Donors choose data analysis

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:** • `Music & The Arts`

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 [2]:

```
# 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 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 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 plotting decision tree
import graphviz
# For using svm classifer - hinge loss function of sgd
```

```
# For using SVM classifier - 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 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 be available.
  warnings.warn("The twython library has not been installed. "
```

```
[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 [4]:

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

Out[4]:

	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

	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 [5]:

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

Out[5]:

	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 [6]:

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

Out[6]:

	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 [7]:

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

Out[7]:

	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 [11]:

```
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 [13]:

```
# 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[13]:

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

In [14]:

```
# 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[14]:

```
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 [15]:

```
# 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[15]:

	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 [16]:

```
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 [17]:

```
# 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[17]:

```
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 [18]:

```
# 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[18]:

	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

subject_sub_categories	2192
TeamSports	
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 [19]:

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

Out[19]:

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

In [20]:

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

Out[20]:

	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 [21]:

```
projectsData.shape
```

Out[21]:

```
(109248, 20)
```

In [22]:

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

```
(109248, 22)
```

Out[22]:

	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 [23]:

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

Out[23]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	proje

	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	

In [24]:

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

Out[24]:

	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', "y
ou're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]);
def preProcessingWithAndWithoutStopWords(texts):
    """
    This function takes list of texts and returns preprocessed list of texts one with
    stop words and one without stopwords.
    """
    # Variable for storing preprocessed text with stop words
    preProcessedTextsWithStopWords = [];
    # Variable for storing preprocessed text without stop words
    preProcessedTextsWithoutStopWords = [];

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

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

        # https://stackoverflow.com/a/47091490/4084039
        # Replacing all below words with adverbs
```

```
        # 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 [26]:

```
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 [27]:

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

In [28]:

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

Out[28]:

```
['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 [29]:

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

Out[29]:

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

In [30]:

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

Out[30]:

```
(109248, 24)
```

Preparing data for classification and modelling

In [31]:

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

Out[31]:

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

	All features in projects data
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 decision trees

Classification using data by decision trees

Splitting Data(Only training and test)

In [32]:

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

Out[32]:

```
(109245, 24)
```

In [33]:

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

```
(109245,)
```

In [0]:

```
trainingData, testData, classesTraining, classesTest = model_selection.train_test_split(projectsDat
a, classesData, test_size =  0.3, random_state = 44, stratify = classesData);
trainingData, crossValidateData, classesTraining, classesCrossValidate =
```

```
model_selection.train_test_split(trainingData, classesTraining, test_size = 0.3, random_state = 0,
stratify = classesTraining);
```

In [35]:

```
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:  (32774, 24)
classesTest:  (32774,)
trainingData shape:  (53529, 24)
classesTraining shape:  (53529,)
```

In [36]:

```
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:  (8105, 24)
Number of positive points:  (45424, 24)
```

In [0]:

```
vectorizedFeatureNames = [];
```

Vectorizing categorical data

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

In [0]:

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

In [39]:

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

```
Features used in vectorizing categories:
======================================================================
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
```

```
['Health_Sports', 'Math_Science', 'Literacy_Language']
==========================================================================
Shape of cleaned_categories matrix after vectorization(one-hot-encoding):  (53529, 9)
==========================================================================
Sample vectors of categories:
==========================================================================
  (0, 0) 1
  (0, 1) 1
  (1, 8) 1
  (2, 7) 1
  (3, 7) 1
```

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

In [0]:

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

In [41]:

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

```
Features used in vectorizing subject sub categories:
==========================================================================
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
==========================================================================
Shape of cleaned_categories matrix after vectorization(one-hot-encoding):  (53529, 30)
==========================================================================
Sample vectors of categories:
==========================================================================
  (0, 8) 1
  (0, 9) 1
  (1, 29) 1
  (2, 28) 1
  (3, 28) 1
```

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 [43]:

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

Out[43]:

```
Counter({'Dr.': 9, 'Mr.': 5268, 'Mrs.': 28095, 'Ms.': 19006, 'Teacher': 1151})
```

In [0]:

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

In [45]:

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

```
Features used in vectorizing teacher_prefix:
======================================================================
['Ms.', 'Teacher', 'Mrs.', 'Mr.', 'Dr.']
======================================================================
Shape of teacher_prefix matrix after vectorization(one-hot-encoding):  (53529, 5)
======================================================================
Sample vectors of teacher_prefix:
======================================================================
  (3, 1) 1
  (4, 1) 1
```

In [46]:

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

Out[46]:

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

In [47]:

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

Out[47]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime
48858	140830	p259363	4800dff2ebf45d898fd462acb9a5cfe4	Ms	FL	2016-11-17 21:11:22
37065	172594	p069087	c66949c3fbb4a283d7e2b3f384f2ab54	Ms	DE	2016-10-11 16:08:15

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime
38285	116819	p140400	e600536514113db6349fbe7bbc16ca4b	Ms	TX	2016-07-06 12:45:17

In [0]:

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

In [49]:

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

```
Features used in vectorizing teacher_prefix:
======================================================================
['Ms', 'Teacher', 'Mrs', 'Mr', 'Dr']
======================================================================
Shape of teacher_prefix matrix after vectorization(one-hot-encoding):  (53529, 5)
======================================================================
Sample vectors of teacher_prefix:
======================================================================
  (0, 0) 1
  (1, 0) 1
  (2, 0) 1
  (3, 1) 1
```

4. Vectorizing school_state - One Hot Encoding

In [0]:

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

In [51]:

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

```
equalsBorder(70);
print("Sample vectors of school_state: ");
equalsBorder(70);
print(schoolStateVectors[0:4]);
```

```
Features used in vectorizing school_state:
======================================================================
['FL', 'DE', 'TX', 'WV', 'PA', 'NJ', 'WA', 'AR', 'IA', 'CA', 'NH', 'NV', 'LA', 'MD', 'GA', 'AZ', 'N
C', 'AL', 'OK', 'AK', 'NY', 'IL', 'VA', 'MA', 'OR', 'SC', 'MO', 'RI', 'IN', 'HI', 'MI', 'OH', 'TN',
'CO', 'WY', 'MN', 'UT', 'ME', 'WI', 'MT', 'KY', 'SD', 'NM', 'KS', 'CT', 'MS', 'NE', 'DC', 'ND', 'II
', 'VT']
======================================================================
Shape of school_state matrix after vectorization(one-hot-encoding):  (53529, 51)
======================================================================
Sample vectors of school_state:
======================================================================
  (0, 0) 1
  (1, 1) 1
  (2, 2) 1
  (3, 3) 1
```

5. Vectorizing project_grade_category - One Hot Encoding

In [52]:

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

Out[52]:

```
Counter({'3-5': 18284,
         '6-8': 8161,
         '9-12': 5424,
         'Grades': 53529,
         'PreK-2': 21660})
```

In [53]:

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

Out[53]:

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

In [54]:

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

Out[54]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetim
48858	140830	p259363	4800dff2ebf45d898fd462acb9a5cfe4	Ms	FL	2016-11-17 21:11:22
37065	172594	p069087	c66949c3fbb4a283d7e2b3f384f2ab54	Ms	DE	2016-10-11 16:08:15

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetim
38285	116819	p140400	e600536514113db6349fbe7bbc16ca4b	Ms	TX	2016-07-06 12:45:17
102612	48244	p174203	21fe8971971fbaca9e39d2013939dc8c	Teacher	WV	2017-01-03 13:01:49

In [0]:

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

In [56]:

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

```
Features used in vectorizing project_grade_category:
======================================================================
['Grades3to5', 'Grades6to8', 'GradesPreKto2', 'Grades9to12']
======================================================================
Shape of school_state matrix after vectorization(one-hot-encoding):  (53529, 4)
======================================================================
Sample vectors of school_state:
======================================================================
  (0, 0) 1
  (1, 0) 1
  (2, 1) 1
  (3, 1) 1
```

Vectorizing Text Data

In [57]:

```
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 [60]:

```
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', 'worst', 'worth', 'worthwhile', 'worthy', 'would', 'wow', 'write'
, 'writer', 'writers', 'writing', 'writings', 'written', 'wrong', 'wrote', 'xylophones', 'yard', '
year', 'yearbook', 'yearly', 'yearn', 'yearning', 'years', 'yes', 'yesterday', 'yet', 'yoga', 'yor
k', 'young', 'younger', 'youngest', 'youth', 'youtube', 'zero', 'zest', 'zip', 'zone', 'zoo']
======================================================================
Shape of preprocessed essay matrix after vectorization:  (53529, 5000)
======================================================================
Sample bag-of-words vector of preprocessed essay:
======================================================================
  (0, 2747) 2
  (0, 4365) 10
  (0, 3100) 9
  (0, 2036) 1
  (0, 395) 1
  (0, 3220) 1
  (0, 3960) 6
  (0, 4907) 1
  (0, 2641) 1
  (0, 1676) 2
  (0, 615) 1
  (0, 2979) 1
  (0, 4456) 1
  (0, 2686) 1
  (0, 3890) 1
  (0, 4967) 8
  (0, 2754) 1
  (0, 4479) 1
  (0, 3580) 2
  (0, 3821) 2
  (0, 3045) 2
  (0, 4390) 1
  (0, 1989) 1
  (0, 250) 1
  (0, 3066) 1
  : :
  (0, 3256) 1
  (0, 571) 1
  (0, 437) 1
  (0, 4191) 1
  (0, 4873) 1
  (0, 4112) 1
  (0, 2266) 1
  (0, 2738) 1
  (0, 3819) 1
  (0, 433) 1
  (0, 553) 1
  (0, 434) 2
  (0, 4752) 1
  (0, 4016) 1
  (0, 900) 1
  (0, 3393) 1
```

```
  (0, 3393) 1
  (0, 438) 1
  (0, 4003) 1
  (0, 2542) 1
  (0, 3519) 1
  (0, 2547) 1
  (0, 2795) 1
  (0, 3991) 1
  (0, 4388) 1
  (0, 3022) 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 [62]:

```
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:
======================================================================
['wish', 'within', 'without', 'wizards', 'wo', 'wobble', 'wobbles', 'wobbling', 'wobbly',
'wonder', 'wonderful', 'wonders', 'word', 'words', 'work', 'workers', 'working', 'workout',
'works', 'workshop', 'world', 'worlds', 'worms', 'worth', 'would', 'wow', 'wrestling', 'write', 'w
riter', 'writers', 'writing', 'written', 'ye', 'year', 'yearbook', 'yes', 'yoga', 'young',
'youth', 'zone']
======================================================================
Shape of preprocessed title matrix after vectorization:  (53529, 2105)
======================================================================
Sample bag-of-words vector of preprocessed title:
======================================================================
  (0, 127) 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 [64]:

```
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', 'worst', 'worth', 'worthwhile', 'worthy', 'would', 'wow', 'write'
, 'writer', 'writers', 'writing', 'writings', 'written', 'wrong', 'wrote', 'xylophones', 'yard', '
year', 'yearbook', 'yearly', 'yearn', 'yearning', 'years', 'yes', 'yesterday', 'yet', 'yoga', 'yor
k', 'young', 'younger', 'youngest', 'youth', 'youtube', 'zero', 'zest', 'zip', 'zone', 'zoo']
==========================================================================
Shape of preprocessed title matrix after tf-idf vectorization:  (53529, 5000)
==========================================================================
Sample Tf-Idf vector of preprocessed essay:
==========================================================================
  (0, 3022) 0.018094205612536794
  (0, 4388) 0.054207619003056214
  (0, 3991) 0.12495528339505618
  (0, 2795) 0.05826650257254188
  (0, 2547) 0.07602232720617802
  (0, 3519) 0.11547298391896377
  (0, 2542) 0.04910401209544946
  (0, 4003) 0.08690283700318722
  (0, 438) 0.08767512357315305
  (0, 3393) 0.09389789515024273
  (0, 900) 0.06949010829630545
  (0, 4016) 0.09458467882329553
  (0, 4752) 0.03083134772248111
  (0, 434) 0.20173224818799645
  (0, 553) 0.11575612507065738
  (0, 433) 0.09290351365633924
  (0, 3819) 0.07714195112279197
  (0, 2738) 0.06274668181708029
  (0, 2266) 0.09193907009115258
  (0, 4112) 0.07852645809939311
  (0, 4873) 0.09240222208320098
  (0, 4191) 0.05924344370687216
  (0, 437) 0.08939357156701812
  (0, 571) 0.05447374482975269
  (0, 3256) 0.13110922135539982
  : :
  (0, 3066) 0.034823913503280624
  (0, 250) 0.04846896890299715
  (0, 1989) 0.052488080232627465
  (0, 4390) 0.04865171440825632
  (0, 3045) 0.05770816524873697
  (0, 3821) 0.0946756617112588
  (0, 3580) 0.07578658579302004
  (0, 4479) 0.0426445957954258
  (0, 2754) 0.03197297735421936
  (0, 4967) 0.2819748929618572
  (0, 3890) 0.053934797345609964
  (0, 2686) 0.08047228671532597
  (0, 4456) 0.10886145404533437
  (0, 2979) 0.06356358894328347
  (0, 615) 0.06940477956922898
  (0, 1676) 0.08905604423804019
  (0, 2641) 0.0252857123904684
  (0, 4907) 0.07757726633148239
  (0, 3960) 0.12064159210540859
  (0, 3220) 0.05647317289665949
  (0, 395) 0.0657736372792621
  (0, 2036) 0.037438676737201124
  (0, 3100) 0.22655991977583992
  (0, 4365) 0.17450267413635387
  (0, 2747) 0.11752449186442786
```

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 [66]:

```
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:
======================================================================
['wish', 'within', 'without', 'wizards', 'wo', 'wobble', 'wobbles', 'wobbling', 'wobbly',
'wonder', 'wonderful', 'wonders', 'word', 'words', 'work', 'workers', 'working', 'workout',
'works', 'workshop', 'world', 'worlds', 'worms', 'worth', 'would', 'wow', 'wrestling', 'write', 'w
riter', 'writers', 'writing', 'written', 'ye', 'year', 'yearbook', 'yes', 'yoga', 'young',
'youth', 'zone']
======================================================================
Shape of preprocessed title matrix after tf-idf vectorization:  (53529, 2105)
======================================================================
Sample Tf-Idf vector of preprocessed title:
======================================================================
  (0, 127) 1.0
```

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 [68]:

```
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 [70]:

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

In [71]:

```
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: 53529,300
==================================================================
Sample essay:
==================================================================
lot students not get lot attention outside school willing learn excited school students bright mot
ivated talent light room would love teacher provide resources need successful future students
always excited new activities would not able home would like provide opportunities school would
not able experience home half school students receive free reduced cost lunches happens weekends c
hildren not school not anything eat home would go without school noticing unusual trend among stud
ents free lunch program growing number conserving food fridays students hoarding food almost like
```

animals eating half sticking rest pants book bag would something eat weekend simply not resources house students looking resource would not go without backpack blessing program sends home food students weekends not go without backpacks use send food home students lately food comes home plastic bags not seem like students would able keep need private backpacks key making program secret success nannan

==========

Word2Vec vector of sample essay:

==========

```
[ 2.64178445e-02  4.11585247e-02 -3.65931370e-02 -9.86259253e-02
  3.11363322e-02  5.02330031e-02 -3.25032849e+00 -1.60717418e-02
 -1.68263788e-02 -1.77343297e-01  1.33223413e-01  5.28117959e-03
  1.59214204e-01 -2.12204749e-01 -5.74203253e-02  1.89232041e-02
 -2.24438164e-02  3.28739110e-03  6.52792692e-02 -2.35127493e-02
  1.46092384e-02 -4.35153082e-03  6.25173671e-03  7.00009370e-02
 -5.82035000e-02 -3.51780562e-02  2.19294829e-02 -6.84360548e-02
 -5.10377130e-02 -7.60905137e-02 -3.03812705e-01 -3.52655733e-02
  1.00956477e-01  7.93720548e-02 -8.16208219e-04  1.16886685e-02
 -1.35053868e-01  1.35777671e-03  3.14055322e-02 -5.49302336e-02
 -5.48055443e-02  9.25238364e-02  7.57590685e-03 -1.99200192e-01
 -1.35755001e-02 -1.79084225e-02  9.40091836e-02 -7.65994315e-02
 -2.32081438e-02 -1.35397847e-01 -7.34642952e-02  7.52849845e-02
  1.73943973e-02  3.69102322e-02  1.15603771e-01 -7.41144671e-02
  7.18730671e-02  8.92285342e-03 -5.35301267e-02  8.50206007e-02
 -1.14478162e-01 -7.81226199e-02  7.75593959e-02 -8.83025685e-03
 -6.17028445e-02  1.31430232e-01  2.25251849e-02 -5.03001932e-02
  1.37625762e-01 -1.45350101e-01 -1.70897788e-01  1.14343000e-02
  3.23514332e-02 -1.13768785e-01 -2.07520726e-02 -1.19485522e-01
  8.33332089e-02  1.14648185e-02  7.34454027e-03  2.78517171e-02
  4.67122192e-02 -3.24567178e-01 -2.21318082e-02  2.97744034e-02
 -1.19149077e-01  2.65314651e-02  1.04533288e-01 -3.14441703e-02
  1.83888242e-01 -1.52575132e-02  1.75732736e-02  3.17310185e-02
 -5.71921740e-02  1.44338822e-02 -3.77974966e-02 -3.96341448e-01
 -2.14370268e+00 -4.19776882e-02  3.38579945e-02  4.76397658e-02
 -1.21946686e-01  1.98988534e-02  1.45189086e-01 -5.49774260e-02
  2.34929137e-02  4.58554856e-02  2.23384144e-02 -2.11502348e-01
  5.69698240e-02  3.41814151e-02 -3.51186575e-02 -4.40401651e-02
  8.71213425e-02  1.98191901e-01  1.43586726e-02  5.17909416e-02
 -2.03039199e-01  9.66194178e-03  1.18659580e-01 -1.90723002e-02
  1.03781212e-01  3.74469568e-02  1.37180137e-03 -1.07355573e-01
 -3.38622466e-03  1.58037500e-02  1.36346394e-01 -9.83904521e-02
  7.56157849e-03  1.29017001e-01  3.32813973e-02 -6.19366507e-03
 -3.35707123e-02 -3.98648014e-02  5.69754603e-02 -1.02293933e-01
  1.47271221e-01 -3.11733095e-02  8.77894904e-02  3.94942363e-01
  2.74282082e-02 -2.35693938e-02 -5.66339726e-02 -2.08486096e-02
 -7.31347616e-02 -5.51548315e-02  6.85621651e-02 -3.94897493e-02
  1.70319799e-01  2.18439705e-03 -1.24370685e-03  6.26751445e-02
  7.12478562e-02 -3.48474726e-02  1.01263157e-01 -1.60795267e-03
 -2.47651171e-02 -6.37218695e-02 -4.31363233e-02  1.85468192e-02
 -3.70014363e-02 -1.10632411e-02 -9.40172411e-02 -5.45686849e-03
 -8.92796116e-02 -1.66452205e-02  1.16005548e-02  3.55625911e-02
  5.52277995e-02 -4.39432658e-02 -4.23908868e-02 -3.22058596e-02
 -6.77250342e-02 -7.18981062e-02 -6.90683040e-02  6.30475644e-02
  5.11858073e-02  4.94661712e-02 -1.61644806e-01 -7.79537038e-02
  2.70485658e-02  2.88038810e-01  4.62749110e-02  3.77251257e-02
 -9.81174548e-02 -1.00831295e-01 -6.59818693e-02 -2.92114616e-02
  1.31811366e-01  2.80260274e-05  2.94397031e-02 -1.20445534e-02
 -2.80758384e-02 -6.21983836e-02  3.16266137e-02 -1.28044112e-01
 -5.81397521e-02  4.49531027e-03  7.58185616e-03 -4.29943610e-03
  1.38576618e-01  1.67571096e-03 -8.27000068e-02  6.70286667e-02
 -9.58940281e-02  2.90674801e-02  9.62612795e-02 -1.69816938e-01
  2.17541666e-01 -3.39641068e-02  9.34559404e-02  4.81947546e-02
 -1.17510205e-03 -1.43365913e-01 -5.08283989e-02  3.29708153e-02
 -1.23135701e-01 -8.79702740e-02 -4.43972041e-02 -5.46582226e-02
 -1.09578452e-01  8.89863014e-05 -7.50223749e-02 -9.88918493e-02
 -2.39510208e+00  6.54201329e-02  3.05463164e-02  7.55921712e-02
 -1.37815116e-02 -1.07346432e-01  7.64600774e-02  1.12853164e-02
 -9.88566466e-02 -7.83722048e-02 -1.36292599e-01  1.17159290e-01
  1.21745427e-01 -1.00550464e-01  9.31926130e-02  1.38560681e-01
 -1.22690287e-01 -3.05568329e-02 -2.60020263e-01  1.80501185e-01
 -5.81790267e-02  2.38550760e-02 -1.36753425e-03 -6.47717221e-02
  8.77596781e-02  3.52596301e-03 -1.09310096e-02 -3.05815349e-02
  1.32019466e-01 -8.18688678e-02  1.08905774e-01  1.47775548e-03
  1.59440881e-01 -7.67154795e-03  8.75721027e-02 -3.26705301e-02
 -4.50245603e-02 -1.51143247e-02  6.56053185e-02 -4.27600719e-02
  6.04811575e-02  1.47913947e-02 -1.47595157e-01  8.22455822e-03
  4.36072377e-02  1.00413420e-01 -2.81824658e-03 -7.47502019e-02
 -1.52511845e-01  1.40392000e-02 -6.99392968e-02 -8.65833082e-02
```

```
  1.14588835e-01 -2.05179071e-02 -2.51619041e-02  7.10258636e-02
  3.27740903e-01 -8.62650644e-02 -2.64754248e-02  4.36036233e-02
 -2.24391747e-02  1.79602370e-01  3.05130630e-02  1.28489692e-01
  5.49444507e-02 -7.92351644e-02 -1.11680753e-02 -1.31574692e-02
 -1.13593021e-01 -9.27983856e-02  2.94359925e-02  4.27592764e-02
 -7.66843014e-02  1.03615236e-01  1.00204215e-01 -3.80285137e-02]
```

2. Vectorizing project_title

In [72]:

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

In [73]:

```
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: 53529, 300
======================================================================
Sample title:
======================================================================
backpack blessing
======================================================================
Word2Vec vector of sample title:
======================================================================
[ 0.321755   -0.032325    0.20554    -0.600965   -0.07439    -0.001725
 -1.5139     -0.013643   -0.381111   -0.34155     0.301139   -0.32066
  0.2462865  -0.415415    0.08197     0.0296465   0.26638     0.3739
  0.1336645   0.26477     0.507595    0.1300295   0.1349275   0.357985
  0.0574315   0.0829305  -0.25652     0.660165    0.150822    0.444865
 -0.3268345  -0.0447      0.06186     0.25688     0.44128    -0.075689
 -0.23253    -0.187925    0.410855   -0.12419     0.02233    -0.045325
 -0.1647205   0.2246      0.32102    -0.12459     0.133932   -0.21097
 -0.057329   -0.033065    0.017981    0.012177    0.382856   -0.0979925
  0.199735   -0.265595   -0.017795   -0.146825    0.115415   -0.570505
 -0.180215   -0.3856425  -0.286265   -0.22614    -0.296295    0.365795
 -0.0152215   0.143773    0.41466     0.252065    0.010353    0.08573
 -0.140455    0.14587572 -0.124965    0.140665    0.08035     0.27449
  0.219185    0.060825   -0.1167135   0.376505   -0.03019     0.296875
 -0.133196   -0.19192     0.109945    0.211763    0.44041    -0.1450485
 -0.15437     0.169675    0.095897   -0.0569235   0.304295   -0.171975
 -1.1615     -0.35454     0.0778515  -0.008015    0.279975   -0.11542
  0.52091    -0.226875    0.44208    -0.164374   -0.15921     0.11959
 -0.061678   -0.25898    -0.1430615  -0.20686    -0.032085    0.03297
  0.232025    0.296785   -0.35757     0.171727    0.21386    -0.631595
  0.30527     0.224278   -0.025359   -0.08231955 -0.253935    0.2451645
  0.124955   -0.245494    0.161938    0.04664    -0.188875   -0.443275
 -0.092984   -0.126575    0.010865   -0.01447065  0.0382655   0.104584
  0.418712    0.232405   -0.1713015   0.073615    0.55051    -0.47407
 -0.359055    0.169825    0.21163    -0.1005415   0.64908     0.13708
  0.094981    0.52086    -0.164547   -0.09116225  0.173089    0.038771
 -0.41934     0.1254365   0.561255    0.112135   -0.59007    -0.027535
 -0.377775    0.2140365  -0.33847     0.28952    -0.1682125   0.097907
  0.24157145 -0.310265    0.273239   -0.12434325 -0.362965    0.243995
  0.053355    0.13819     0.535475    0.36646    -0.349505    0.1521995
 -0.01774435  0.361915   -0.0814385  -0.109861    0.3147     -0.199745
 -0.03939    -0.46605    -0.264485   -0.2457385   0.1240665  -0.01222
  0.082725    0.299539    0.322355   -0.192285   -0.062955    0.1146
  0.22345     0.1059585   0.2958455   0.211755    0.184275    0.278205
 -0.01611    -0.227675   -0.09526    -0.2508515   0.20571    -0.4398585
 -0.0940565  -0.11471     0.125921    0.42353    -0.030515    0.363465
 -0.1392805  -0.1267485  -0.1880414  -0.069784    0.3439      0.32631
  0.37406     0.17519    -1.6642     -0.51405    -0.480135    0.01252
```

```
 -0.118549    0.2967225  -0.049375   -0.30497    -0.54004    -0.17487
  0.070924    0.28063    -0.090985   -0.362275    0.258784    0.27267
  0.22494201  0.094035   -0.3873275   0.569105   -0.497435    0.211685
 -0.14664    -0.13349271  0.125245    0.1574925  -0.0122695  -0.0566175
  0.2816425   0.07031     0.2059138   0.17325     0.1300675   0.298584
 -0.24331    -0.104145   -0.082855   -0.007065   -0.286076    0.00182
 -0.10138     0.502835    0.032735    0.026995    0.04304     0.15826999
  0.0966835  -0.04445    -0.716135   -0.072346   -0.019113   -0.117785
  0.006371    0.0346615   0.06374325 -0.01002     0.222354    0.183061
  0.095995    0.25149     0.3785875  -0.21892    -0.0871865   0.631515
 -0.157546    0.02034    -0.122824    0.22827     0.161612   -0.239607
  0.10694     0.3348141  -0.286185   -0.01752     0.1474715   0.295965  ]
```

Tf-Idf Weighted Word2Vec Vectorization

1. Vectorizing project_essay

In [0]:

```
# 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 [75]:

```
# 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 [76]:

```
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: 53529, 300
==========================================================================
Sample Essay:
==========================================================================
lot students not get lot attention outside school willing learn excited school students bright mot
ivated talent light room would love teacher provide resources need successful future students
always excited new activities would not able home would like provide opportunities school would
not able experience home half school students receive free reduced cost lunches happens weekends c
hildren not school not anything eat home would go without school noticing unusual trend among stud
ents free lunch program growing number conserving food fridays students hoarding food almost like
animals eating half sticking rest pants book bag would something eat weekend simply not resources
house students looking resource would not go without backpack blessing program sends home food stu
dents weekends not go without backpacks use send food home students lately food comes home plastic
bags not seem like students would able keep need private backpacks key making program secret succe
ss nannan
==========================================================================
Tf-Idf Weighted Word2Vec vector of sample essay:
==========================================================================
[ 2.87245642e-02  5.93005468e-02 -3.47539398e-02 -9.59522805e-02
  6.21683562e-02  2.94011995e-02 -3.29206558e+00 -2.70379313e-02
 -9.98258490e-03 -2.11481777e-01  1.74833111e-01 -2.64674200e-02
  1.38102913e-01 -2.57853328e-01 -3.45596278e-02  2.49728349e-02
 -3.64500658e-02 -7.74601448e-03  1.16697602e-01 -4.51907447e-02
  4.01455362e-03 -1.51334738e-02 -1.14713435e-02  6.76244571e-02
 -1.26209135e-01 -6.61193603e-02  9.81270735e-03 -1.11629936e-01
 -6.96764704e-02 -9.43370295e-02 -3.21979779e-01 -3.68805680e-02
  1.34004952e-01  7.42488439e-02  5.06548918e-03  4.27971940e-02
 -1.70615628e-01 -2.32722297e-02  7.13160199e-02 -8.05238140e-02
 -2.06444227e-02  1.06968579e-01  1.40058832e-02 -1.93621463e-01
 -3.33402649e-02 -1.63772601e-02  1.02684860e-01 -9.26284587e-02
 -4.99248651e-02 -1.79388542e-01 -9.89649311e-02  9.34402336e-02
  4.01817391e-02  8.00903962e-02  1.35060096e-01 -1.06200578e-01
  3.94260823e-02  2.84382871e-02 -8.17757845e-02  4.24312431e-02
 -1.44865602e-01 -1.08732161e-01  9.58575118e-02  8.33713632e-04
 -4.98348000e-02  1.30196379e-01  4.20994901e-02 -4.74041001e-02
  1.31569393e-01 -1.79163736e-01 -2.21257268e-01  6.76405771e-02
  1.11530369e-01 -1.11315628e-01 -1.20334173e-02 -8.26322282e-02
  9.21714455e-02 -2.72817436e-02  5.86181784e-03  6.26320975e-02
  7.03948184e-02 -3.65113223e-01 -4.23990497e-03  5.06695934e-02
 -9.51023697e-02  1.82768053e-02  6.23771643e-02  3.78987446e-03
  2.11626499e-01 -1.21834526e-02  3.95358905e-03  4.16094599e-02
 -1.12238328e-01 -8.75880552e-03 -4.40750428e-02 -4.68836401e-01
 -2.14798796e+00 -9.58387216e-02  1.58611556e-02  3.79662990e-02
 -1.34192575e-01  2.65678967e-02  1.33542277e-01 -5.95021504e-02
 -1.08778771e-02  5.65156016e-02  2.80905194e-02 -2.40965845e-01
  8.97111220e-02  2.20681417e-02 -1.01242265e-02 -6.27890902e-02
  1.17872649e-01  1.90343999e-01  5.35585750e-03  4.80991907e-02
 -2.02271180e-01 -1.53763132e-02  9.82291571e-02 -5.51687285e-02
  1.21823704e-01  1.62688027e-02 -2.84852560e-02 -1.43344497e-01
 -2.60353593e-02  1.56416484e-02  1.83609256e-01 -1.03059313e-01
 -7.37123794e-03  1.52935230e-01  1.52920445e-02 -2.53192116e-02
 -4.58220424e-02 -2.25816885e-02  7.32616353e-02 -1.02588559e-01
  1.64419803e-01 -5.73094685e-02  1.12104411e-01  4.30298683e-01
 -4.99231372e-03 -1.95134573e-02 -8.47117741e-02 -4.88257787e-02
 -3.95367850e-02 -5.84603001e-02  1.36311961e-01 -3.11409143e-02
  2.32064150e-01 -2.10744757e-02  5.49507648e-03  1.01063012e-01
  5.17837899e-02 -3.16628819e-02  9.18126473e-02 -9.33139686e-03
 -2.39186665e-02 -9.44940271e-02 -5.86675589e-02  5.53351272e-02
 -3.18912718e-02 -2.50482240e-02 -1.21320867e-01 -6.46038072e-03
 -1.16816969e-01 -3.82792208e-02  3.41745543e-02  2.97571189e-02
  5.13189714e-02 -2.17698348e-02 -7.36774060e-03 -6.41784664e-02
 -8.71120123e-02 -2.37978682e-02 -9.46445169e-02  7.76271515e-02
  7.15078267e-02  6.64144482e-02 -1.70544822e-01 -1.02757267e-01
  1.59219133e-02  3.12091063e-01  4.56132120e-02  7.23347086e-02
 -1.14819378e-01 -1.33555272e-01 -4.27948869e-02 -3.79465247e-02
  1.43638534e-01  3.66869516e-02  7.17301764e-02  8.60961120e-03
  2.63249328e-02 -8.69078190e-02  2.80793660e-02 -1.55152424e-01
 -6.19196796e-02  1.25976026e-02  4.17518112e-02 -1.76960292e-02
  1.10058965e-01 -9.96881980e-03 -8.36898789e-02  6.09153523e-02
 -1.07768771e-01  4.99643839e-02  1.08617031e-01 -1.77549052e-01
  2.72630437e-01 -4.97003514e-02  1.44976733e-01  6.65341880e-02
 -4.08312127e-02 -1.48960142e-01 -2.12999733e-02  5.30607857e-02
 -1.41022826e-01 -1.19398705e-01 -9.47219132e-02 -7.00181708e-02
 -1.40425935e-01  1.33524606e-02 -6.16619247e-02 -1.16965329e-01
 -2.54929662e+00  1.30487222e-01  4.23568409e-02  7.18932061e-02
```

```
 6.59392918e-03 -1.27994346e-01  1.22783294e-01  9.08005539e-03
-1.21135195e-01 -1.14201642e-01 -1.35879901e-01  1.55846388e-01
 1.21247221e-01 -1.25806635e-01  1.10094720e-01  1.37293357e-01
-1.82538263e-01 -4.60953629e-02 -3.16675537e-01  1.86584922e-01
-4.71928061e-02  1.37149262e-03  4.65403904e-02 -4.78853014e-02
 1.65289634e-01  3.34052618e-02 -1.27130835e-02 -5.30255235e-02
 1.45479295e-01 -7.79300767e-02  1.20848360e-01  6.42753827e-03
 1.74801950e-01 -5.44340707e-03  1.04012342e-01  1.29326309e-02
-3.30780628e-02 -4.04972259e-02  8.62701524e-02 -5.48549731e-02
 7.68602062e-02  3.49803496e-02 -1.37250679e-01 -1.30360463e-02
 3.67492519e-02  9.26716364e-02  9.27717202e-03 -9.76967787e-02
-1.26998933e-01  6.30327649e-03 -8.00194244e-02 -1.29442389e-01
 1.17833277e-01 -2.27582978e-02 -3.90298603e-02  8.63480666e-02
 3.81650028e-01 -9.90373356e-02 -9.41248427e-03  3.05024181e-02
-3.69208838e-02  1.87680136e-01  6.35882712e-02  1.66003515e-01
 7.86419548e-02 -1.41743237e-01 -1.70778827e-02 -3.58563801e-03
-1.43739901e-01 -9.59063040e-02  5.62382404e-02  4.63013780e-02
-1.32824081e-01  6.33643179e-02  8.64734451e-02 -7.89997752e-02]
```

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 [78]:

```
# 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 [79]:

```
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: 53529, 300
======================================================================
Sample Title:
======================================================================
backpack blessing
======================================================================
Tf-Idf Weighted Word2Vec vector of sample title:
======================================================================
[ 2.50718268e-01 -6.78771436e-02  1.64945903e-01 -5.58781892e-01
 -1.57515554e-01  2.84074698e-02 -1.56501858e+00 -4.22254254e-03
 -3.41084483e-01 -3.32777248e-01  2.55680825e-01 -2.95874949e-01
  2.25708247e-01 -4.39503940e-01  5.08571015e-02  1.84672039e-02
  2.55236361e-01  3.44834899e-01  1.38834573e-01  3.15915601e-01
  5.32437032e-01  1.48715861e-01  1.56665232e-01  3.55325683e-01
  6.66728486e-02  9.02252070e-02 -3.47335076e-01  6.22555660e-01
  1.37913942e-01  4.43856374e-01 -3.54948602e-01 -7.44582738e-02
  3.72923004e-02  2.64479057e-01  4.40914616e-01 -9.30019969e-02
 -1.93378643e-01 -1.85749729e-01  3.86529911e-01 -1.54445244e-01
  5.08288135e-02 -1.67128115e-02 -1.34640371e-01  2.18770297e-01
  2.99576279e-01 -1.26725913e-01  1.40386499e-01 -1.98682038e-01
 -6.87065553e-02 -9.17226300e-03  3.15835235e-02  3.27877211e-03
  4.33418161e-01 -9.36751477e-02  1.96288520e-01 -2.55749005e-01
 -4.04188778e-02 -1.47697341e-01  8.79870802e-02 -5.55966575e-01
 -1.87377593e-01 -3.47990571e-01 -2.25839691e-01 -2.28737637e-01
 -3.14018496e-01  3.47589807e-01 -1.99603955e-02  1.32415887e-01
  3.92148322e-01  3.01179839e-01 -1.44116413e-03  2.55003065e-02
 -1.13843615e-01  1.62940030e-01 -9.58499673e-02  1.75641457e-01
  4.44894531e-02  3.34517616e-01  2.23573726e-01  2.09669591e-02
 -1.38230827e-01  3.14182276e-01  1.39392796e-02  2.99018550e-01
 -1.06636426e-01 -1.98486347e-01  1.09708264e-01  1.94496293e-01
  4.37369437e-01 -1.27378284e-01 -1.95090983e-01  1.67646588e-01
  1.17409216e-01 -3.85125286e-02  2.66088826e-01 -1.25151157e-01
 -1.15904452e+00 -3.29949978e-01  8.06757105e-02 -2.63858496e-02
  2.81811909e-01 -8.10127669e-02  4.79992814e-01 -2.37925237e-01
  4.36551063e-01 -1.42739245e-01 -1.55492712e-01  9.22778048e-02
 -7.80429946e-02 -2.65688506e-01 -1.34687205e-01 -2.03062821e-01
 -5.76484017e-02  8.40439342e-02  2.39624644e-01  3.00834188e-01
 -4.08343168e-01  1.88109735e-01  2.06361982e-01 -5.26725546e-01
  2.37827782e-01  2.43582518e-01 -3.75119693e-02 -9.15987235e-02
 -2.36942273e-01  2.23008162e-01  9.47543878e-02 -2.78141983e-01
  1.44470978e-01  6.71825961e-02 -2.25829938e-01 -4.42259325e-01
 -9.64811171e-02 -8.60566823e-02  3.38448633e-02 -1.34619833e-02
  3.78416658e-02  9.48201281e-02  3.77327451e-01  1.91940726e-01
 -1.54066338e-01  1.72852129e-01  5.25229154e-01 -4.92920784e-01
 -3.62215987e-01  1.31570656e-01  2.15440102e-01 -1.15106183e-01
  6.65436536e-01  6.55034152e-02  7.58460152e-02  5.03762122e-01
 -1.87303403e-01 -1.02304450e-01  1.64157523e-01  5.23324031e-02
 -4.17247557e-01  1.50395842e-01  5.07546421e-01  1.43624444e-01
 -5.60996675e-01 -9.58589581e-02 -3.62334275e-01  1.91390476e-01
 -2.85882834e-01  2.73338520e-01 -1.85801180e-01  9.64251369e-02
  2.71069907e-01 -3.18112542e-01  3.10476612e-01 -1.08959888e-01
 -3.77610338e-01  1.63393654e-01  9.30309361e-02  1.88936146e-01
  5.05577504e-01  3.36466752e-01 -3.55833435e-01  1.44847107e-01
 -1.63448864e-02  3.44096339e-01 -6.95357192e-02 -1.16209878e-01
  2.50130338e-01 -2.10051544e-01 -8.30869276e-02 -3.95269977e-01
 -2.53037657e-01 -2.66153212e-01  1.07149140e-01 -2.23996582e-02
  1.04421906e-01  3.30889340e-01  3.33225482e-01 -1.93075100e-01
 -9.28748184e-02  8.47771084e-02  2.33794727e-01  1.03872930e-01
  2.66970199e-01  2.14155846e-01  1.40288467e-01  2.13091961e-01
  3.29361091e-02 -2.38449142e-01 -1.20361090e-01 -2.26201736e-01
  1.45426262e-01 -3.92460557e-01 -1.07245998e-01 -7.23676972e-02
  1.21710385e-01  3.99271291e-01 -4.83653823e-02  3.47299381e-01
 -1.24747421e-01 -1.34015245e-01 -2.09049080e-01 -8.06792714e-02
  3.60883916e-01  3.85525781e-01  4.01688235e-01  1.83923981e-01
 -1.62675690e+00 -5.48632289e-01 -5.00732228e-01  3.87101946e-02
 -9.59443901e-02  2.63396441e-01 -8.27959300e-02 -3.01627496e-01
 -5.69708984e-01 -1.79504861e-01  5.46924706e-02  2.87831951e-01
 -1.31865178e-01 -3.73026820e-01  2.37307852e-01  2.63623504e-01
  2.51425690e-01  1.28955064e-01 -4.39622417e-01  5.98709953e-01
 -4.66598783e-01  1.71386383e-01 -1.54879361e-01 -1.17900158e-01
  1.23747629e-01  1.73257782e-01 -1.50236882e-02 -7.00700514e-02
  2.40360223e-01  4.93256529e-02  2.29341430e-01  1.81227365e-01
```

```
 1.54288907e-01  3.27128163e-01 -3.10973094e-01 -1.40373869e-01
-4.16329354e-02 -5.30723088e-02 -3.13105522e-01  2.65651049e-02
-1.33789955e-01  5.34451330e-01  7.46314398e-02  5.97685768e-02
 4.24122671e-02  1.76879927e-01  9.52905158e-02 -4.41763729e-02
-7.29418664e-01 -8.33021296e-02 -2.38480772e-02 -1.17907774e-01
 1.83122578e-02  2.75773296e-02  5.67602065e-02 -3.46816892e-02
 2.42759841e-01  1.59731730e-01  4.38607459e-02  2.65120838e-01
 4.12089202e-01 -1.67067127e-01 -1.02280462e-01  6.06591902e-01
-1.70615720e-01 -2.78167328e-02 -1.39511379e-01  2.40869302e-01
 1.84620413e-01 -2.58601470e-01  1.52109039e-01  2.95207375e-01
-3.02271903e-01  2.18182243e-04  1.20692872e-01  2.79213121e-01]
```

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 [83]:

```
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:  (53529, 1)
======================================================================
Sample original prices:
======================================================================
[112.78 104.35  84.39  94.73  11.02]
Sample standardized prices:
======================================================================
[[0.01121089]
 [0.01036775]
 [0.00837141]
 [0.00940559]
 [0.00103317]]
```

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 [85]:

```
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:  (53529, 1)
======================================================================
Sample original quantities:
```

```
==========================================================================
[21  8 88 80 30]
Sample standardized quantities:
==========================================================================
[[0.02478315]
 [0.0086741 ]
 [0.10780669]
 [0.09789343]
 [0.03593556]]
```

3. Vectorizing teacher_number_of_previously_posted_projects

In [0]:

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

In [87]:

```
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:  (53529, 1)
==========================================================================
Sample original quantities:
==========================================================================
[35  1  1  2  1]
Sample standardized teacher_number_of_previously_posted_projects:
==========================================================================
[[0.08009153]
 [0.00228833]
 [0.00228833]
 [0.00457666]
 [0.00228833]]
```

In [0]:

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

# Text data
bowEssayModelSub = bowEssayModel[0:numberOfPoints];
bowTitleModelSub = bowTitleModel[0:numberOfPoints];
tfIdfEssayModelSub = tfIdfEssayModel[0:numberOfPoints];
tfIdfTitleModelSub = tfIdfTitleModel[0:numberOfPoints];

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

# Classes
classesTrainingSub = classesTraining;
```

In [21]:

```
decisionTreeResultsDataFrame = pd.DataFrame(columns =  ['Vectorizer', 'Model', 'Max Depth', 'Min
Samples Split' 'AUC']);
decisionTreeResultsDataFrame
```

Out[21]:

	Vectorizer	Model	Max Depth	Min Samples SplitAUC

Preparing cross validate data for analysis

In [90]:

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

# 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 [91]:

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

```
projectGradeVectorizer.transform(testData['project_grade_category']);

# Test data text features transformation
preProcessedEssaysTemp = preProcessingWithAndWithoutStopWords(testData['project_essay'])[1];
preProcessedTitlesTemp = preProcessingWithAndWithoutStopWords(testData['project_title'])[1];
bowEssayTransformedTestData = bowEssayVectorizer.transform(preProcessedEssaysTemp);
bowTitleTransformedTestData = bowTitleVectorizer.transform(preProcessedTitlesTemp);
tfIdfEssayTransformedTestData = tfIdfEssayVectorizer.transform(preProcessedEssaysTemp);
tfIdfTitleTransformedTestData = tfIdfTitleVectorizer.transform(preProcessedTitlesTemp);
avgWord2VecEssayTransformedTestData = getWord2VecVectors(preProcessedEssaysTemp);
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));
```

Classification using data vectorized by various models by decision trees

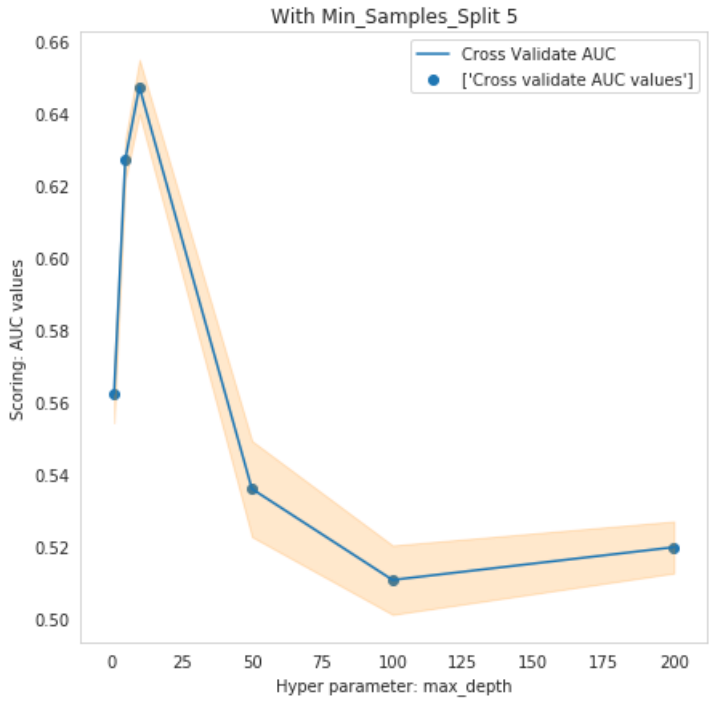
Classification using bag of words vectorized data by decision trees

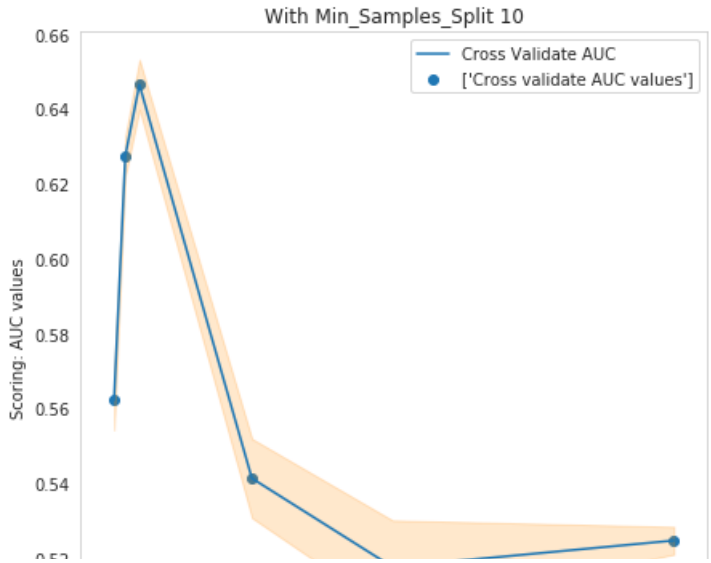
In [0]:

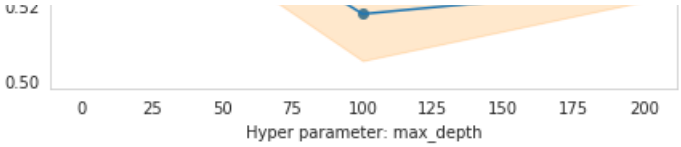
```
techniques = ['Bag of words'];
for index, technique in enumerate(techniques):
    trainingMergedData = hstack((categoriesVectorsSub,\
                                 subCategoriesVectorsSub,\
                                 teacherPrefixVectorsSub,\
                                 schoolStateVectorsSub,\
                                 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));

    dtClassifier = tree.DecisionTreeClassifier(class_weight="balanced");
    tunedParameters = {'max_depth': [1, 5, 10, 50, 100, 200], 'min_samples_split': [5, 10, 100, 500]
};    classifier = GridSearchCV(dtClassifier, tunedParameters, cv = 5, scoring = 'roc_auc');
    classifier.fit(trainingMergedData, classesTrainingSub);
```

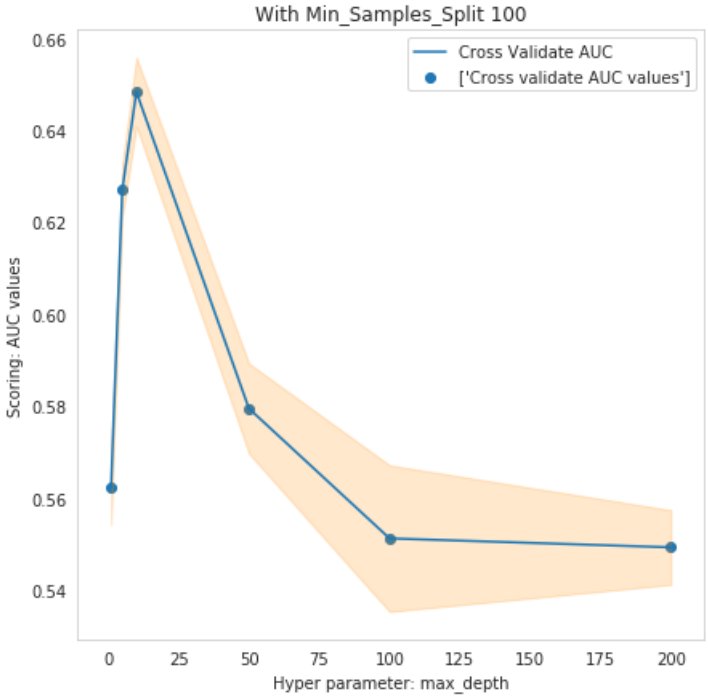
```
    testScoresDataFrame = pd.DataFrame(data = np.hstack((classifier.cv_results_['param_max_depth'].
data[:, None], classifier.cv_results_['param_min_samples_split'].data[:, None],
classifier.cv_results_['mean_test_score'][:, None], classifier.cv_results_['std_test_score'][:,
None])), columns = ['max_depth', 'min_samples_split', 'mts', 'stdts']);
    testScoresDataFrame = testScoresDataFrame.astype(float);

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

    for splitValue in tunedParameters['min_samples_split']:
      plt.plot(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['max_dep
th'], testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['mts'], label = "
Cross Validate AUC");
      plt.scatter(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['max_
depth'], testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['mts'], label
= ['Cross validate AUC values']);
      plt.gca().fill_between(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitV
alue]['max_depth'].values, np.array(testScoresDataFrame[testScoresDataFrame['min_samples_split'] ==
splitValue]['mts'].values - testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitVa
lue]['stdts'].values, dtype = float),\
                              np.array(testScoresDataFrame[testScoresDataFrame['min_samples_split']
= splitValue]['mts'].values + testScoresDataFrame[testScoresDataFrame['min_samples_split'] == split
Value]['stdts'].values, dtype = float), alpha = 0.2, color = 'darkorange');
      plt.xlabel('Hyper parameter: max_depth');
      plt.ylabel('Scoring: AUC values');
      plt.title("With Min_Samples_Split {}".format(splitValue));
      plt.grid();
      plt.legend();
      plt.show();


    optimalHypParamValue = classifier.best_params_['max_depth'];
    optimalHypParam2Value = classifier.best_params_['min_samples_split'];
    dtClassifier = tree.DecisionTreeClassifier(class_weight = 'balanced', max_depth = optimalHypPar
amValue, min_samples_split = optimalHypParam2Value);
    dtClassifier.fit(trainingMergedData, classesTrainingSub);
    predScoresTraining = dtClassifier.predict_proba(trainingMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTraining, predScoresTraining[:, 1]);
    predScoresTest = dtClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predScoresTest[:, 1]);
    predictionClassesTest = dtClassifier.predict(testMergedData);

    falsePositivePointsIndexes = [];
    for index, classValue in enumerate(classesTest):
      if(classValue == 0 and predictionClassesTest[index] == 1):
        falsePositivePointsIndexes.append(index);
    falsePositiveData = testData.iloc[falsePositivePointsIndexes]
    essayWords = '';
    for essay in falsePositiveData['preprocessed_essays'].values:
      essayWords = essayWords + " " + essay;
    equalsBorder(60);
    print("Word Cloud of Essays of False Positive Data: ");
    equalsBorder(50);
    wordCloud = WordCloud(width = 800, height = 800, background_color = 'white', min_font_size = 10
).generate(essay);
    plt.figure(figsize = (8, 8), facecolor = None);
    plt.imshow(wordCloud);
    plt.axis('off');
    plt.tight_layout(pad= 0);
    plt.show();

    equalsBorder(90);
    sbrn.boxplot(falsePositiveData['price'], orient = 'v', palette = 'Set3').set_title('Box plot of
price feature in false positive data');
    plt.show();

    equalsBorder(70);
    sbrn.distplot(falsePositiveData['teacher_number_of_previously_posted_projects'], color = 'g').s
et_title('Pdf of previously posted feature in false positive data');
    plt.show();

    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");
```

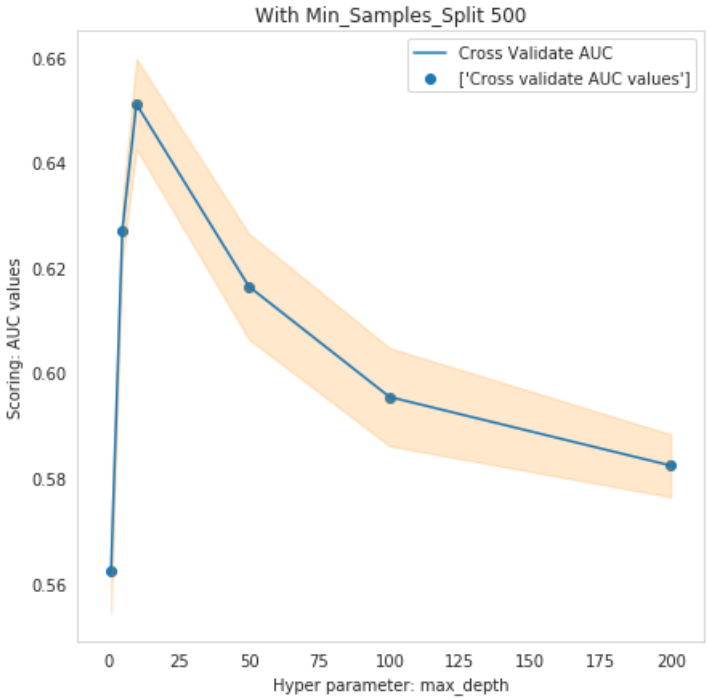
```
    plt.ylabel("tpr values");
    plt.grid();
    plt.legend();
    plt.show();

    areaUnderRocValueTest = auc(fprTest, tprTest);

    print("Results of analysis using {} vectorized text features merged with other features using
decision tree classifier: ".format(technique));
    equalsBorder(70);
    print("Optimal Max_Depth Value: ", optimalHypParamValue);
    equalsBorder(40);
    print("Optimal Min_Samples_Split Value: ", optimalHypParam2Value);
    equalsBorder(40);
    print("AUC value of test data: ", str(areaUnderRocValueTest));
    # Predicting classes of test data projects
    predictionClassesTest = dtClassifier.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();
```



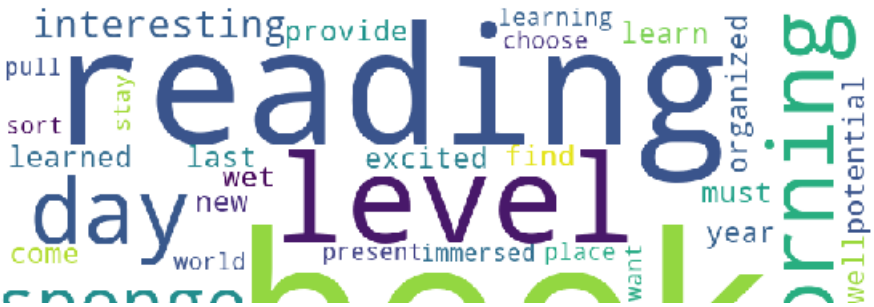


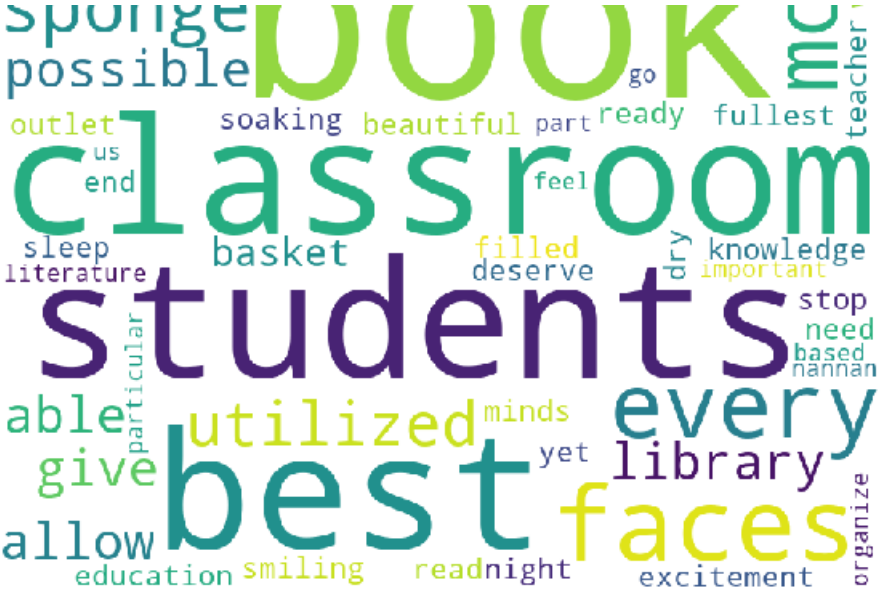




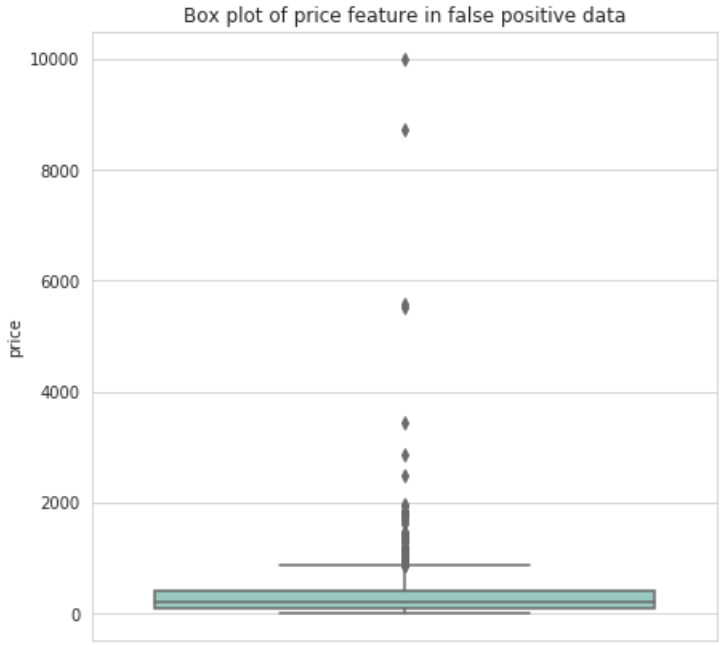


Word Cloud of Essays of False Positive Data:
==========

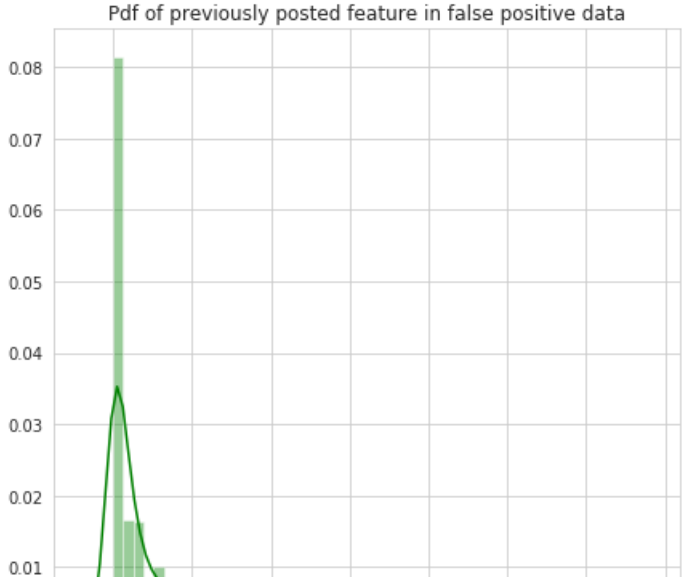


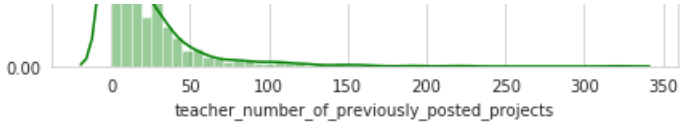

sponge
possible
book
go
outlet
soaking
beautiful
part
ready
fullest
teacher
us
end
classroom
feel
sleep
literature
basket
filled
deserve
dry
knowledge
important
students
stop
need
based
nannan
particular
able
utilized
minds
every
give
yet
library
best
faces
allow
education
smiling
readnight
excitement
organize

==========

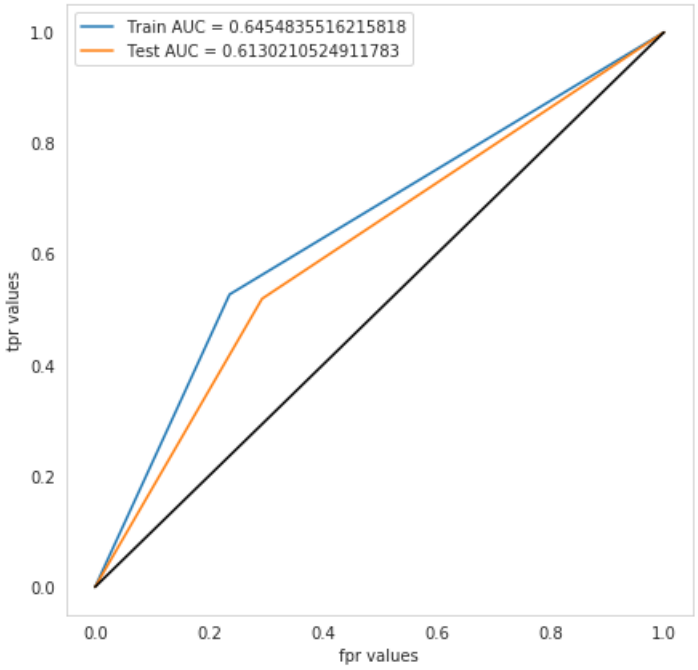

Box plot of price feature in false positive data
price
10000
8000
6000
4000
2000
0

==========

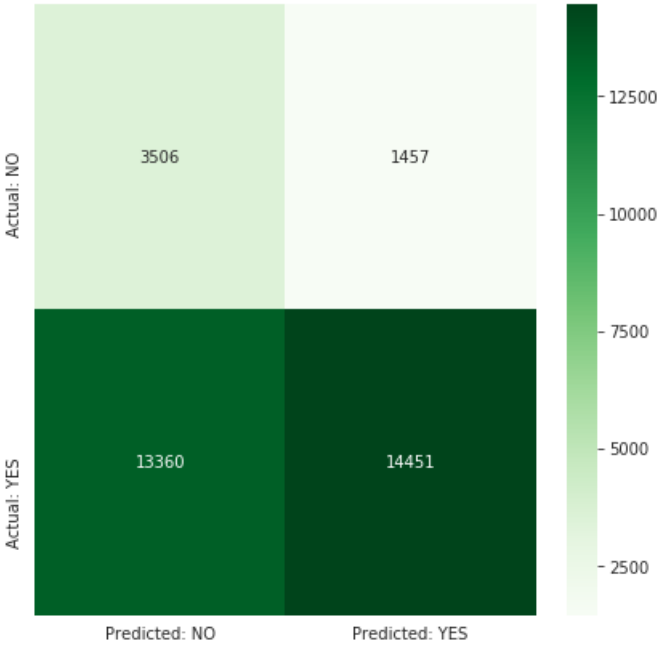

Pdf of previously posted feature in false positive data
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01



==========




```
Results of analysis using Bag of words vectorized text features merged with other features using d
ecision tree classifier:
====================================================================
Optimal Max_Depth Value:  10
========================================
Optimal Min_Samples_Split Value:  500
========================================
AUC value of test data:  0.6130210524911783
========================================
Confusion Matrix :
============================================================
```

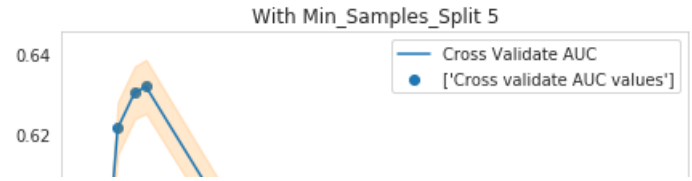


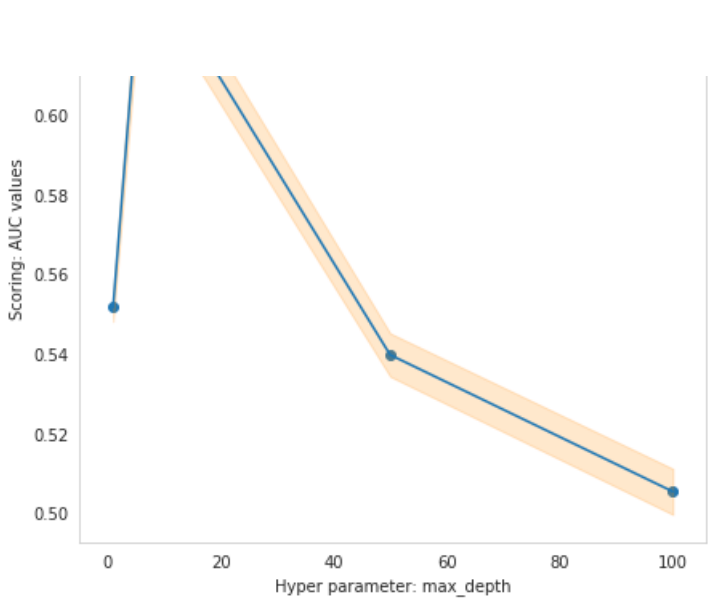
Classification using tf-idf vectorized data by decision trees

In [0]:

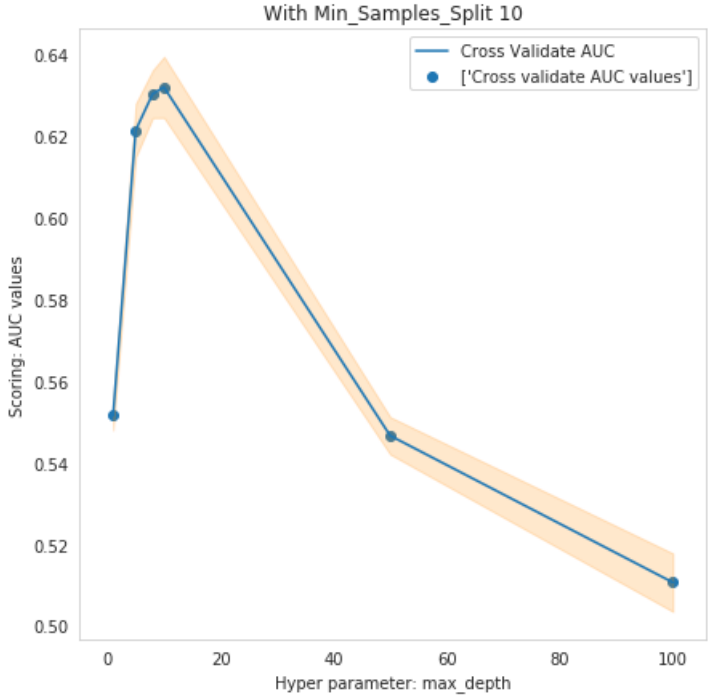
```
techniques = ['Tf-Idf'];
for index, technique in enumerate(techniques):
    trainingMergedData = hstack((categoriesVectorsSub,\
                                 subCategoriesVectorsSub,\
                                 teacherPrefixVectorsSub,\
                                 schoolStateVectorsSub,\
                                 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));

    dtClassifier = tree.DecisionTreeClassifier(class_weight="balanced");
    tunedParameters = {'max_depth': [1, 5, 8, 10, 50, 100], 'min_samples_split': [5, 10, 100, 500]};

    classifier = GridSearchCV(dtClassifier, tunedParameters, cv = 5, scoring = 'roc_auc');
    classifier.fit(trainingMergedData, classesTrainingSub);

    testScoresDataFrame = pd.DataFrame(data = np.hstack((classifier.cv_results_['param_max_depth'].
data[:, None], classifier.cv_results_['param_min_samples_split'].data[:, None],
classifier.cv_results_['mean_test_score'][:, None], classifier.cv_results_['std_test_score'][:,
None])), columns = ['max_depth', 'min_samples_split', 'mts', 'stdts']);
    testScoresDataFrame = testScoresDataFrame.astype(float);

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

    for splitValue in tunedParameters['min_samples_split']:
      plt.plot(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['max_dep
th'], testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['mts'], label = "
Cross Validate AUC");
      plt.scatter(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['max_
depth'], testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['mts'], label
= ['Cross validate AUC values']);
      plt.gca().fill_between(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitV
alue]['max_depth'].values, np.array(testScoresDataFrame[testScoresDataFrame['min_samples_split'] ==
splitValue]['mts'].values - testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitVa
lue]['stdts'].values, dtype = float),\
                        np.array(testScoresDataFrame[testScoresDataFrame['min_samples_split']
= splitValue]['mts'].values + testScoresDataFrame[testScoresDataFrame['min_samples_split'] == split
Value]['stdts'].values, dtype = float), alpha = 0.2, color = 'darkorange');
      plt.xlabel('Hyper parameter: max_depth');
      plt.ylabel('Scoring: AUC values');
      plt.title("With Min_Samples_Split {}".format(splitValue));
      plt.grid();
      plt.legend();
      plt.show();


    optimalHypParamValue = classifier.best_params_['max_depth'];
    optimalHypParam2Value = classifier.best_params_['min_samples_split'];
    dtClassifier = tree.DecisionTreeClassifier(class_weight = 'balanced', max_depth = optimalHypPar
```

```
amValue, min_samples_split = optimalHypParam2Value);
    dtClassifier.fit(trainingMergedData, classesTrainingSub);
    predScoresTraining = dtClassifier.predict_proba(trainingMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTraining, predScoresTraining[:, 1]);
    predScoresTest = dtClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predScoresTest[:, 1]);
    predictionClassesTest = dtClassifier.predict(testMergedData);

    falsePositivePointsIndexes = [];
    for index, classValue in enumerate(classesTest):
      if(classValue == 0 and predictionClassesTest[index] == 1):
        falsePositivePointsIndexes.append(index);
    falsePositiveData = testData.iloc[falsePositivePointsIndexes]
    essayWords = '';
    for essay in falsePositiveData['preprocessed_essays'].values:
      essayWords = essayWords + " " + essay;
    equalsBorder(60);
    print("Word Cloud of Essays of False Positive Data: ");
    equalsBorder(50);
    wordCloud = WordCloud(width = 800, height = 800, background_color = 'white', min_font_size = 10
).generate(essay);
    plt.figure(figsize = (8, 8), facecolor = None);
    plt.imshow(wordCloud);
    plt.axis('off');
    plt.tight_layout(pad= 0);
    plt.show();

    equalsBorder(90);
    sbrn.boxplot(falsePositiveData['price'], orient = 'v', palette = 'Set3').set_title('Box plot of
price feature in false positive data');
    plt.show();

    equalsBorder(70);
    sbrn.distplot(falsePositiveData['teacher_number_of_previously_posted_projects'], color = 'g').s
et_title('Pdf of previously posted feature in false positive data');
    plt.show();

    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
decision tree classifier: ".format(technique));
    equalsBorder(70);
    print("Optimal Max_Depth Value: ", optimalHypParamValue);
    equalsBorder(40);
    print("Optimal Min_Samples_Split Value: ", optimalHypParam2Value);
    equalsBorder(40);
    print("AUC value of test data: ", str(areaUnderRocValueTest));
    # Predicting classes of test data projects
    predictionClassesTest = dtClassifier.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();
```

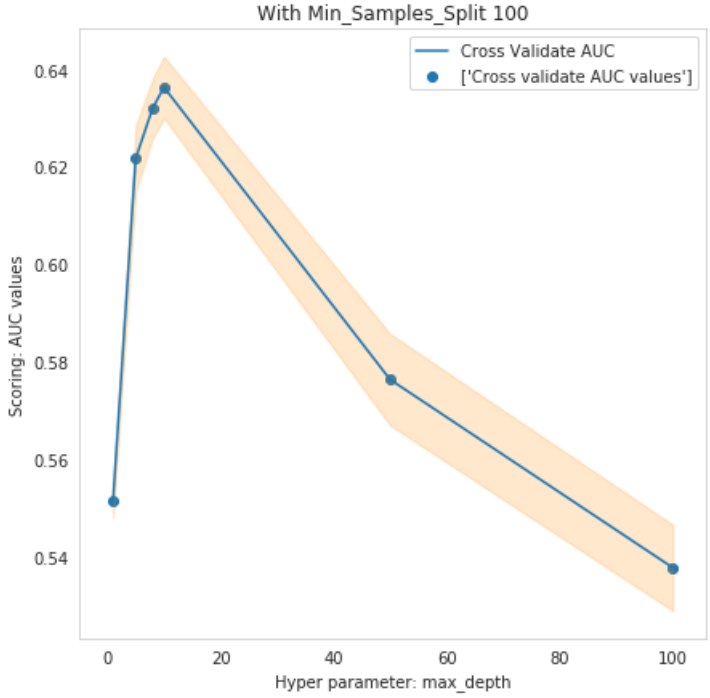




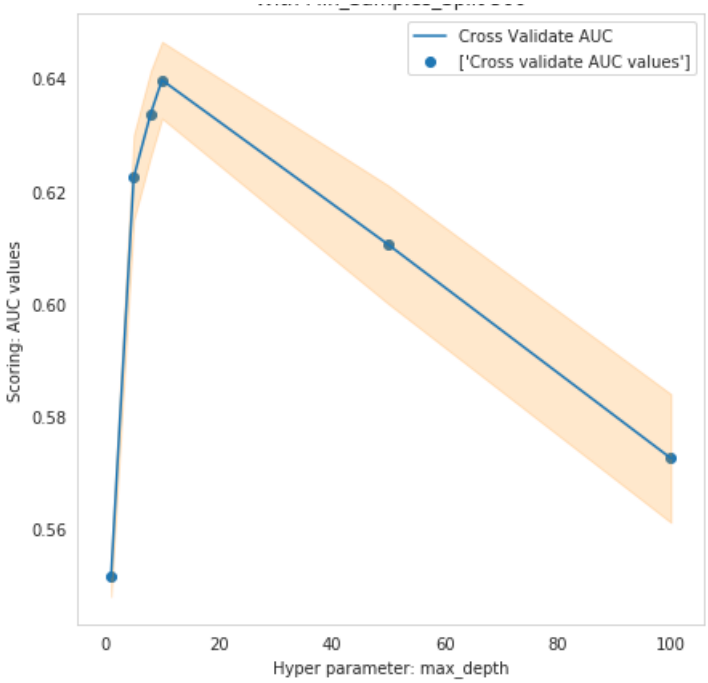
With Min_Samples_Split 10



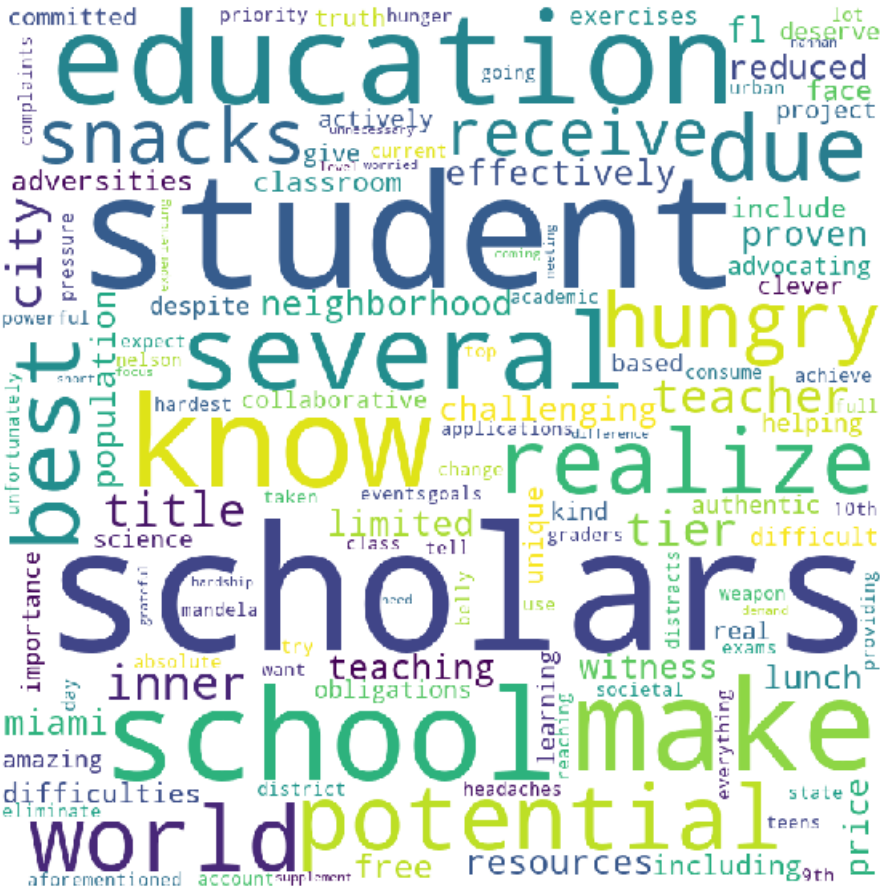
With Min_Samples_Split 100



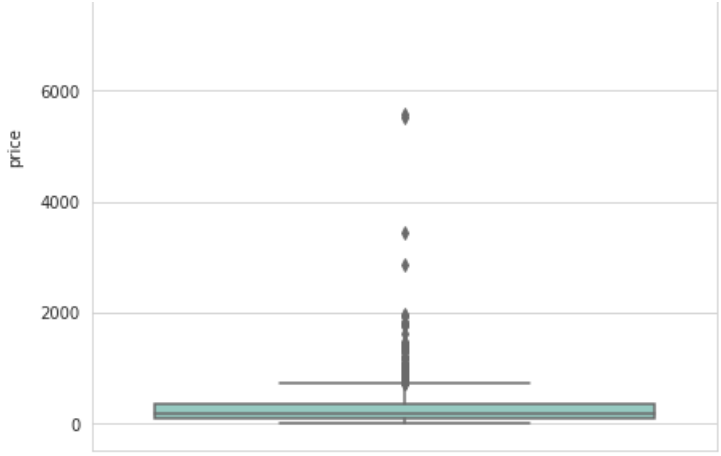
With Min_Samples_Split 500



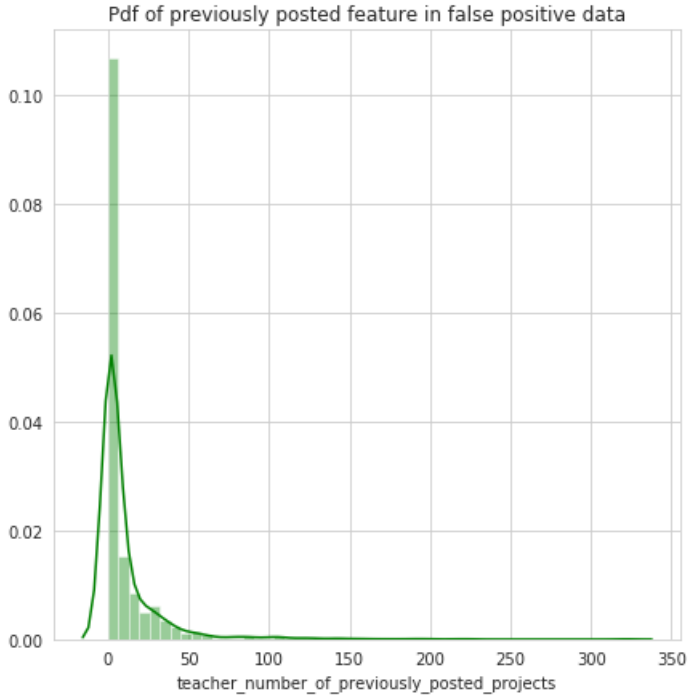
==========
Word Cloud of Essays of False Positive Data:
==========



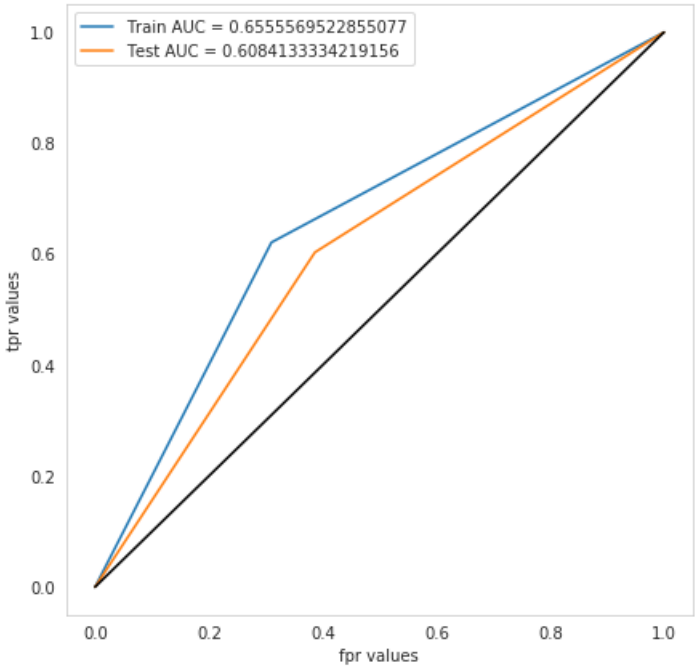
==========


price
6000
4000
2000
0

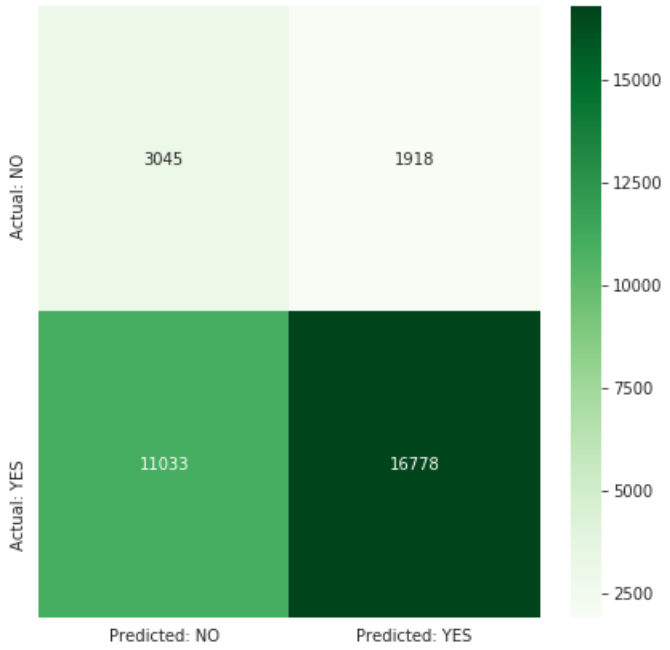
==========


Pdf of previously posted feature in false positive data
0.10
0.08
0.06
0.04
0.02
0.00
0
50
100
150
200
250
300
350
teacher_number_of_previously_posted_projects

==========


Train AUC = 0.6555569522855077
Test AUC = 0.6084133334219156
tpr values
fpr values
0.0
0.2
0.4
0.6
0.8
1.0

```
Results of analysis using Tf-Idf vectorized text features merged with other features using
decision tree classifier:
========================================================================
Optimal Max_Depth Value:  10
======================================
Optimal Min_Samples_Split Value:  500
======================================
AUC value of test data:  0.6084133334219156
======================================
Confusion Matrix :
============================================================
```



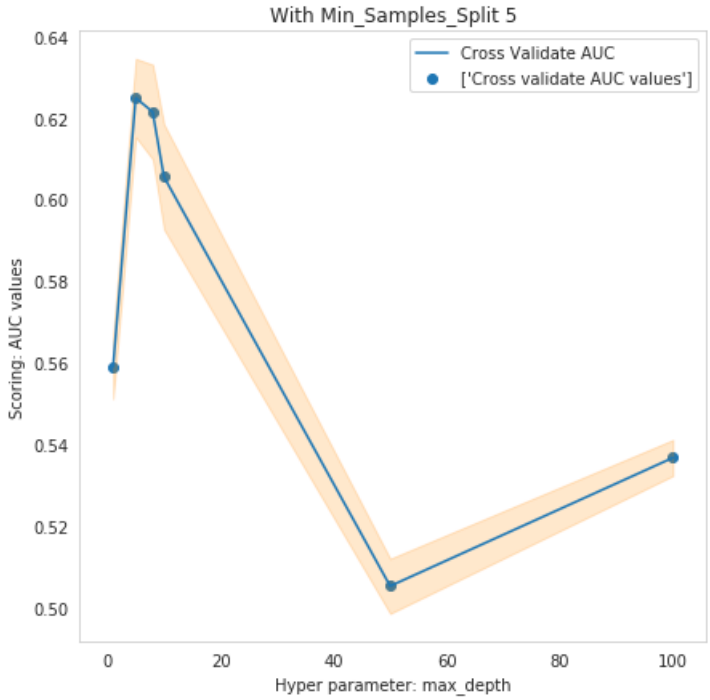
Classification using word2vec vectorized data by decision trees

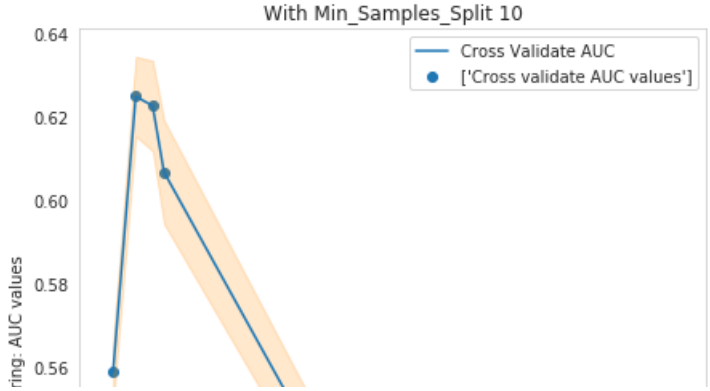
In [0]:

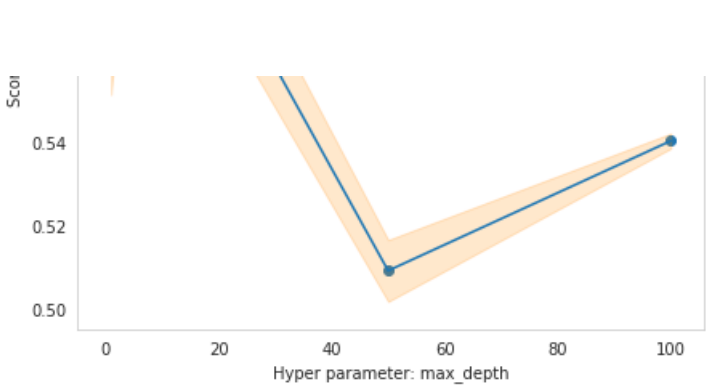
```
techniques = ['Average Word2Vec'];
for index, technique in enumerate(techniques):
    trainingMergedData = hstack((categoriesVectorsSub,\
                                 subCategoriesVectorsSub,\
                                 teacherPrefixVectorsSub,\
                                 schoolStateVectorsSub,\
                                 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,\
```

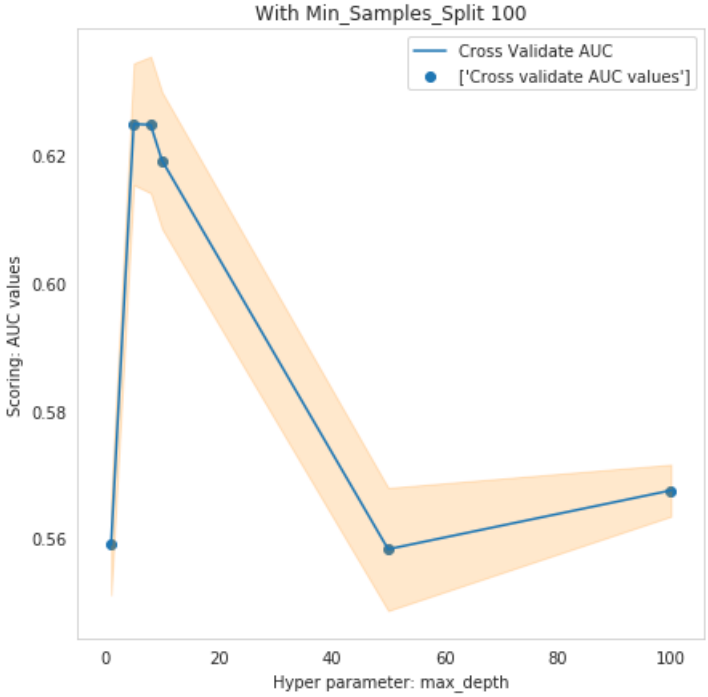
```
                                    avgWord2VecEssayTransformedTestData));

    dtClassifier = tree.DecisionTreeClassifier(class_weight="balanced");
    tunedParameters = {'max_depth': [1, 5, 8, 10, 50, 100], 'min_samples_split': [5, 10, 100, 500]};
classifier = GridSearchCV(dtClassifier, tunedParameters, cv = 5, scoring = 'roc_auc');
    classifier.fit(trainingMergedData, classesTrainingSub);

    testScoresDataFrame = pd.DataFrame(data = np.hstack((classifier.cv_results_['param_max_depth'].
data[:, None], classifier.cv_results_['param_min_samples_split'].data[:, None],
classifier.cv_results_['mean_test_score'][:, None], classifier.cv_results_['std_test_score'][:,
None])), columns = ['max_depth', 'min_samples_split', 'mts', 'stdts']);
    testScoresDataFrame = testScoresDataFrame.astype(float);

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

    for splitValue in tunedParameters['min_samples_split']:
      plt.plot(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['max_dep
th'], testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['mts'], label = "
Cross Validate AUC");
      plt.scatter(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['max_
depth'], testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['mts'], label
= ['Cross validate AUC values']);
      plt.gca().fill_between(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitV
alue]['max_depth'].values, np.array(testScoresDataFrame[testScoresDataFrame['min_samples_split'] ==
splitValue]['mts'].values - testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitVa
lue]['stdts'].values, dtype = float),\
                          np.array(testScoresDataFrame[testScoresDataFrame['min_samples_split']
= splitValue]['mts'].values + testScoresDataFrame[testScoresDataFrame['min_samples_split'] == split
Value]['stdts'].values, dtype = float), alpha = 0.2, color = 'darkorange');
      plt.xlabel('Hyper parameter: max_depth');
      plt.ylabel('Scoring: AUC values');
      plt.title("With Min_Samples_Split {}".format(splitValue));
      plt.grid();
      plt.legend();
      plt.show();


    optimalHypParamValue = classifier.best_params_['max_depth'];
    optimalHypParam2Value = classifier.best_params_['min_samples_split'];
    dtClassifier = tree.DecisionTreeClassifier(class_weight = 'balanced', max_depth = optimalHypPar
amValue, min_samples_split = optimalHypParam2Value);
    dtClassifier.fit(trainingMergedData, classesTrainingSub);
    predScoresTraining = dtClassifier.predict_proba(trainingMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTraining, predScoresTraining[:, 1]);
    predScoresTest = dtClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predScoresTest[:, 1]);
    predictionClassesTest = dtClassifier.predict(testMergedData);

    falsePositivePointsIndexes = [];
    for index, classValue in enumerate(classesTest):
      if(classValue == 0 and predictionClassesTest[index] == 1):
        falsePositivePointsIndexes.append(index);
    falsePositiveData = testData.iloc[falsePositivePointsIndexes]
    essayWords = '';
    for essay in falsePositiveData['preprocessed_essays'].values:
      essayWords = essayWords + " " + essay;
    equalsBorder(60);
    print("Word Cloud of Essays of False Positive Data: ");
    equalsBorder(50);
    wordCloud = WordCloud(width = 800, height = 800, background_color = 'white', min_font_size = 10
).generate(essay);
    plt.figure(figsize = (8, 8), facecolor = None);
    plt.imshow(wordCloud);
    plt.axis('off');
    plt.tight_layout(pad= 0);
    plt.show();

    equalsBorder(90);
    sbrn.boxplot(falsePositiveData['price'], orient = 'v', palette = 'Set3').set_title('Box plot of
price feature in false positive data');
    plt.show();

    equalsBorder(70);
    sbrn.distplot(falsePositiveData['teacher_number_of_previously_posted_projects'], color = 'g').s
et_title('Pdf of previously posted feature in false positive data');
    plt.show();
```

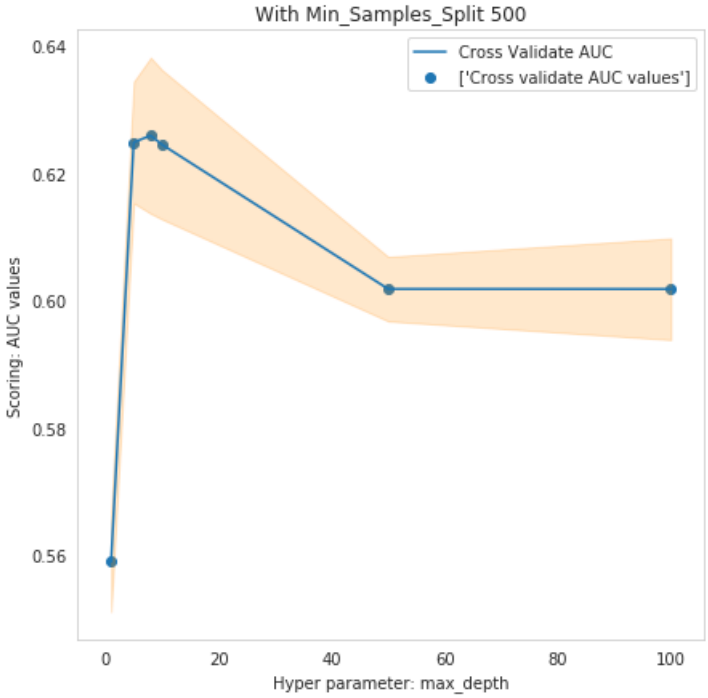
```
    plt.show();

    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
decision tree classifier: ".format(technique));
    equalsBorder(70);
    print("Optimal Max_Depth Value: ", optimalHypParamValue);
    equalsBorder(40);
    print("Optimal Min_Samples_Split Value: ", optimalHypParam2Value);
    equalsBorder(40);
    print("AUC value of test data: ", str(areaUnderRocValueTest));
    # Predicting classes of test data projects
    predictionClassesTest = dtClassifier.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();
```



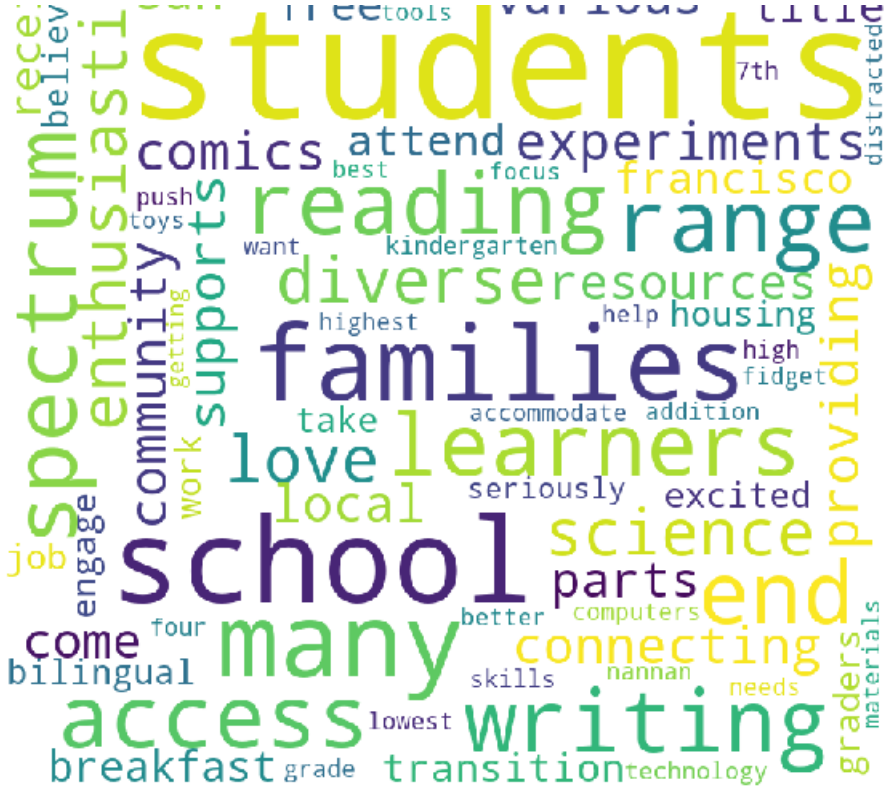




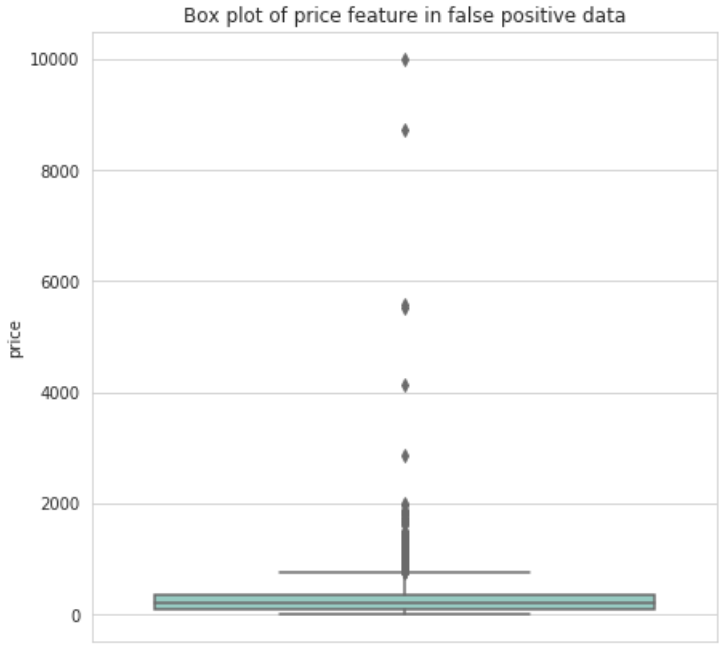




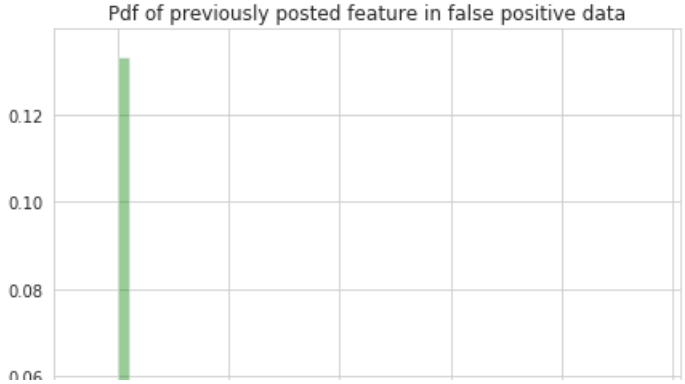
==========
Word Cloud of Essays of False Positive Data:
==========

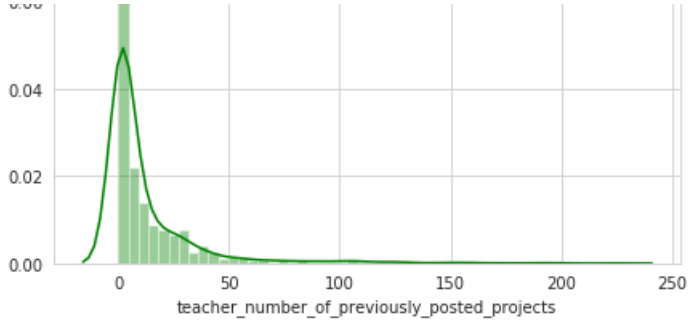

students
reading
range
spectrum
enthusiasti
comics
attend
experiments
francisco
diverse
resources
families
providing
housing
learners
love
local
science
school
parts
end
many
connecting
access
writing
breakfast
transition
technology
tools
7th
distracted
best
focus
push
toys
want
kindergarten
highest
help
high
fidget
take
accommodate
addition
seriously
excited
job
engage
work
getting
four
come
bilingual
better
computers
skills
nannan
needs
graders
materials
lowest
grade

==========

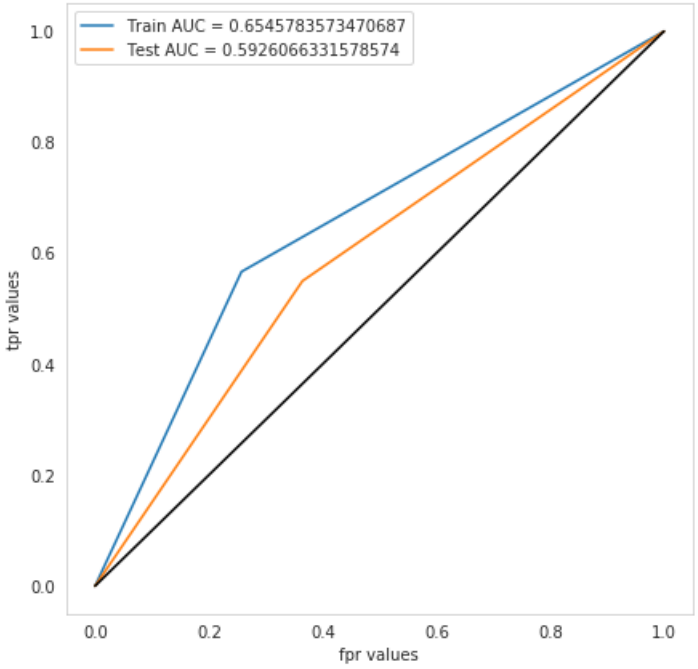

Box plot of price feature in false positive data
price
10000
8000
6000
4000
2000
0

==========


Pdf of previously posted feature in false positive data
0.12
0.10
0.08
0.06



==========



Results of analysis using Average Word2Vec vectorized text features merged with other features usi
ng decision tree classifier:

==========

Optimal Max_Depth Value: 8

==========

Optimal Min_Samples_Split Value: 500

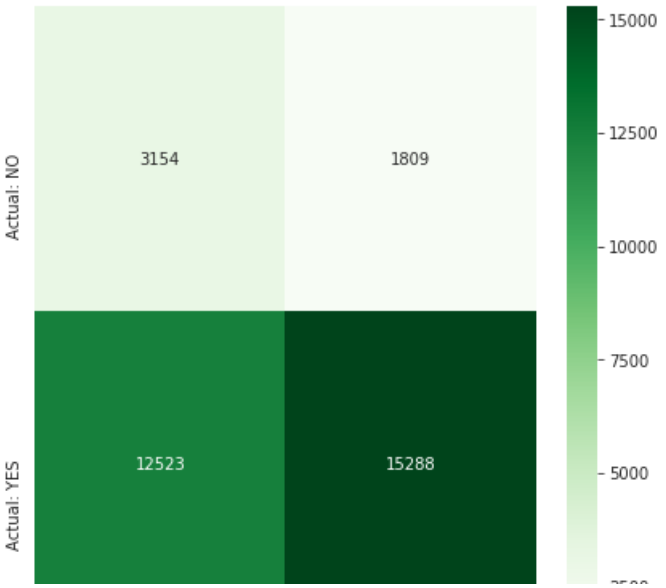
==========

AUC value of test data: 0.5926066331578574

==========

Confusion Matrix :

==========

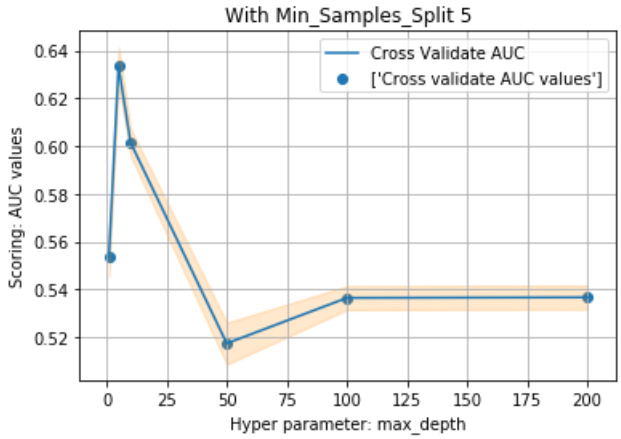


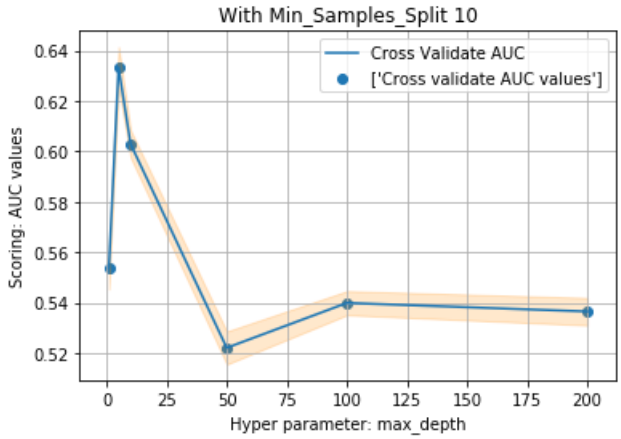
Classification using tf-idf weighted word2vec vectorized data by decision trees

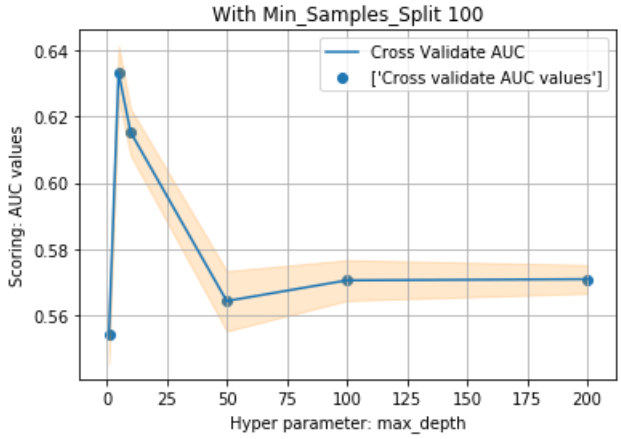
In [0]:

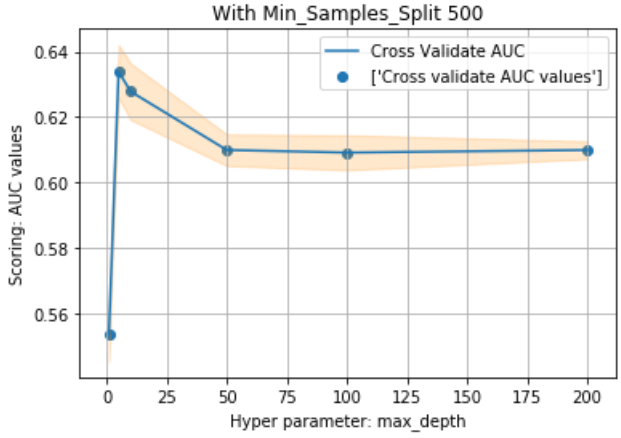
```
techniques = ['Tf-Idf Weighted Word2Vec'];
for index, technique in enumerate(techniques):
    trainingMergedData = hstack((categoriesVectorsSub,\
                                 subCategoriesVectorsSub,\
                                 teacherPrefixVectorsSub,\
                                 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));

    dtClassifier = tree.DecisionTreeClassifier(class_weight="balanced");
    tunedParameters = {'max_depth': [1, 5, 10, 50, 100, 200], 'min_samples_split': [5, 10, 100, 500]
};    classifier = GridSearchCV(dtClassifier, tunedParameters, cv = 5, scoring = 'roc_auc');
    classifier.fit(trainingMergedData, classesTrainingSub);

    testScoresDataFrame = pd.DataFrame(data = np.hstack((classifier.cv_results_['param_max_depth'].
data[:, None], classifier.cv_results_['param_min_samples_split'].data[:, None],
classifier.cv_results_['mean_test_score'][:, None], classifier.cv_results_['std_test_score'][:,
None])), columns = ['max_depth', 'min_samples_split', 'mts', 'stdts']);
    testScoresDataFrame = testScoresDataFrame.astype(float);

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

    for splitValue in tunedParameters['min_samples_split']:
      plt.plot(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['max_dep
th'], testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['mts'], label = "
Cross Validate AUC");
      plt.scatter(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['max_
depth'], testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['mts'], label
= ['Cross validate AUC values']);
      plt.gca().fill_between(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitV
alue]['max_depth'].values, np.array(testScoresDataFrame[testScoresDataFrame['min_samples_split'] ==
splitValue]['mts'].values - testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitVa
lue]['stdts'].values, dtype = float),\
                        np.array(testScoresDataFrame[testScoresDataFrame['min_samples_split']
= splitValue]['mts'].values + testScoresDataFrame[testScoresDataFrame['min_samples_split'] == split
Value]['stdts'].values, dtype = float), alpha = 0.2, color = 'darkorange');
      plt.xlabel('Hyper parameter: max_depth');
      plt.ylabel('Scoring: AUC values');
      plt.title("With Min_Samples_Split {}".format(splitValue));
      plt.grid();
      plt.legend();
```

```
      plt.show();


    optimalHypParamValue = classifier.best_params_['max_depth'];
    optimalHypParam2Value = classifier.best_params_['min_samples_split'];
    dtClassifier = tree.DecisionTreeClassifier(class_weight = 'balanced', max_depth = optimalHypPar
amValue, min_samples_split = optimalHypParam2Value);
    dtClassifier.fit(trainingMergedData, classesTrainingSub);
    predScoresTraining = dtClassifier.predict_proba(trainingMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTraining, predScoresTraining[:, 1]);
    predScoresTest = dtClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predScoresTest[:, 1]);
    predictionClassesTest = dtClassifier.predict(testMergedData);

    falsePositivePointsIndexes = [];
    for index, classValue in enumerate(classesTest):
      if(classValue == 0 and predictionClassesTest[index] == 1):
        falsePositivePointsIndexes.append(index);
    falsePositiveData = testData.iloc[falsePositivePointsIndexes]
    essayWords = '';
    for essay in falsePositiveData['preprocessed_essays'].values:
      essayWords = essayWords + " " + essay;
    equalsBorder(60);
    print("Word Cloud of Essays of False Positive Data: ");
    equalsBorder(50);
    wordCloud = WordCloud(width = 800, height = 800, background_color = 'white', min_font_size = 10
).generate(essay);
    plt.figure(figsize = (8, 8), facecolor = None);
    plt.imshow(wordCloud);
    plt.axis('off');
    plt.tight_layout(pad= 0);
    plt.show();

    equalsBorder(90);
    sbrn.boxplot(falsePositiveData['price'], orient = 'v', palette = 'Set3').set_title('Box plot of
price feature in false positive data');
    plt.show();

    equalsBorder(70);
    sbrn.distplot(falsePositiveData['teacher_number_of_previously_posted_projects'], color = 'g').s
et_title('Pdf of previously posted feature in false positive data');
    plt.show();

    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
decision tree classifier: ".format(technique));
    equalsBorder(70);
    print("Optimal Max_Depth Value: ", optimalHypParamValue);
    equalsBorder(40);
    print("Optimal Min_Samples_Split Value: ", optimalHypParam2Value);
    equalsBorder(40);
    print("AUC value of test data: ", str(areaUnderRocValueTest));
    # Predicting classes of test data projects
    predictionClassesTest = dtClassifier.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();
```

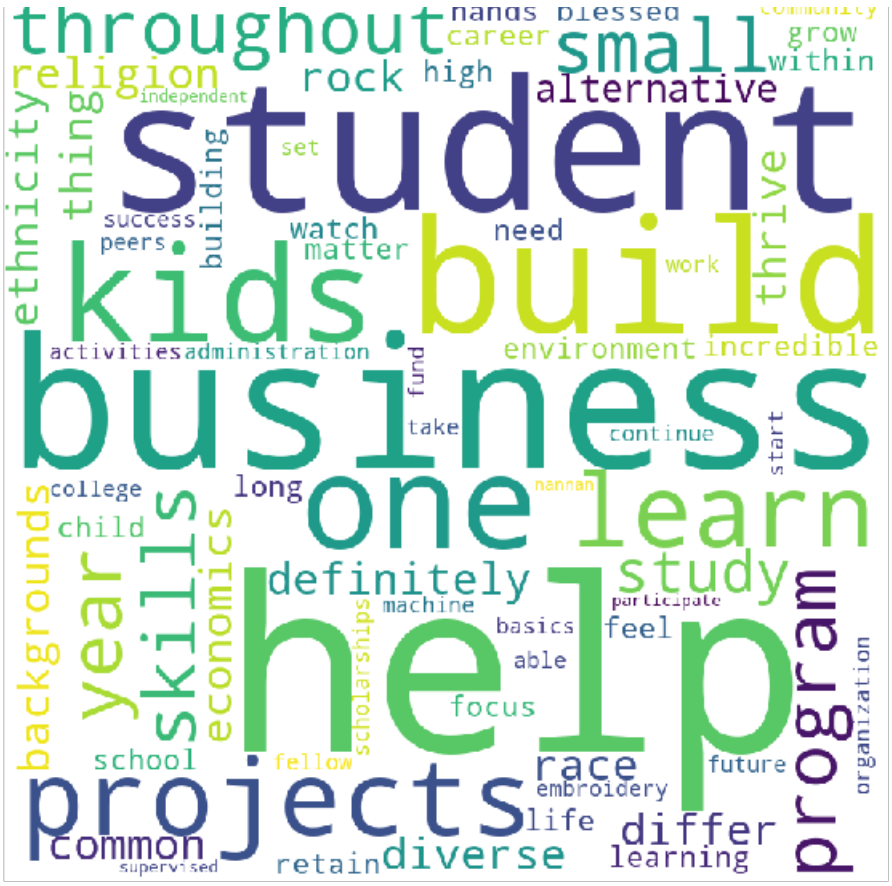




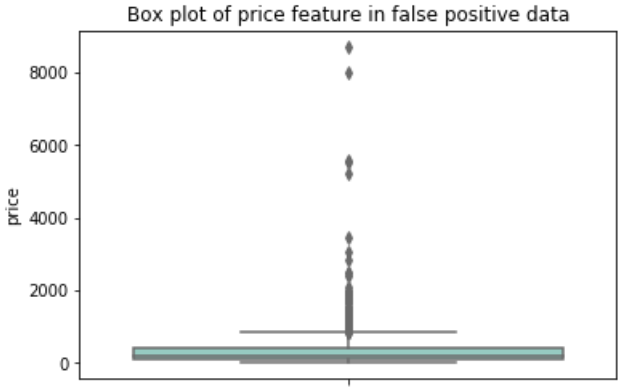




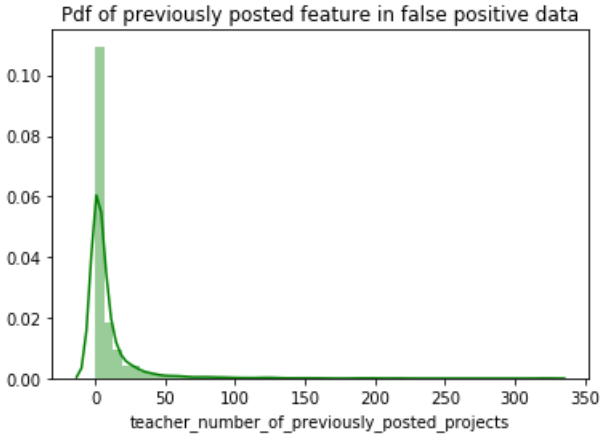
==========
Word Cloud of Essays of False Positive Data:
==========


throughout
hands
blessed
career
small
grow
within
religion
independent
rock
high
alternative
student
ethnicity
thing
set
building
success
peers
watch
matter
need
thrive
kids
build
work
activities
administration
environment
incredible
business
fund
take
continue
start
college
long
one
learn
nannan
backgrounds
child
economics
skills
year
definitely
study
machine
participate
basics
feel
able
help
program
organization
scholarships
focus
school
fellow
race
future
embroidery
projects
life
differ
common
supervised
retain
diverse
learning

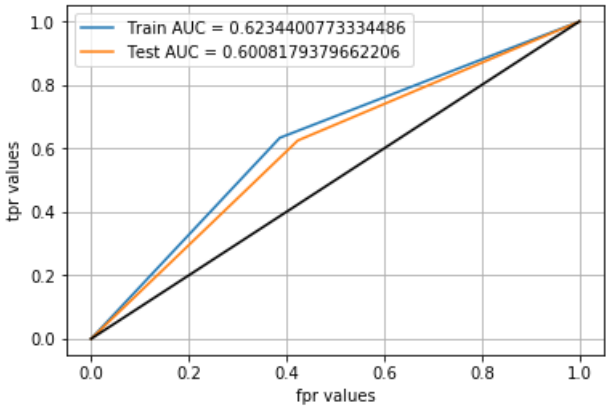
==========


Box plot of price feature in false positive data
price
8000
6000
4000
2000
0

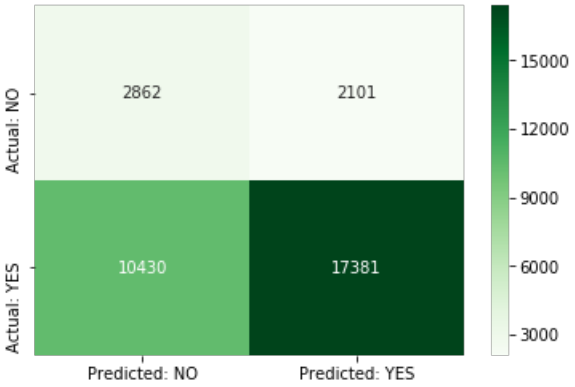
==========


Pdf of previously posted feature in false positive data
0.10
0.08
0.06
0.04
0.02
0.00
0
50
100
150
200
250
300
350
teacher_number_of_previously_posted_projects

==========




```
Results of analysis using Tf-Idf Weighted Word2Vec vectorized text features merged with other feat
ures using decision tree classifier:
========================================================================
Optimal Max_Depth Value:  5
========================================
Optimal Min_Samples_Split Value:  500
========================================
AUC value of test data:  0.6008179379662206
========================================
Confusion Matrix :
============================================================
```



Classification using tf-idf vectorized data containing top 5k features by decision trees

In [93]:

```
featureImportances = np.array(dtClassifier.feature_importances_)
importantFeaturesIndices = np.argsort(featureImportances)[::-1];
importantFeaturesIndices
```

Out[93]:

```
array([  99, 5038,  100, ..., 4783, 4784,    0])
```

In [94]:

```
print(featureImportances[99]);
print(featureImportances[5038]);
```

```
0.16196861971238172
0.10121099256606561
```

In [0]:

```
importantFeaturesIndicesTop5k = importantFeaturesIndices[0:5000];
```

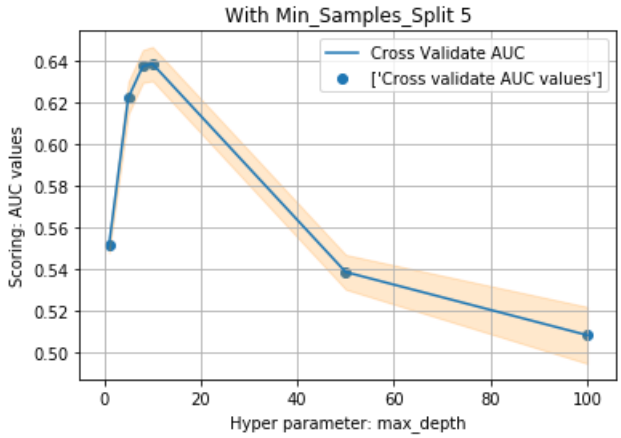
In [0]:

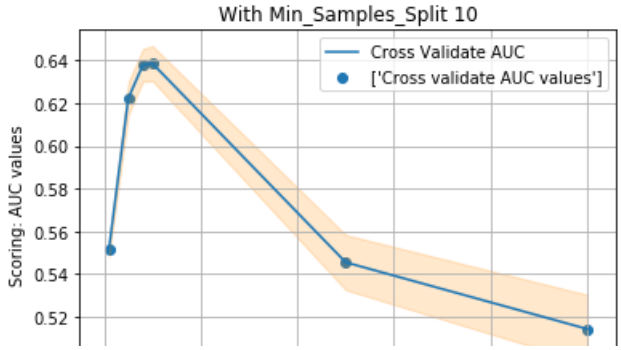
```
trainingMergedData = trainingMergedData.tocsr();
trainingMergedData = trainingMergedData[:, importantFeaturesIndicesTop5k]
crossValidateMergedData = crossValidateMergedData.tocsr();
crossValidateMergedData = crossValidateMergedData[:, importantFeaturesIndicesTop5k]
testMergedData = testMergedData.tocsr();
testMergedData = testMergedData[:, importantFeaturesIndicesTop5k]
```

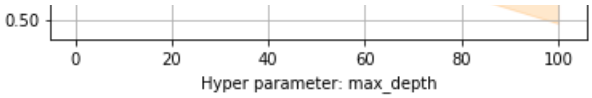
In [106]:

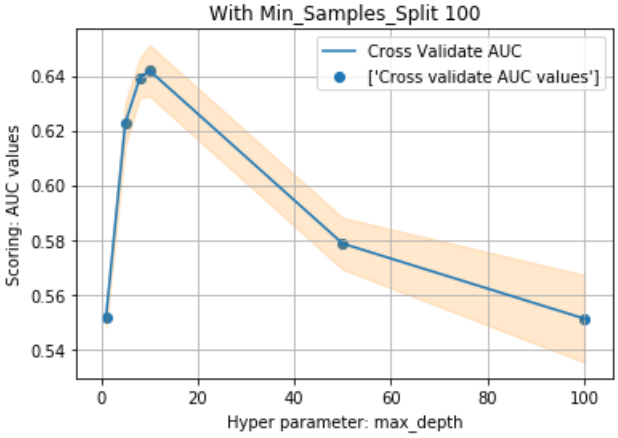
```
techniques = ['Tf-Idf(With reduced features)'];
for index, technique in enumerate(techniques):
    dtClassifier = tree.DecisionTreeClassifier(class_weight='balanced');
    tunedParameters = {'max_depth': [1, 5, 8, 10, 50, 100], 'min_samples_split': [5, 10, 100, 500]};

    classifier = GridSearchCV(dtClassifier, tunedParameters, cv = 5, scoring = 'roc_auc');
    classifier.fit(trainingMergedData, classesTrainingSub);

    testScoresDataFrame = pd.DataFrame(data = np.hstack((classifier.cv_results_['param_max_depth'].
data[:, None], classifier.cv_results_['param_min_samples_split'].data[:, None],
classifier.cv_results_['mean_test_score'][:, None], classifier.cv_results_['std_test_score'][:,
None])), columns = ['max_depth', 'min_samples_split', 'mts', 'stdts']);
    testScoresDataFrame = testScoresDataFrame.astype(float);

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

    for splitValue in tunedParameters['min_samples_split']:
      plt.plot(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['max_dep
th'], testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['mts'], label = "
Cross Validate AUC");
      plt.scatter(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['max_
depth'], testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitValue]['mts'], label
= ['Cross validate AUC values']);
      plt.gca().fill_between(testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitV
alue]['max_depth'].values, np.array(testScoresDataFrame[testScoresDataFrame['min_samples_split'] ==
splitValue]['mts'].values - testScoresDataFrame[testScoresDataFrame['min_samples_split'] == splitVa
lue]['stdts'].values, dtype = float),\
                          np.array(testScoresDataFrame[testScoresDataFrame['min_samples_split']
= splitValue]['mts'].values + testScoresDataFrame[testScoresDataFrame['min_samples_split'] == split
Value]['stdts'].values, dtype = float), alpha = 0.2, color = 'darkorange');
      plt.xlabel('Hyper parameter: max_depth');
      plt.ylabel('Scoring: AUC values');
      plt.title("With Min_Samples_Split {}".format(splitValue));
      plt.grid();
      plt.legend();
      plt.show();


    optimalHypParamValue = classifier.best_params_['max_depth'];
    optimalHypParam2Value = classifier.best_params_['min_samples_split'];
    dtClassifier = tree.DecisionTreeClassifier(class_weight = 'balanced', max_depth = optimalHypPar
amValue, min_samples_split = optimalHypParam2Value);
    dtClassifier.fit(trainingMergedData, classesTrainingSub);
    predScoresTraining = dtClassifier.predict_proba(trainingMergedData);
    fprTrain, tprTrain, thresholdTrain = roc_curve(classesTraining, predScoresTraining[:, 1]);
    predScoresTest = dtClassifier.predict_proba(testMergedData);
    fprTest, tprTest, thresholdTest = roc_curve(classesTest, predScoresTest[:, 1]);
    predictionClassesTest = dtClassifier.predict(testMergedData);

    falsePositivePointsIndexes = [];
    for index, classValue in enumerate(classesTest):
      if(classValue == 0 and predictionClassesTest[index] == 1):
        falsePositivePointsIndexes.append(index);
    falsePositiveData = testData.iloc[falsePositivePointsIndexes]
    essayWords = '';
    for essay in falsePositiveData['preprocessed_essays'].values:
      essayWords = essayWords + " " + essay;
    equalsBorder(60);
    print("Word Cloud of Essays of False Positive Data: ");
    equalsBorder(50);
    wordCloud = WordCloud(width = 800, height = 800, background_color = 'white', min_font_size = 10
).generate(essay);
    plt.figure(figsize = (8, 8), facecolor = None);
    plt.imshow(wordCloud);
    plt.axis('off');
    plt.tight_layout(pad= 0);
```

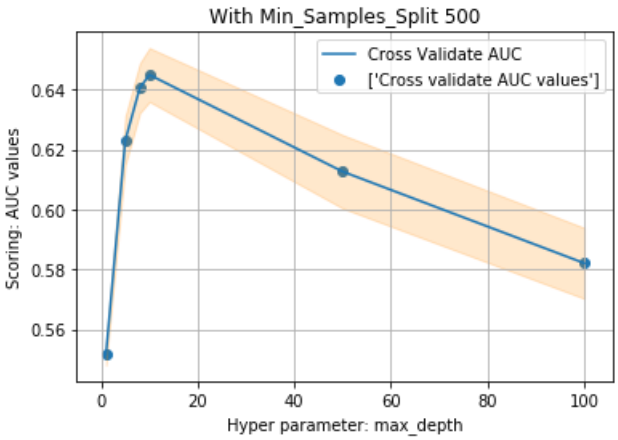
```
    plt.tight_layout(pad= 0);
    plt.show();

    equalsBorder(90);
    sbrn.boxplot(falsePositiveData['price'], orient = 'v', palette = 'Set3').set_title('Box plot of
price feature in false positive data');
    plt.show();

    equalsBorder(70);
    sbrn.distplot(falsePositiveData['teacher_number_of_previously_posted_projects'], color = 'g').s
et_title('Pdf of previously posted feature in false positive data');
    plt.show();

    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
decision tree classifier: ".format(technique));
    equalsBorder(70);
    print("Optimal Max_Depth Value: ", optimalHypParamValue);
    equalsBorder(40);
    print("Optimal Min_Samples_Split Value: ", optimalHypParam2Value);
    equalsBorder(40);
    print("AUC value of test data: ", str(areaUnderRocValueTest));
    # Predicting classes of test data projects
    predictionClassesTest = dtClassifier.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();
```



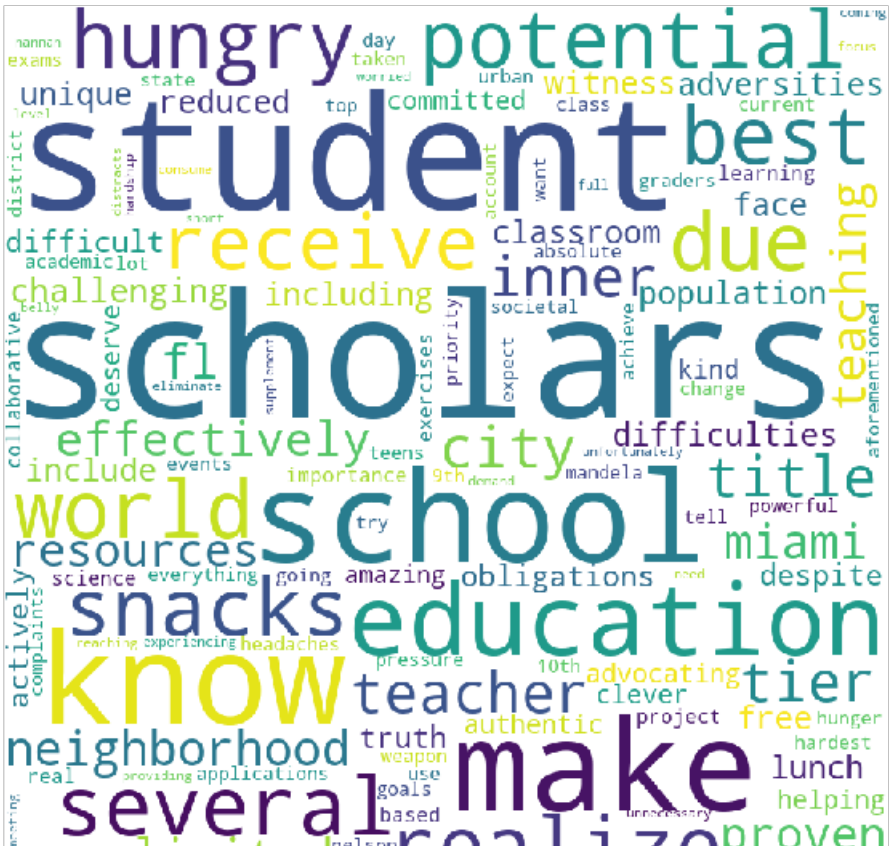




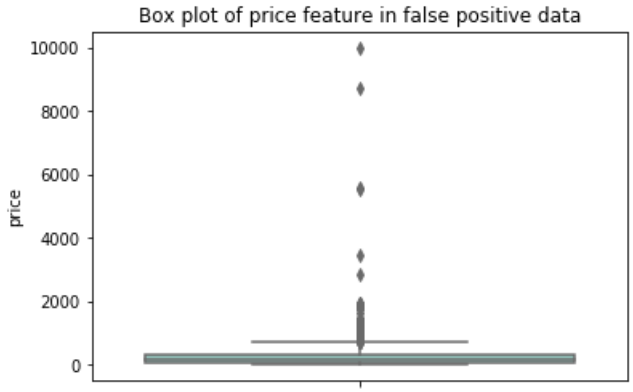




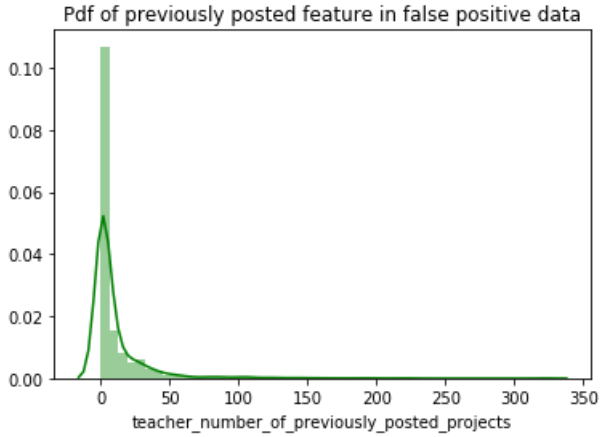
==========
Word Cloud of Essays of False Positive Data:
==========



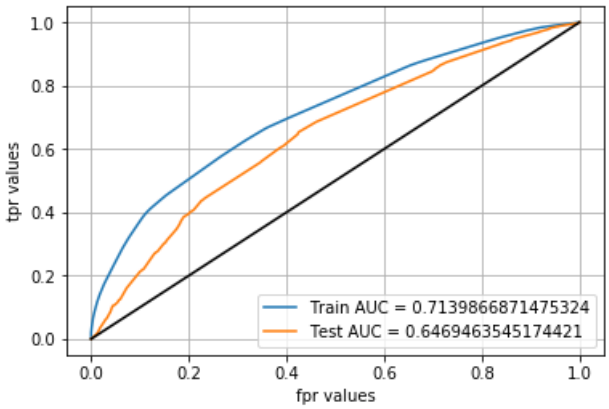
==========



==========



==========



Results of analysis using Tf-Idf(With reduced features) vectorized text features merged with other features using decision tree classifier:

==========

Optimal Max_Depth Value: 10

==========

Optimal Min_Samples_Split Value: 500

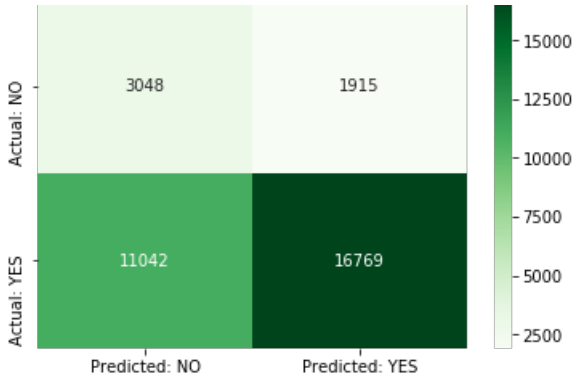
==========

AUC value of test data: 0.6469463545174421

==========

Confusion Matrix :

==========



Summary of results of above classification using decision trees

In [24]:

```
techniques = ['Bag of words', 'Tf-Idf', 'Average Word2Vec', 'Tf-Idf Weighted Word2Vec', 'Tf-
Idf(data with less features)'];
aucValues = [0.6130, 0.6484, 0.5926, 0.6008, 0.6469]
maxDepthValues = [10, 10, 8, 5, 10]
minSamplesSplitValues = [500, 500, 500, 500, 500]
for i,technique in enumerate(techniques):
    decisionTreeResultsDataFrame = decisionTreeResultsDataFrame.append({'Vectorizer': technique,
'Model': 'DecisionTrees', 'Max_Depth': maxDepthValues[i], 'Min_Samples_Split':
minSamplesSplitValues[i], 'AUC': aucValues[i]}, ignore_index = True);
decisionTreeResultsDataFrame
```

Out[24]:

	Vectorizer	Model	AUC	Max_Depth	Min_Samples_Split
0	Bag of words	DecisionTrees	0.6130	10.0	500.0
1	Tf-Idf	DecisionTrees	0.6484	10.0	500.0
2	Average Word2Vec	DecisionTrees	0.5926	8.0	500.0
3	Tf-Idf Weighted Word2Vec	DecisionTrees	0.6008	5.0	500.0
4	Tf-Idf(data with less features)	DecisionTrees	0.6469	10.0	500.0

Conclusions of above analysis

1. It seems like by seeing above results table the model builded using data containing tf-idf vectorized text would be best as it is giving best auc value among others.
2. The model trained with data containing less features is also giving good auc value with just little difference and so it is best to use this combination as it takes less time for training.
3. The best depth and samples split value would be 10 and 500.
4. Most of the models trained with data using various vectorization techniques have same fault like they are all predicting most of the 'yes' values as 'no'.
5. The essays of data points with actual classes as 'no' but predicted as 'yes' contains words like students, families, resources etc... So using these words in essay would increase probability of predicting the proposal as yes.