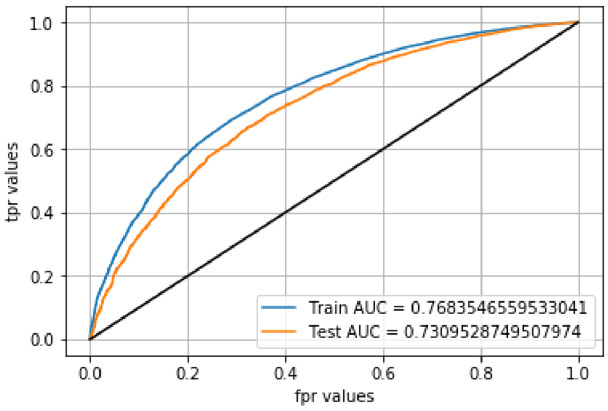


Classification using tf-idf vectorized data by gradient boosting classifier

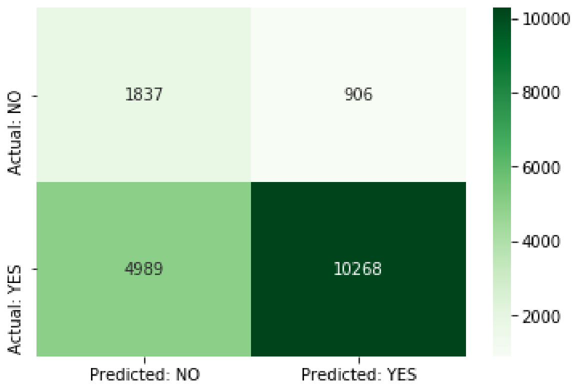
In [0]:

```
techniques = ['Tf-Idf'];
for index, technique in enumerate(techniques):
    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,\
                                     tfIdfTitleModelSub,\
                                     tfIdfEssayModelSub));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                     tfIdfTitleTransformedCrossValidateData,\
                                     tfIdfEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                     tfIdfTitleTransformedTestData,\
                                     tfIdfEssayTransformedTestData));

    trainingLength = trainingMergedData.shape[0];
    crossValidationLength = crossValidateMergedData.shape[0];
    totalLength = trainingLength+crossValidationLength;
    classesTrainingSub = np.concatenate((classesTraining.values, classesCrossValidate.values));

    trainingAndCrossValidateMergedData = vstack((trainingMergedData, crossValidateMergedData));
    gbdtClassifier = xgb.XGBClassifier(n_jobs = -1, min_samples_split = 500, reg_alpha = 1,
reg_lambda = 0, subsample = 0.5, colsample_bytree = 0.5, scale_pos_weight = 0.18);
    tunedParameters = {'n_estimators': [100, 250, 400, 500, 700], 'max_depth': [1, 3, 4, 5, 7]};
    classifier = GridSearchCV(gbdtClassifier, tunedParameters, cv = 5, scoring = 'roc_auc', n_jobs=
-1);
    classifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);

    testScoresDataFrame = pd.DataFrame(data =
np.hstack((classifier.cv_results_['param_n_estimators'].data[:, None],
classifier.cv_results_['param_max_depth'].data[:, None], classifier.cv_results_['mean_test_score']
[:, None], classifier.cv_results_['std_test_score'][:, None])), columns = ['n_estimators', 'max_dep
th', 'mts', 'stdts']);
    testScoresDataFrame = testScoresDataFrame.astype(float);

    crossValidateAucMeanValues = classifier.cv_results_['mean_test_score'];
    crossValidateAucStdValues = classifier.cv_results_['std_test_score'];

    trace1 = go.Scatter3d(x = tunedParameters['n_estimators'], y = tunedParameters['max_depth'], z
= crossValidateAucMeanValues, name = 'Cross-Validate');
    data = [trace1];

    layout = go.Layout(scene = dict(
            xaxis = dict(title='n_estimators'),
            yaxis = dict(title='max_depth'),
            zaxis = dict(title='AUC'),))

    fig = go.Figure(data=data, layout=layout)
    configure_plotly_browser_state()
    offline.iplot(fig, filename='3d-scatter-colorscale')

    optimalHypParamValue = classifier.best_params_['n_estimators'];
    optimalHypParam2Value = classifier.best_params_['max_depth'];
    optimalHypParam2Value = classifier.best_params_['max_depth'];
    gbdtClassifier = xgb.XGBClassifier(n_estimators = optimalHypParamValue, max_depth = optimalHypParam2Value, n_jobs = -1, min_samples_split = 500, reg_alpha = 1, reg_lambda = 0, subsample = 0.5, colsample_bytree = 0.5, scale_pos_weight = 0.18);
    gbdtClassifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);
    predScoresTraining = gbdtClassifier.predict_proba(trainingAndCrossValidateMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTrainingSub, predScoresTraining[:, 1]);
    predScoresTest = gbdtClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predScoresTest[:, 1]);
    predictionClassesTest = gbdtClassifier.predict(testMergedData);

    equalsBorder(70);
    plt.plot(fprTrain, tprTrain, label = "Train AUC = " + str(auc(fprTrain, tprTrain)));
    plt.plot(fprTest, tprTest, label = "Test AUC = " + str(auc(fprTest, tprTest)));
    plt.plot([0, 1], [0, 1], 'k-');
    plt.xlabel("fpr values");
    plt.ylabel("tpr values");
    plt.grid();
    plt.legend();
    plt.show();

    areaUnderRocValueTest = auc(fprTest, tprTest);

    print("Results of analysis using {} vectorized text features merged with other features using gradient boosting classifier: ".format(technique));
    equalsBorder(70);
    print("Optimal n_estimators Value: ", optimalHypParamValue);
    equalsBorder(40);
    print("Optimal max_depth Value: ", optimalHypParam2Value);
    equalsBorder(40);
    print("AUC value of test data: ", str(areaUnderRocValueTest));
    # Predicting classes of test data projects
    predictionClassesTest = gbdtClassifier.predict(testMergedData);
    equalsBorder(40);
    # 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);
    sbrn.heatmap(confusionMatrixDataFrame, annot = True, fmt = 'd', cmap="Greens");
    plt.show();
```

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



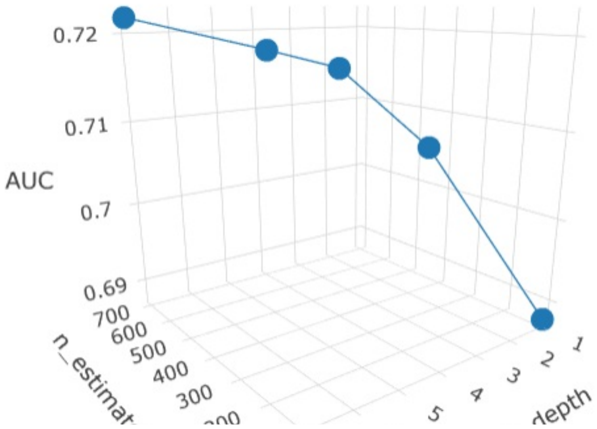
```
Results of analysis using Tf-Idf vectorized text features merged with other features using
gradient boosting classifier:
======================================================================
Optimal n_estimators Value:  700
========================================
Optimal max_depth Value:  1
========================================
AUC value of test data:  0.7309528749507974
========================================
Confusion Matrix :
============================================================
```



In [14]:

```python
print("Cross-validation curve: ");
print("="*40);
display(Image(filename = "GBDT - Tf-Idf.png", unconfined = True, width = '450px'));
```

```
Cross-validation curve: 
========================================
```



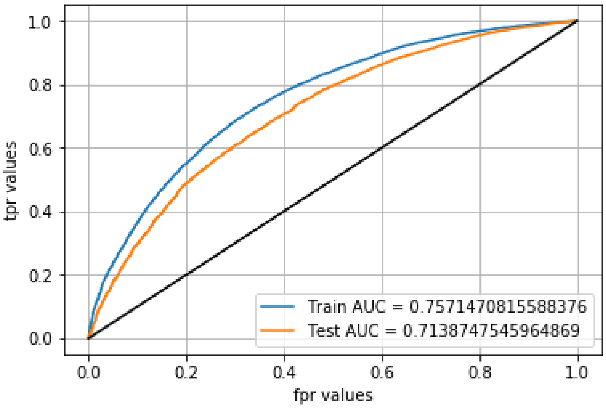
Classification using word2vec vectorized data by gradient boosting classifier

In [0]:

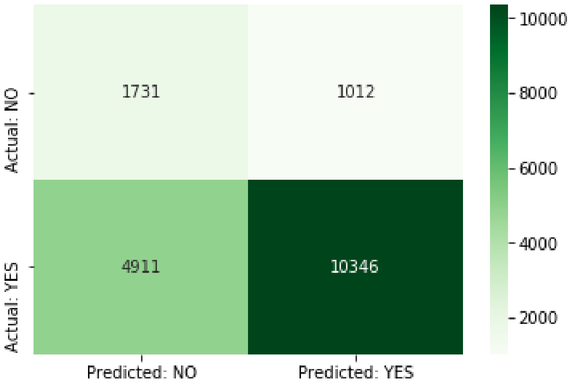
```
techniques = ['Average Word2Vec'];
for index, technique in enumerate(techniques):
    trainingMergedData = hstack((categoriesVectorsSub,\
                                     subCategoriesVectorsSub,\
                                     teacherPrefixVectorsSub,\
                                     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,\
                                     word2VecTitlesVectors,\
                                     word2VecEssaysVectors));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                 avgWord2VecTitleTransformedCrossValidateData,\
                                 avgWord2VecEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                 avgWord2VecTitleTransformedTestData,\
                                 avgWord2VecEssayTransformedTestData));

    trainingLength = trainingMergedData.shape[0];
    crossValidationLength = crossValidateMergedData.shape[0];
    totalLength = trainingLength+crossValidationLength;
    classesTrainingSub = np.concatenate((classesTraining.values, classesCrossValidate.values));

    trainingAndCrossValidateMergedData = vstack((trainingMergedData, crossValidateMergedData));
    gbdtClassifier = xgb.XGBClassifier(n_jobs = -1, min_samples_split = 500, reg_alpha = 1,
reg_lambda = 0, subsample = 0.5, colsample_bytree = 0.5, scale_pos_weight = 0.18);
    tunedParameters = {'n_estimators': [100, 250, 400, 500, 700], 'max_depth': [1, 3, 4, 5, 7]};
    classifier = GridSearchCV(gbdtClassifier, tunedParameters, cv = 5, scoring = 'roc_auc', n_jobs=
-1);
    classifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);

    testScoresDataFrame = pd.DataFrame(data =
np.hstack((classifier.cv_results_['param_n_estimators'].data[:, None],
classifier.cv_results_['param_max_depth'].data[:, None], classifier.cv_results_['mean_test_score']
[:, None], classifier.cv_results_['std_test_score'][:, None])), columns = ['n_estimators', 'max_dep
th', 'mts', 'stdts']);
    testScoresDataFrame = testScoresDataFrame.astype(float);

    crossValidateAucMeanValues = classifier.cv_results_['mean_test_score'];
    crossValidateAucStdValues = classifier.cv_results_['std_test_score'];

    trace1 = go.Scatter3d(x = tunedParameters['n_estimators'], y = tunedParameters['max_depth'], z
= crossValidateAucMeanValues, name = 'Cross-Validate');
    data = [trace1];

    layout = go.Layout(scene = dict(
            xaxis = dict(title='n_estimators'),
            yaxis = dict(title='max_depth'),
            zaxis = dict(title='AUC'),))
```

```python
    fig = go.Figure(data=data, layout=layout)
    configure_plotly_browser_state()
    offline.iplot(fig, filename='3d-scatter-colorscale')

    optimalHypParamValue = classifier.best_params_['n_estimators'];
    optimalHypParam2Value = classifier.best_params_['max_depth'];
    gbdtClassifier = xgb.XGBClassifier(n_estimators = optimalHypParamValue, max_depth = optimalHypParam2Value, n_jobs = -1, min_samples_split = 500, reg_alpha = 1, reg_lambda = 0, subsample = 0.5, colsample_bytree = 0.5, scale_pos_weight = 0.18);
    gbdtClassifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);
    predScoresTraining = gbdtClassifier.predict_proba(trainingAndCrossValidateMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTrainingSub, predScoresTraining[:, 1]);
    predScoresTest = gbdtClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predScoresTest[:, 1]);
    predictionClassesTest = gbdtClassifier.predict(testMergedData);

    equalsBorder(70);
    plt.plot(fprTrain, tprTrain, label = "Train AUC = " + str(auc(fprTrain, tprTrain)));
    plt.plot(fprTest, tprTest, label = "Test AUC = " + str(auc(fprTest, tprTest)));
    plt.plot([0, 1], [0, 1], 'k-');
    plt.xlabel("fpr values");
    plt.ylabel("tpr values");
    plt.grid();
    plt.legend();
    plt.show();

    areaUnderRocValueTest = auc(fprTest, tprTest);

    print("Results of analysis using {} vectorized text features merged with other features using gradient boosting classifier: ".format(technique));
    equalsBorder(70);
    print("Optimal n_estimators Value: ", optimalHypParamValue);
    equalsBorder(40);
    print("Optimal max_depth Value: ", optimalHypParam2Value);
    equalsBorder(40);
    print("AUC value of test data: ", str(areaUnderRocValueTest));
    # Predicting classes of test data projects
    predictionClassesTest = gbdtClassifier.predict(testMergedData);
    equalsBorder(40);
    # 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);
    sbrn.heatmap(confusionMatrixDataFrame, annot = True, fmt = 'd', cmap="Greens");
    plt.show();
```

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



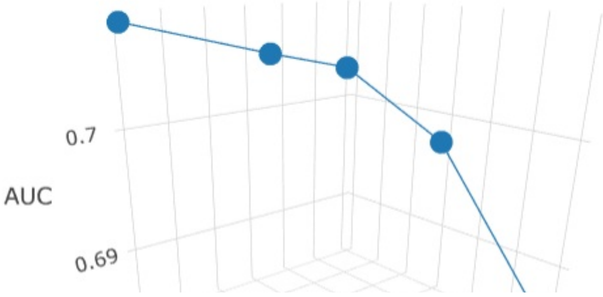
```
Results of analysis using Average Word2Vec vectorized text features merged with other features usi
ng gradient boosting classifier:
======================================================================
Optimal n_estimators Value:  700
========================================
Optimal max_depth Value:  1
========================================
AUC value of test data:  0.7138747545964869
========================================
Confusion Matrix :
============================================================
```

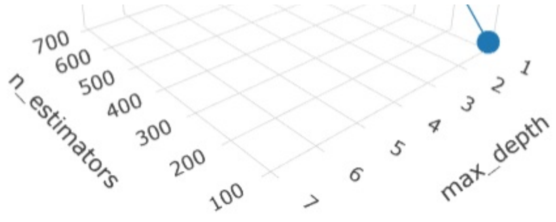


In [15]:

```
print("Cross-validation curve: ");
print("="*40);
display(Image(filename = "GBDT - Average Word2Vec.png", unconfined = True, width = '450px'));
```

```
Cross-validation curve: 
========================================
```



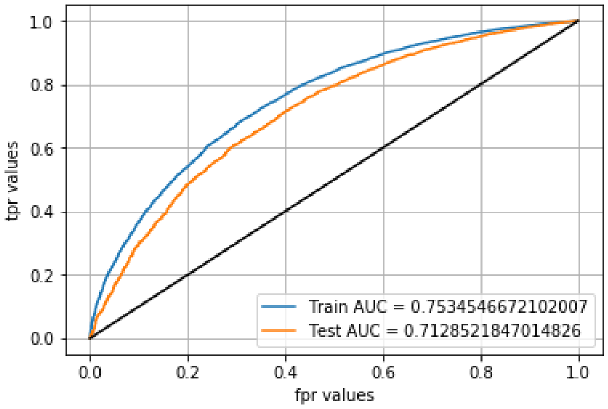


Classification using tf-idf weighted word2vec vectorized data by gradient boosting classifier

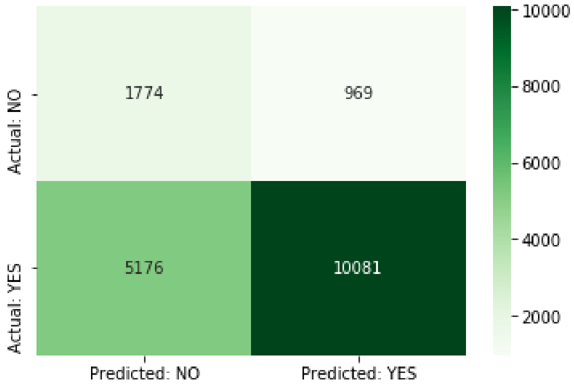
In [88]:

```
techniques = ['Tf-Idf Weighted Word2Vec'];
for index, technique in enumerate(techniques):
    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,\
                                     tfIdfWeightedWord2VecTitlesVectors,\
                                     tfIdfWeightedWord2VecEssaysVectors));
        crossValidateMergedData = hstack((crossValidateMergedData,\
                                 tfIdfWeightedWord2VecTitleTransformedCrossValidateData,\
                                 tfIdfWeightedWord2VecEssayTransformedCrossValidateData));
        testMergedData = hstack((testMergedData,\
                                 tfIdfWeightedWord2VecTitleTransformedTestData,\
                                 tfIdfWeightedWord2VecEssayTransformedTestData));

    trainingLength = trainingMergedData.shape[0];
    crossValidationLength = crossValidateMergedData.shape[0];
    totalLength = trainingLength+crossValidationLength;
    classesTrainingSub = np.concatenate((classesTraining.values, classesCrossValidate.values));

    trainingAndCrossValidateMergedData = vstack((trainingMergedData, crossValidateMergedData));
    gbdtClassifier = xgb.XGBClassifier(n_jobs = -1, min_samples_split = 500, reg_alpha = 1,
reg_lambda = 0, subsample = 0.5, colsample_bytree = 0.5, scale_pos_weight = 0.18);
    tunedParameters = {'n_estimators': [100, 250, 400, 500, 700], 'max_depth': [1, 3, 4, 5, 7]};
    classifier = GridSearchCV(gbdtClassifier, tunedParameters, cv = 5, scoring = 'roc_auc', n_jobs=
-1);
    classifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);

    testScoresDataFrame = pd.DataFrame(data =
np.hstack((classifier.cv_results_['param_n_estimators'].data[:, None],
classifier.cv_results_['param_max_depth'].data[:, None], classifier.cv_results_['mean_test_score']
[:, None], classifier.cv_results_['std_test_score'][:, None])), columns = ['n_estimators', 'max_dep
th', 'mts', 'stdts']);
    testScoresDataFrame = testScoresDataFrame.astype(float);

    crossValidateAucMeanValues = classifier.cv_results_['mean_test_score'];
    crossValidateAucStdValues = classifier.cv_results_['std_test_score'];

    trace1 = go.Scatter3d(x = tunedParameters['n_estimators'], y = tunedParameters['max_depth'], z
= crossValidateAucMeanValues, name = 'Cross-Validate');
    data = [trace1];
    layout = go.Layout(scene = dict(
            xaxis = dict(title='n_estimators'),
            yaxis = dict(title='max_depth'),
            zaxis = dict(title='AUC'),))

    fig = go.Figure(data=data, layout=layout)
    configure_plotly_browser_state()
    offline.iplot(fig, filename='3d-scatter-colorscale')

    optimalHypParamValue = classifier.best_params_['n_estimators'];
    optimalHypParam2Value = classifier.best_params_['max_depth'];
    gbdtClassifier = xgb.XGBClassifier(n_estimators = optimalHypParamValue, max_depth = optimalHypP
aram2Value, n_jobs = -1, min_samples_split = 500, reg_alpha = 1, reg_lambda = 0, subsample = 0.5, c
olsample_bytree = 0.5, scale_pos_weight = 0.18);
    gbdtClassifier.fit(trainingAndCrossValidateMergedData, classesTrainingSub);
    predScoresTraining = gbdtClassifier.predict_proba(trainingAndCrossValidateMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTrainingSub, predScoresTraining[:, 1]);
    predScoresTest = gbdtClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predScoresTest[:, 1]);
    predictionClassesTest = gbdtClassifier.predict(testMergedData);

    equalsBorder(70);
    plt.plot(fprTrain, tprTrain, label = "Train AUC = " + str(auc(fprTrain, tprTrain)));
    plt.plot(fprTest, tprTest, label = "Test AUC = " + str(auc(fprTest, tprTest)));
    plt.plot([0, 1], [0, 1], 'k-');
    plt.xlabel("fpr values");
    plt.ylabel("tpr values");
    plt.grid();
    plt.legend();
    plt.show();

    areaUnderRocValueTest = auc(fprTest, tprTest);

    print("Results of analysis using {} vectorized text features merged with other features using
gradient boosting classifier: ".format(technique));
    equalsBorder(70);
    print("Optimal n_estimators Value: ", optimalHypParamValue);
    equalsBorder(40);
    print("Optimal max_depth Value: ", optimalHypParam2Value);
    equalsBorder(40);
    print("AUC value of test data: ", str(areaUnderRocValueTest));
    # Predicting classes of test data projects
    predictionClassesTest = gbdtClassifier.predict(testMergedData);
    equalsBorder(40);
    # 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);
    sbrn.heatmap(confusionMatrixDataFrame, annot = True, fmt = 'd', cmap="Greens");
    plt.show();
```

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



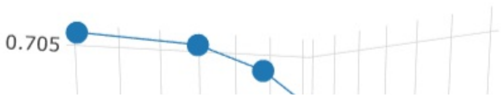
```
Results of analysis using Tf-Idf Weighted Word2Vec vectorized text features merged with other feat
ures using gradient boosting classifier:
======================================================================
Optimal n_estimators Value:  700
========================================
Optimal max_depth Value:  1
========================================
AUC value of test data:  0.7128521847014826
========================================
Confusion Matrix :
============================================================
```

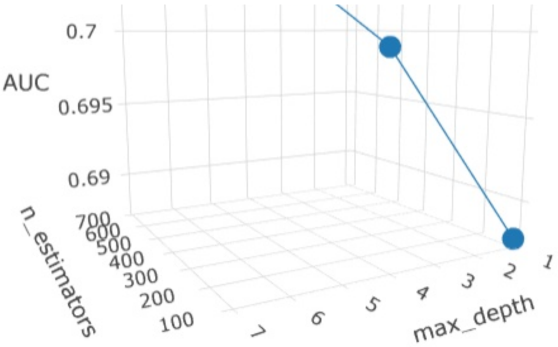


In [16]:

```
print("Cross-validation curve: ");
print("="*40);
display(Image(filename = "GBDT - Tf-Idf Weighted Word2Vec.png", unconfined = True, width = '450px')
);
```

```
Cross-validation curve:
========================================
```





Summary of results of above classification using random forests and gradient boosting classifier

In [89]:

```
techniques = ['Bag of words', 'Tf-Idf', 'Average Word2Vec', 'Tf-Idf Weighted Word2Vec', 'Bag of words', 'Tf-Idf', 'Average Word2Vec', 'Tf-Idf Weighted Word2Vec'];
aucValues = [0.7060, 0.7057, 0.6972, 0.7036, 0.7349, 0.7309, 0.7138, 0.7128]
models = ['Random Forests', 'Random Forests', 'Random Forests', 'Random Forests', 'Gradient Boosting - DT', 'Gradient Boosting - DT', 'Gradient Boosting - DT', 'Gradient Boosting - DT'];
nEstimatorsValues = [700, 250, 700, 700, 400, 700, 700, 700]
maxDepthValues = [12, 12, 12, 12, 3, 1, 1, 1]
for i,technique in enumerate(techniques):
    randomForestsAndGbdtResultsDataFrame = randomForestsAndGbdtResultsDataFrame.append({'Vectorizer': technique, 'Model': models[i], 'Max Depth': maxDepthValues[i], 'N Estimators': nEstimatorsValues[i], 'AUC': aucValues[i]}, ignore_index = True);
randomForestsAndGbdtResultsDataFrame
```

Out[89]:

	Vectorizer	Model	Max Depth	N Estimators	AUC
0	Bag of words	Random Forests	12	700	0.7060
1	Tf-Idf	Random Forests	12	250	0.7057
2	Average Word2Vec	Random Forests	12	700	0.6972
3	Tf-Idf Weighted Word2Vec	Random Forests	12	700	0.7036
4	Bag of words	Gradient Boosting - DT	3	400	0.7349
5	Tf-Idf	Gradient Boosting - DT	1	700	0.7309
6	Average Word2Vec	Gradient Boosting - DT	1	700	0.7138
7	Tf-Idf Weighted Word2Vec	Gradient Boosting - DT	1	700	0.7128

Conclusions of above analysis

1. The model trained using gradient boosting decision trees are better than models trained using random forests as you can observe from auc values.
2. It seems like by seeing above results table the model builded using gradient boosting and data containing bag of words vectorized text would be best compared to all other models .
3. The best depth and n_estimators would be 3 and 400 with gradient boosting decision trees.
4. The n-estimators in most of the cases is the max value of the list of values given for cross-validation which means we could get much better results if we increase n_estimators values but at the same time we might overfit.