DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Desc
`project_id`	A unique identifier for the proposed project. **Example:** `p0`
`project_title`	Title of the project. **Exa** • `Art Will Make You H` • `First Grad`
`project_grade_category`	Grade level of students for which the project is targeted. One of the fo enumerated v • `Grades P` • `Grade` • `Grade` • `Grades`
`project_subject_categories`	One or more (comma-separated) subject categories for the project fr following enumerated list of v • `Applied Lea` • `Care & H` • `Health & S` • `History & C` • `Literacy & Lan` • `Math & Sc` • `Music & The` • `Special` • `W` **Exar** • `Music & The` • `Literacy & Language, Math & Sc`

Feature	Desc
school_state	State where school is located (Two-letter U.S. posta (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_c **Exampl**
project_subject_subcategories	One or more (comma-separated) subject subcategories for the p **Exan** • Lit • Literature & Writing, Social Sci
project_resource_summary	An explanation of the resources needed for the project. **Exa** • My students need hands on literacy materials to ma sensory needs!<
project_essay_1	First application
project_essay_2	Second application
project_essay_3	Third application
project_essay_4	Fourth application
project_submitted_datetime	Datetime when project application was submitted. **Example:** 2016-0 12:43:5
teacher_id	A unique identifier for the teacher of the proposed project. **Exa** bdf8baa8fedef6bfeec7ae4ff1c
teacher_prefix	Teacher's title. One of the following enumerated v • • • • • • Tea
teacher_number_of_previously_posted_projects	Number of project applications previously submitted by the same te **Exam**

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

1.1 Reading Data

In [2]:

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

In [3]:

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

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

In [4]:

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

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

Out[4]:

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

1.2 preprocessing of `project_subject_categories`

In [5]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [6]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

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

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

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

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

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

1.3 Text preprocessing

In [7]:

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

In [8]:

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

Out[8]:

Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL

In [9]:

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

In [10]:

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

```
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 language to our school. \r\n\r\n We hav
e 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 a
nd experiences to us that open our eyes to new cultures, beliefs, and respec
t.\"The limits of your language are the limits of your world.\"-Ludwig Wittg
enstein  Our English learner's have a strong support system at home that beg
s for more resources.  Many times our parents are learning to read and speak
English along side of their children.  Sometimes this creates barriers for p
arents to be able to help their child learn phonetics, letter recognition, a
nd other reading skills.\r\n\r\nBy providing these dvd's and players, studen
ts 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 pr
oficiency status, will be a offered to be a part of this program.  These edu
cational 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 deve
lop early reading skills.\r\n\r\nParents that do not have access to a dvd pl
ayer will have the opportunity to check out a dvd player to use for the yea
r.  The plan is to use these videos and educational dvd's for the years to c
ome for other EL students.\r\nnannan
==================================================
The 51 fifth grade students that will cycle through my classroom this year a
ll love learning, at least most of the time. At our school, 97.3% of the stu
dents receive free or reduced price lunch. Of the 560 students, 97.3% are mi
nority students. \r\nThe school has a vibrant community that loves to get to
gether and celebrate. Around Halloween there is a whole school parade to sho
w off the beautiful costumes that students wear. On Cinco de Mayo we put on
a big festival with crafts made by the students, dances, and games. At the e
nd of the year the school hosts a carnival to celebrate the hard work put in
during the school year, with a dunk tank being the most popular activity.My
students will use these five brightly colored Hokki stools in place of regul
ar, stationary, 4-legged chairs. As I will only have a total of ten in the c
lassroom and not enough for each student to have an individual one, they wil
l be used in a variety of ways. During independent reading time they will be
used as special chairs students will each use on occasion. I will utilize th
em in place of chairs at my small group tables during math and reading time
s. The rest of the day they will be used by the students who need the highes
t amount of movement in their life in order to stay focused on school.\r\n\r
\nWhenever asked what the classroom is missing, my students always say more
Hokki Stools. They can't get their fill of the 5 stools we already have. Whe
n the students are sitting in group with me on the Hokki Stools, they are al
ways moving, but at the same time doing their work. Anytime the students get
to pick where they can sit, the Hokki Stools are the first to be taken. Ther
e are always students who head over to the kidney table to get one of the st
ools who are disappointed as there are not enough of them. \r\n\r\nWe ask a
```

lot of students to sit for 7 hours a day. The Hokki stools will be a comprom
ise that allow my students to do desk work and move at the same time. These
stools will help students to meet their 60 minutes a day of movement by allo
wing them to activate their core muscles for balance while they sit. For man
y of my students, these chairs will take away the barrier that exists in sch
ools for a child who can't sit still.nannan

==========

How do you remember your days of school? Was it in a sterile environment wit
h plain walls, rows of desks, and a teacher in front of the room? A typical
day in our room is nothing like that. I work hard to create a warm inviting
themed room for my students look forward to coming to each day.\r\n\r\nMy cl
ass is made up of 28 wonderfully unique boys and girls of mixed races in Ark
ansas.\r\nThey attend a Title I school, which means there is a high enough p
ercentage of free and reduced-price lunch to qualify. Our school is an \"ope
n classroom\" concept, which is very unique as there are no walls separating
the classrooms. These 9 and 10 year-old students are very eager learners; th
ey are like sponges, absorbing all the information and experiences and keep
on wanting more.With these resources such as the comfy red throw pillows and
the whimsical nautical hanging decor and the blue fish nets, I will be able
to help create the mood in our classroom setting to be one of a themed nauti
cal environment. Creating a classroom environment is very important in the s
uccess in each and every child's education. The nautical photo props will be
used with each child as they step foot into our classroom for the first time
on Meet the Teacher evening. I'll take pictures of each child with them, hav
e them developed, and then hung in our classroom ready for their first day o
f 4th grade. This kind gesture will set the tone before even the first day
of school! The nautical thank you cards will be used throughout the year by
the students as they create thank you cards to their team groups.\r\n\r\nYou
r generous donations will help me to help make our classroom a fun, invitin
g, learning environment from day one.\r\n\r\nIt costs lost of money out of m
y own pocket on resources to get our classroom ready. Please consider helpin
g with this project to make our new school year a very successful one. Thank
you!nannan

==========

My kindergarten students have varied disabilities ranging from speech and la
nguage delays, cognitive delays, gross/fine motor delays, to autism. They ar
e eager beavers and always strive to work their hardest working past their l
imitations. \r\n\r\nThe materials we have are the ones I seek out for my stu
dents. I teach in a Title I school where most of the students receive free o
r reduced price lunch. Despite their disabilities and limitations, my stude
nts love coming to school and come eager to learn and explore.Have you ever
felt like you had ants in your pants and you needed to groove and move as yo
u were in a meeting? This is how my kids feel all the time. The want to be a
ble to move as they learn or so they say.Wobble chairs are the answer and I
love then because they develop their core, which enhances gross motor and in
Turn fine motor skills. \r\nThey also want to learn through games, my kids d
on't want to sit and do worksheets. They want to learn to count by jumping a
nd playing. Physical engagement is the key to our success. The number toss a
nd color and shape mats can make that happen. My students will forget they a
re doing work and just have the fun a 6 year old deserves.nannan

==========

The mediocre teacher tells. The good teacher explains. The superior teacher
demonstrates. The great teacher inspires. -William A. Ward\r\n\r\nMy school
has 803 students which is makeup is 97.6% African-American, making up the la
rgest segment of the student body. A typical school in Dallas is made up of
23.2% African-American students. Most of the students are on free or reduced
lunch. We aren't receiving doctors, lawyers, or engineers children from rich
backgrounds or neighborhoods. As an educator I am inspiring minds of young c
hildren and we focus not only on academics but one smart, effective, efficie
nt, and disciplined students with good character.In our classroom we can uti
lize the Bluetooth for swift transitions during class. I use a speaker which

```
doesn't amplify the sound enough to receive the message. Due to the volume o
f my speaker my students can't hear videos or books clearly and it isn't mak
ing the lessons as meaningful. But with the bluetooth speaker my students wi
ll be able to hear and I can stop, pause and replay it at any time.\r\nThe c
art will allow me to have more room for storage of things that are needed fo
r the day and has an extra part to it I can use.  The table top chart has al
l of the letter, words and pictures for students to learn about different le
tters and it is more accessible.nannan
==================================================
```

In [11]:

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

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

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

In [12]:

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

```
My kindergarten students have varied disabilities ranging from speech and la
nguage delays, cognitive delays, gross/fine motor delays, to autism. They ar
e eager beavers and always strive to work their hardest working past their l
imitations. \r\n\r\nThe materials we have are the ones I seek out for my stu
dents. I teach in a Title I school where most of the students receive free o
r reduced price lunch.  Despite their disabilities and limitations, my stude
nts love coming to school and come eager to learn and explore.Have you ever
felt like you had ants in your pants and you needed to groove and move as yo
u were in a meeting? This is how my kids feel all the time. The want to be a
ble to move as they learn or so they say.Wobble chairs are the answer and I
love then because they develop their core, which enhances gross motor and in
Turn fine motor skills. \r\nThey also want to learn through games, my kids d
o not want to sit and do worksheets. They want to learn to count by jumping
and playing. Physical engagement is the key to our success. The number toss
and color and shape mats can make that happen. My students will forget they
are doing work and just have the fun a 6 year old deserves.nannan
==================================================
```

In [13]:

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

```
My kindergarten students have varied disabilities ranging from speech and la
nguage delays, cognitive delays, gross/fine motor delays, to autism. They ar
e eager beavers and always strive to work their hardest working past their l
imitations.     The materials we have are the ones I seek out for my student
s. I teach in a Title I school where most of the students receive free or re
duced price lunch.  Despite their disabilities and limitations, my students
love coming to school and come eager to learn and explore.Have you ever felt
like you had ants in your pants and you needed to groove and move as you wer
e in a meeting? This is how my kids feel all the time. The want to be able t
o move as they learn or so they say.Wobble chairs are the answer and I love
then because they develop their core, which enhances gross motor and in Turn
fine motor skills.   They also want to learn through games, my kids do not w
ant to sit and do worksheets. They want to learn to count by jumping and pla
ying. Physical engagement is the key to our success. The number toss and col
or and shape mats can make that happen. My students will forget they are doi
ng work and just have the fun a 6 year old deserves.nannan
```

In [14]:

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

```
My kindergarten students have varied disabilities ranging from speech and la
nguage delays cognitive delays gross fine motor delays to autism They are ea
ger beavers and always strive to work their hardest working past their limit
ations The materials we have are the ones I seek out for my students I teach
in a Title I school where most of the students receive free or reduced price
lunch Despite their disabilities and limitations my students love coming to
school and come eager to learn and explore Have you ever felt like you had a
nts in your pants and you needed to groove and move as you were in a meeting
This is how my kids feel all the time The want to be able to move as they le
arn or so they say Wobble chairs are the answer and I love then because they
develop their core which enhances gross motor and in Turn fine motor skills
They also want to learn through games my kids do not want to sit and do work
sheets They want to learn to count by jumping and playing Physical engagemen
t is the key to our success The number toss and color and shape mats can mak
e that happen My students will forget they are doing work and just have the
fun a 6 year old deserves nannan
```

In [15]:

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

In [16]:

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

100%|██████████| 109248/109248 [01:11<00:00, 1521.09it/s]

In [17]:

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

Out[17]:

```
'my kindergarten students varied disabilities ranging speech language delays
cognitive delays gross fine motor delays autism they eager beavers always st
rive work hardest working past limitations the materials ones i seek student
s i teach title i school students receive free reduced price lunch despite d
isabilities limitations students love coming school come eager learn explore
have ever felt like ants pants needed groove move meeting this kids feel tim
e the want able move learn say wobble chairs answer i love develop core enha
nces gross motor turn fine motor skills they also want learn games kids not
want sit worksheets they want learn count jumping playing physical engagemen
t key success the number toss color shape mats make happen my students forge
t work fun 6 year old deserves nannan'
```

1.4 Preprocessing of `project_title`

In [18]:

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

100%|██████████| 109248/109248 [00:03<00:00, 33451.66it/s]

1.5 Preparing data for models

In [19]:

```
project_data['clean_essay']=preprocessed_essays
project_data['clean_titles']=preprocessed_titles
```

In [20]:

```
project_data.columns
```

Out[20]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_titl
e',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       'clean_categories', 'clean_subcategories', 'essay', 'clean_essay',
       'clean_titles'],
      dtype='object')
```

we are going to consider

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

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

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

1.5.1 Vectorizing Categorical data

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

In [21]:

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

```
['Care_Hunger', 'History_Civics', 'Health_Sports', 'Math_Science', 'Music_Ar
ts', 'AppliedLearning', 'SpecialNeeds', 'Literacy_Language', 'Warmth']
Shape of matrix after one hot encodig  (109248, 9)
```

In [22]:

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

```
['CommunityService', 'Health_LifeScience', 'ESL', 'Mathematics', 'NutritionE
ducation', 'Health_Wellness', 'SocialSciences', 'Music', 'CharacterEducatio
n', 'Civics_Government', 'VisualArts', 'Gym_Fitness', 'ForeignLanguages', 'H
istory_Geography', 'EnvironmentalScience', 'Literacy', 'TeamSports', 'Financ
ialLiteracy', 'Other', 'Warmth', 'PerformingArts', 'Care_Hunger', 'College_C
areerPrep', 'AppliedSciences', 'Economics', 'Literature_Writing', 'ParentInv
olvement', 'SpecialNeeds', 'EarlyDevelopment', 'Extracurricular']
Shape of matrix after one hot encodig  (109248, 30)
```

In [23]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
# you can do the similar thing with state, teacher_prefix and project_grade_category also
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(" ")# to handle nan

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

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

vectorizer = CountVectorizer(vocabulary=list(prefix.keys()), lowercase=False, binary=True)
prefix_one_hot = vectorizer.fit_transform(project_data['teacher_prefix'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",prefix_one_hot.shape)
```

```
['Ms.', 'Mrs.', 'Teacher', 'Dr.', 'Mr.']
Shape of matrix after one hot encodig  (109248, 5)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [0]:

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

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

In [0]:

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

1.5.2.2 TFIDF vectorizer

In [0]:

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

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

1.5.2.3 Using Pretrained Models: Avg W2V

In [0]:

```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

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

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

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

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

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

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

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


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickl

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


'''
```

Out[26]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4
```

```
084039\ndef (https://stackoverflow.com/a/38230349/4084039\ndef) loadGloveMod
el(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFil
e,\'r\', encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n
 splitLine = line.split()\n        word = splitLine[0]\n        embedding =
 np.array([float(val) for val in splitLine[1:]])\n        model[word] = embe
dding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmo
del = loadGloveModel(\'glove.42B.300d.txt\')\n\n# ========================
==\nOutput:\n    \nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\nDon
e. 1917495  words loaded!\n\n# ============================\n\nwords = []\nf
or i in preproced_texts:\n    words.extend(i.split(\' \'))\n\nfor i in prepr
oced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
 coupus", len(words))\nwords = set(words)\nprint("the unique words in the co
upus", len(words))\n\ninter_words = set(model.keys()).intersection(words)\np
rint("The number of words that are present in both glove vectors and our cou
pus",       len(inter_words),"(",np.round(len(inter_words)/len(words)*100,
3),"%)")\n\nwords_courpus = {}\nwords_glove = set(model.keys())\nfor i in wo
rds:\n    if i in words_glove:\n        words_courpus[i] = model[i]\nprint
("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pi
ckle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-
load-variables-in-python/\n\nimport (http://www.jessicayung.com/how-to-use-p
ickle-to-save-and-load-variables-in-python/\n\nimport) pickle\nwith open(\'g
love_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [0]:

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

In [0]:

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

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

```
100%|██████████████████████████████████████████████████████████████
██| 109248/109248 [00:31<00:00, 3508.17it/s]

109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [0]:

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

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

```
100%|██████████████████████████████████████████████████████████████
██| 109248/109248 [03:28<00:00, 524.21it/s]

109248
300
```

In [0]:

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

1.5.3 Vectorizing Numerical features

In [26]:

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

1.5.4 Merging all the above features

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

In [27]:

```
# please write all the code with proper documentation, and proper titles for each subsectio
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
X=project_data['clean_essay'].values
```

__ Computing Sentiment Scores__

In [28]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

#import nltk
#nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = X
'''a person is a person no matter how small dr seuss i teach the smallest students with the
for learning my students learn in many different ways using all of our senses and multiple
of techniques to help all my students succeed students in my class come from a variety of d
for wonderful sharing of experiences and cultures including native americans our school is
learners which can be seen through collaborative student project based learning in and out
in my class love to work with hands on materials and have many different opportunities to p
mastered having the social skills to work cooperatively with friends is a crucial aspect of
montana is the perfect place to learn about agriculture and nutrition my students love to r
in the early childhood classroom i have had several kids ask me can we try cooking with rea
and create common core cooking lessons where we learn important math and writing concepts w
food for snack time my students will have a grounded appreciation for the work that went in
of where the ingredients came from as well as how it is healthy for their bodies this proje
nutrition and agricultural cooking recipes by having us peel our own apples to make homemad
and mix up healthy plants from our classroom garden in the spring we will also create our o
shared with families students will gain math and literature skills as well as a life long e
nannan'''
NEG=[]
NEU=[]
POS=[]
COMP=[]
s={}
h=[]
for i in for_sentiment:

    ss = sid.polarity_scores(i)

    for k in ss:
        s={k:ss[k]}

        h.append(s)
# neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975
```

In [29]:

```
g=pd.DataFrame(h)
print(g.head(5))
```

```
   compound    neg    neu    pos
0       NaN    NaN    NaN  0.144
1       NaN    NaN  0.844    NaN
2    0.9694    NaN    NaN    NaN
3       NaN  0.012    NaN    NaN
4       NaN    NaN    NaN  0.283
```

In [30]:

```
compound=list(g['compound'].dropna())
#print(compound)
pos=list(g['pos'].dropna())
#print(pos)
neu=list(g['neu'].dropna())
#print(neu)
neg=list(g['neg'].dropna())
#print(neg)
```

In [31]:

```
q={'pos':pos,'neg':neg,'compound':compound,'neu':neu}
```

In [32]:

```
q=pd.DataFrame(q)
print(q.head(10))
```

```
   compound    neg    neu    pos
0    0.9694  0.012  0.844  0.144
1    0.9856  0.048  0.669  0.283
2    0.9816  0.122  0.659  0.219
3    0.9656  0.106  0.649  0.246
4    0.8524  0.066  0.791  0.143
5    0.9776  0.111  0.647  0.242
6    0.9743  0.079  0.680  0.241
7    0.9891  0.011  0.768  0.222
8    0.9975  0.009  0.630  0.361
9    0.9893  0.105  0.559  0.336
```

In [33]:

```
project_data['compound']=compound
project_data['neg']=neg
project_data['pos']=pos
project_data['neu']=neu
```

In [34]:

```
X_train=project_data
y_train=project_data['project_is_approved']
```

In [35]:

```
print("="*100)

print("train=>",X_train.shape, y_train.shape)

#print("test=>",X_test.shape, y_test.shape)

print("="*100)
```

```
====================================================================================================
train=> (109248, 26) (109248,)
====================================================================================================
```

numerical

In [36]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

'''encode numerical feature price'''
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

normalizer.fit(X_train['price'].values.reshape(-1,1)) # use code from sample

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))

#X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)

#print(X_test_price_norm.shape, y_test.shape)

print("="*100)
```

```
After vectorizations
(109248, 1) (109248,)
====================================================================================================
```

In [37]:

```
'''encode numerical feature teacher_number_of_previously_posted_projects'''

normalizer = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)

X_train_posted_norm= normalizer.transform(X_train['teacher_number_of_previously_posted_proj

#X_test_posted_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_proj

print("After vectorizations")
print(X_train_posted_norm.shape, y_train.shape)

#print(X_test_posted_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(109248, 1) (109248,)
====================================================================================================
```

In [38]:

```
'''encode numerical feature compound from sentimental'''

normalizer = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['compound'].values.reshape(-1,1))

X_train_comp_norm= normalizer.transform(X_train['compound'].values.reshape(-1,1))

#X_test_comp_norm = normalizer.transform(X_test['compound'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_comp_norm.shape, y_train.shape)

#print(X_test_comp_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(109248, 1) (109248,)
====================================================================================================
```

In [39]:

```
'''encode numerical feature pos from sentimental'''

normalizer = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['pos'].values.reshape(-1,1))

X_train_pos_norm= normalizer.transform(X_train['pos'].values.reshape(-1,1))

#X_test_pos_norm = normalizer.transform(X_test['pos'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_pos_norm.shape, y_train.shape)

#print(X_test_pos_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(109248, 1) (109248,)
====================================================================================================
```

In [40]:

```
'''encode numerical feature neg from sentimental'''

normalizer = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['neg'].values.reshape(-1,1))

X_train_neg_norm= normalizer.transform(X_train['neg'].values.reshape(-1,1))

#X_test_neg_norm = normalizer.transform(X_test['neg'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_neg_norm.shape, y_train.shape)

#print(X_test_neg_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(109248, 1) (109248,)
====================================================================================================
```

In [41]:

```
'''encode numerical feature neu from sentimental'''

normalizer = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['neu'].values.reshape(-1,1))

X_train_neu_norm= normalizer.transform(X_train['neu'].values.reshape(-1,1))

#X_test_neu_norm = normalizer.transform(X_test['neu'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_neu_norm.shape, y_train.shape)

#print(X_test_neu_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(109248, 1) (109248,)
====================================================================================================
```

In [42]:

```
'''encode numerical feature quantity from sentimental'''

normalizer = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['quantity'].values.reshape(-1,1))

X_train_quantity_norm= normalizer.transform(X_train['quantity'].values.reshape(-1,1))

#X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)

#print(X_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(109248, 1) (109248,)
====================================================================================================
```

In [43]:

```
#How to calculate number of words in a string in DataFrame: https://stackoverflow.com/a/374
'''count no of words in titles'''
title_word_count_train = X_train['clean_titles'].str.split().apply(len)
title_word_count_train = title_word_count_train.values
```

In [44]:

```
normalizer = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(title_word_count_train.reshape(-1,1))

X_train_title_norm= normalizer.transform(title_word_count_train.reshape(-1,1))

#X_test_title_norm = normalizer.transform(title_word_count_test.reshape(-1,1))

print("After vectorizations")
print(X_train_title_norm.shape, y_train.shape)

#print(X_test_title_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(109248, 1) (109248,)
====================================================================================================
```

In [45]:

```
#How to calculate number of words in a string in DataFrame: https://stackoverflow.com/a/374
'''count no of words in essays'''
essay_word_count_train = X_train['clean_essay'].str.split().apply(len)
essay_word_count_train = essay_word_count_train.values
```

In [46]:

```
normalizer = Normalizer()
# normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(essay_word_count_train.reshape(-1,1))

X_train_essay_norm= normalizer.transform(essay_word_count_train.reshape(-1,1))

#X_test_essay_norm = normalizer.transform(essay_word_count_test.reshape(-1,1))

print("After vectorizations")
print(X_train_essay_norm.shape, y_train.shape)

#print(X_test_essay_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(109248, 1) (109248,)
====================================================================================================
```

categorical_features

In [47]:

```
'''encoding project grade'''
from collections import Counter
my_counter = Counter()
for word in project_data['project_grade_category'].values:
    my_counter.update(word.split(","))

grade_dict = dict(my_counter)
grade_dict = dict(sorted(grade_dict.items(), key=lambda kv: kv[1]))
vectorizer_1 = CountVectorizer(vocabulary=list(grade_dict.keys()), lowercase=False, binary=
vectorizer_1.fit(X_train['project_grade_category'].values) # fit has to happen only on trai



# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer_1.transform(X_train['project_grade_category'].values.astype(
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np
#X_test_grade_ohe = vectorizer_1.transform(X_test['project_grade_category'].values.astype('

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)

#print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer_1.get_feature_names())
print("="*100)
```

```
After vectorizations
(109248, 4) (109248,)
['Grades PreK-2', 'Grades 3-5', 'Grades 9-12', 'Grades 6-8']
====================================================================================
========================
```

In [48]:

```
'''encode categorical feature teacher_prefix'''
vectorizer_2 = CountVectorizer(vocabulary=list(prefix.keys()), lowercase=False, binary=True
vectorizer_2.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer_2.transform(X_train['teacher_prefix'].values.astype('U'))
#https://stackoverflow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np
#X_test_teacher_ohe = vectorizer_2.transform(X_test['teacher_prefix'].values.astype('U'))

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)

#print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer_2.get_feature_names())
print("="*100)
```

```
After vectorizations
(109248, 5) (109248,)
['Ms.', 'Mrs.', 'Teacher', 'Dr.', 'Mr.']
====================================================================================
========================
```

In [49]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
from collections import Counter
my_counter = Counter()
for word in project_data['school_state'].values:
    my_counter.update(word.split())

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

In [50]:

```
for word in project_data['school_state'].values:
    my_counter.update(word.split(" "))

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

vectorizer_3 = CountVectorizer(vocabulary=list(state_dict.keys()), lowercase=False, binary=
vectorizer_3.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer_3.transform(X_train['school_state'].values)

#X_test_state_ohe = vectorizer_3.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)

#print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer_3.get_feature_names())
print("="*100)
```

```
After vectorizations
(109248, 51) (109248,)
['WV', 'KS', 'GA', 'NY', 'CO', 'HI', 'ID', 'NC', 'IL', 'IN', 'FL', 'WI', 'C
A', 'OK', 'AZ', 'IA', 'DE', 'VT', 'MN', 'TX', 'AL', 'UT', 'OR', 'CT', 'TN',
'RI', 'NJ', 'LA', 'NV', 'MS', 'SC', 'OH', 'ND', 'MI', 'NM', 'MA', 'NH', 'P
A', 'WY', 'SD', 'ME', 'KY', 'VA', 'WA', 'AK', 'MD', 'DC', 'AR', 'MO', 'NE',
'MT']
====================================================================================
========================
```

In [51]:

```
'''clean sub categories'''
vectorizer_5 = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False
vectorizer_5.fit(X_train['clean_subcategories'].values)
# we use the fitted CountVectorizer to convert the text to vector



X_train_subclean_cate_ohe = vectorizer_5.transform(X_train['clean_subcategories'].values)

#X_test_subclean_cate_ohe = vectorizer_5.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subclean_cate_ohe.shape, y_train.shape)

#print(X_test_subclean_cate_ohe.shape, y_test.shape)
print(vectorizer_5.get_feature_names())
print("="*100)
```

```
After vectorizations
(109248, 30) (109248,)
['CommunityService', 'Health_LifeScience', 'ESL', 'Mathematics', 'NutritionE
ducation', 'Health_Wellness', 'SocialSciences', 'Music', 'CharacterEducatio
n', 'Civics_Government', 'VisualArts', 'Gym_Fitness', 'ForeignLanguages', 'H
istory_Geography', 'EnvironmentalScience', 'Literacy', 'TeamSports', 'Financ
ialLiteracy', 'Other', 'Warmth', 'PerformingArts', 'Care_Hunger', 'College_C
areerPrep', 'AppliedSciences', 'Economics', 'Literature_Writing', 'ParentInv
olvement', 'SpecialNeeds', 'EarlyDevelopment', 'Extracurricular']
====================================================================================
========================
```

In [52]:

```
'''clean categories'''
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_4 = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, bi
vectorizer_4.fit(X_train['clean_categories'].values)
#print(vectorizer.get_feature_names())
 # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_cate_ohe = vectorizer_4.transform(X_train['clean_categories'].values)

#X_test_clean_cate_ohe = vectorizer_4.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_cate_ohe.shape, y_train.shape)

#print(X_test_clean_cate_ohe.shape, y_test.shape)
print(vectorizer_4.get_feature_names())
print("="*100)
```

```
After vectorizations
(109248, 9) (109248,)
['Care_Hunger', 'History_Civics', 'Health_Sports', 'Math_Science', 'Music_Ar
ts', 'AppliedLearning', 'SpecialNeeds', 'Literacy_Language', 'Warmth']
====================================================================================
========================
```

Assignment 11: TruncatedSVD

- step 1 Select the top 2k words from essay text and project_title (concatinate essay text with project title and then find the top 2k words) based on their `idf_` (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) values
- step 2 Compute the co-occurance matrix with these 2k words, with window size=5 (ref (https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/))

 step 3 Use TruncatedSVD (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html) on calculated co-occurance matrix and reduce its dimensions, choose the number of components (`n_components`) using elbow method (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/pca-code-example-using-non-visualization/)

> - The shape of the matrix after TruncatedSVD will be 2000*n, i.e. each row represents a vector form of the corresponding word.
> - Vectorize the essay text and project titles using these word vectors. (while vectorizing, do ignore all the words which are not in top 2k words)

- step 4 Concatenate these truncatedSVD matrix, with the matrix with features
  - **school_state** : categorical data
  - **clean_categories** : categorical data
  - **clean_subcategories** : categorical data
  - **project_grade_category** :categorical data
  - **teacher_prefix** : categorical data
  - **quantity** : numerical data
  - **teacher_number_of_previously_posted_projects** : numerical data
  - **price** : numerical data
  - **sentiment score's of each of the essay** : numerical data
  - **number of words in the title** : numerical data
  - **number of words in the combine essays** : numerical data
  - **word vectors calculated in** step 3 : numerical data
- step 5: Apply GBDT on matrix that was formed in step 4 of this assignment, **DO REFER THIS BLOG: XGBOOST DMATRIX (https://www.kdnuggets.com/2017/03/simple-xgboost-tutorial-iris-dataset.html)**
- step 6:Hyper parameter tuning (Consider any two hyper parameters)
  - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
  - Find the best hyper paramter using k-fold cross validation or simple cross validation data
  - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

2. TruncatedSVD

2.1 Selecting top 2000 words from `essay` and `project_title`

text

In [53]:

```
# merge texts
# merge two column text dataf
X_train["text"] = X_train["clean_titles"].map(str) +\
                  X_train["clean_essay"].map(str)
```

In [54]:

```
X_train['text'].shape
```

Out[54]:

```
(109248,)
```

In [55]:

```
# Please write all the code with proper documentation
'''tfidf_text'''
vectorizer_8 = TfidfVectorizer(min_df=10)
vectorizer_8.fit(X_train['text'].values)# fit for train

# transform for all
X_train_titles_tfidf= vectorizer_8.transform(X_train['text'].values)

#X_test_titles_tfidf = vectorizer_8.transform(X_test['clean_titles'].values)

print("After vectorizations")
print(X_train_titles_tfidf.shape, y_train.shape)

#print(X_test_titles_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(109248, 18330) (109248,)
====================================================================================
========================
```

In [56]:

```
# storing idf_values
idf = vectorizer_8.idf_
```

In [57]:

```
idf.shape
```

Out[57]:

```
(18330,)
```

In [58]:

```
# storing features names
feat=vectorizer_8.get_feature_names()
```

In [59]:

```
d=pd.DataFrame(zip(feat,idf))
```

In [60]:

```
# dataframe  of features and their values
d.head()
```

Out[60]:

	0	1
0	00	7.185285
1	000	5.895746
2	00am	10.116478
3	00pm	9.768172
4	03	9.605653

In [61]:

```
# sort on idf values
cd=d.sort_values(by=[1])
```

In [ ]:

```

```

In [62]:

```
# top 2k words
corpuse=cd.head(2000)
```

In [63]:

```
corpuse.shape
```

Out[63]:

```
(2000, 2)
```

In [64]:

```
corpuse.drop([1],axis=1)
```

Out[64]:

	0
15740	students
10804	nannan
14271	school
9392	learning
10765	my
3078	classroom
11057	not
9366	learn
16432	the
7701	help
16478	they
10006	many
10877	need
17824	we
18106	work
3323	come
17411	use
354	able
9794	love
894	also
4256	day
3057	class
16506	this
11438	our
9913	make
10950	new
18238	year
11274	one
16473	these
18173	would
...	...
5491	emphasis
10080	match
3993	creation

	0
14712	showcase
7328	green
4796	discovering
1857	bigger
16143	tactile
4401	delays
5350	efficiently
11441	out
17731	wall
4426	demands
11043	normal
16924	tremendously
17987	will
18260	yes
3653	consistent
3813	coordination
3645	consideration
12812	protect
11015	noise
6835	functioning
12876	publish
13605	reports
10164	meal
8737	involves
10825	nation
5006	donate
12470	prefer

2000 rows × 1 columns

In [ ]:

In [65]:

```
import sys
import math

import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score

# you might need to install this one
import xgboost as xgb

class XGBoostClassifier():
    def __init__(self, num_boost_round=10, **params):
        self.clf = None
        self.num_boost_round = num_boost_round
        self.params = params
        self.params.update({'objective': 'multi:softprob'})

    def fit(self, X, y, num_boost_round=None):
        num_boost_round = num_boost_round or self.num_boost_round
        self.label2num = {label: i for i, label in enumerate(sorted(set(y)))}
        dtrain = xgb.DMatrix(X, label=[self.label2num[label] for label in y])
        self.clf = xgb.train(params=self.params, dtrain=dtrain, num_boost_round=num_boost_r

    def predict(self, X):
        num2label = {i: label for label, i in self.label2num.items()}
        Y = self.predict_proba(X)
        y = np.argmax(Y, axis=1)
        return np.array([num2label[i] for i in y])

    def predict_proba(self, X):
        dtest = xgb.DMatrix(X)
        return self.clf.predict(dtest)

    def score(self, X, y):
        Y = self.predict_proba(X)[:,1]
        return roc_auc_score(y, Y)

    def get_params(self, deep=True):
        return self.params

    def set_params(self, **params):
        if 'num_boost_round' in params:
            self.num_boost_round = params.pop('num_boost_round')
        if 'objective' in params:
            del params['objective']
        self.params.update(params)
        return self


clf = XGBoostClassifier(eval_metric = 'auc', num_class = 2, nthread = 4,)
########################################################################
#               Change from here                                       #
########################################################################
parameters = {
    'num_boost_round': [100, 250, 500],
    'eta': [0.05, 0.1, 0.3],
    'max_depth': [6, 9, 12],
    'subsample': [0.9, 1.0],
    'colsample_bytree': [0.9, 1.0],
```

```
}

clf = GridSearchCV(clf, parameters)
X = np.array([[1,2], [3,4], [2,1], [4,3], [1,0], [4,5]])
Y = np.array([0, 1, 0, 1, 0, 1])
clf.fit(X, Y)

print("clf.scores =" ,max(clf.cv_results_['mean_test_score']))
best_parameters = max(clf.cv_results_, key=lambda x: x[1])
best_parameters=clf.best_params_
#print('score:', score)
print("best_parameters:",best_parameters)
print("best_score:",clf.best_score_)
```

```
clf.scores = 1.0
best_parameters: {'eta': 0.05, 'subsample': 0.9, 'max_depth': 6, 'num_boost_
round': 100, 'colsample_bytree': 0.9}
best_score: 1.0
```

2.2 Computing Co-occurance matrix

In [66]:

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

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

In [67]:

```
from sklearn.decomposition import TruncatedSVD
```

In [68]:

```
# getting top 2kwords in top_words
top_words=corpuse[0]
```

In [69]:

```
top_words
```

Out[69]:

```
15740         students
10804           nannan
14271           school
9392          learning
10765                my
3078         classroom
11057              not
9366             learn
16432              the
7701              help
16478             they
10006             many
10877             need
17824               we
18106             work
3323              come
17411              use
354               able
9794              love
894               also
4256               day
3057             class
16506             this
11438              our
9913              make
10950              new
18238             year
11274              one
16473            these
18173            would
            ...
5491          emphasis
10080            match
3993          creation
14712         showcase
7328             green
4796       discovering
1857             bigger
16143         tactile
4401           delays
5350       efficiently
11441              out
17731             wall
4426           demands
11043           normal
16924     tremendously
17987              will
18260              yes
3653        consistent
3813      coordination
3645     consideration
12812          protect
11015            noise
6835       functioning
12876          publish
```

```
13605          reports
10164             meal
8737           involves
10825            nation
5006             donate
12470            prefer
Name: 0, Length: 2000, dtype: object
```

In [70]:

```
top_words=list(top_words)
```

In [92]:

```
w=5
```

In [69]:

```
# an array null arrat of 2k X 2k
m = np.zeros([2000,2000])
```

In [70]:

```
# to get index of words in sentence
def duplicate(list,item):
    return[i for i ,x in enumerate(list) if x==item]
```

code to get co-occurance matrix

In [71]:

```
for sent in X_train['text'].values:

    word=sent.split(" ")


    for i  in top_words:
        index=duplicate(word,i)




        for k in index:
            for ind in range(max(k-w,0),min(w+k+1,len(word))):
                if(word[ind] in top_words):
                    if(word[ind]!=i):
                        l=top_words.index(word[ind])
                        m[top_words.index(i)][l]+=1
```

In [72]:

```
print(m)
```

```
[[0.00000e+00 1.75750e+04 1.26472e+05 ... 3.43000e+02 3.16000e+02
  6.22000e+02]
 [1.75750e+04 0.00000e+00 4.21000e+03 ... 3.00000e+01 1.29000e+02
  1.50000e+01]
 [1.26472e+05 4.21000e+03 0.00000e+00 ... 2.38000e+02 1.02000e+02
  2.80000e+01]
 ...
 [3.43000e+02 3.00000e+01 2.38000e+02 ... 0.00000e+00 0.00000e+00
  0.00000e+00]
 [3.16000e+02 1.29000e+02 1.02000e+02 ... 0.00000e+00 0.00000e+00
  0.00000e+00]
 [6.22000e+02 1.50000e+01 2.80000e+01 ... 0.00000e+00 0.00000e+00
  0.00000e+00]]
```

In [73]:

```
pd_cooc=pd.DataFrame(m,index=top_words,columns=top_words)
```

In [74]:

```
pd_cooc
```

Out[74]:

	students	nannan	school	learning	my	classroom	not	learn	
students	0.0	17575.0	126472.0	96541.0	138387.0	82987.0	62537.0	74429.0	67
nannan	17575.0	0.0	4210.0	10182.0	296.0	7277.0	2895.0	4294.0	
school	126472.0	4210.0	0.0	17411.0	27622.0	13561.0	20727.0	21211.0	19
learning	96541.0	10182.0	17411.0	0.0	20513.0	21904.0	10898.0	9933.0	13
my	138387.0	296.0	27622.0	20513.0	0.0	18562.0	9780.0	19172.0	1
classroom	82987.0	7277.0	13561.0	21904.0	18562.0	0.0	12186.0	12021.0	14
not	62537.0	2895.0	20727.0	10898.0	9780.0	12186.0	0.0	8466.0	8
learn	74429.0	4294.0	21211.0	9933.0	19172.0	12021.0	8466.0	0.0	8
the	67953.0	569.0	19686.0	13656.0	1335.0	14595.0	8013.0	8867.0	
help	65067.0	8005.0	9323.0	15447.0	7708.0	14066.0	7959.0	11895.0	13
they	38648.0	627.0	19668.0	18242.0	7100.0	9132.0	10570.0	26631.0	2
many	69148.0	1427.0	23168.0	9123.0	6649.0	10743.0	17143.0	6938.0	4
need	55283.0	3183.0	9467.0	9497.0	16940.0	9893.0	7167.0	8233.0	5
we	32639.0	690.0	24083.0	10616.0	2173.0	11371.0	7684.0	7853.0	1
work	48360.0	3516.0	11368.0	7656.0	7873.0	8352.0	6436.0	6973.0	7
come	52291.0	1563.0	27868.0	5204.0	11945.0	6209.0	6599.0	12702.0	5
use	44059.0	2725.0	5157.0	9515.0	8536.0	11037.0	5819.0	5946.0	8
able	44176.0	2193.0	4871.0	8624.0	5880.0	7955.0	8518.0	6136.0	4
love	43506.0	2283.0	11649.0	17407.0	17268.0	7825.0	4076.0	14046.0	4
also	28984.0	1227.0	5969.0	7212.0	3669.0	5063.0	7158.0	4762.0	5
day	34415.0	2372.0	20916.0	7841.0	8332.0	9547.0	5157.0	12104.0	4
class	34085.0	2417.0	7112.0	6547.0	8791.0	4040.0	5205.0	5056.0	7
this	29061.0	710.0	7696.0	7081.0	864.0	7056.0	5219.0	3942.0	
our	33214.0	104.0	37777.0	6523.0	735.0	8079.0	4366.0	4849.0	
make	26047.0	3474.0	6797.0	10935.0	4955.0	7864.0	3771.0	4341.0	4
new	26716.0	2224.0	10327.0	9832.0	6377.0	6706.0	3949.0	10644.0	4
year	27232.0	2206.0	20431.0	4850.0	5049.0	6085.0	3854.0	3553.0	3
one	23773.0	1269.0	12257.0	4560.0	4142.0	6199.0	4322.0	3987.0	3
these	31629.0	365.0	4566.0	6537.0	476.0	5232.0	4221.0	4443.0	
would	32455.0	2525.0	4508.0	6450.0	4443.0	10450.0	6825.0	3118.0	5
...	...	...	...	...	...	...	...	...	
emphasis	275.0	9.0	257.0	150.0	61.0	88.0	35.0	28.0	
match	347.0	34.0	40.0	128.0	65.0	86.0	81.0	33.0	
creation	396.0	47.0	59.0	83.0	39.0	68.0	43.0	40.0	
showcase	495.0	45.0	114.0	217.0	71.0	64.0	45.0	33.0	

	students	nannan	school	learning	my	classroom	not	learn
green	443.0	36.0	256.0	40.0	72.0	81.0	49.0	50.0
discovering	397.0	21.0	63.0	227.0	113.0	70.0	22.0	88.0
bigger	331.0	43.0	104.0	44.0	51.0	86.0	77.0	34.0
tactile	488.0	30.0	12.0	217.0	88.0	49.0	29.0	158.0
delays	581.0	2.0	62.0	197.0	130.0	67.0	53.0	32.0
efficiently	433.0	85.0	25.0	79.0	47.0	139.0	62.0	85.0
out	423.0	12.0	121.0	36.0	56.0	85.0	32.0	29.0
wall	408.0	32.0	78.0	45.0	35.0	163.0	104.0	14.0
demands	402.0	21.0	146.0	59.0	67.0	84.0	78.0	30.0
normal	408.0	35.0	170.0	60.0	66.0	166.0	130.0	41.0
tremendously	526.0	99.0	91.0	112.0	108.0	169.0	64.0	34.0
will	336.0	34.0	89.0	49.0	60.0	58.0	44.0	51.0
yes	311.0	20.0	117.0	60.0	69.0	100.0	77.0	78.0
consistent	448.0	26.0	217.0	89.0	89.0	125.0	117.0	64.0
coordination	316.0	40.0	16.0	52.0	44.0	17.0	43.0	55.0
consideration	230.0	534.0	47.0	70.0	33.0	102.0	22.0	29.0
protect	350.0	66.0	39.0	42.0	63.0	86.0	56.0	38.0
noise	522.0	60.0	35.0	93.0	43.0	213.0	104.0	36.0
functioning	536.0	36.0	78.0	69.0	129.0	129.0	89.0	33.0
publish	481.0	45.0	55.0	48.0	68.0	91.0	48.0	38.0
reports	362.0	50.0	56.0	44.0	42.0	71.0	37.0	40.0
meal	485.0	23.0	310.0	30.0	53.0	38.0	196.0	41.0
involves	294.0	9.0	60.0	173.0	70.0	216.0	31.0	47.0
nation	343.0	30.0	238.0	47.0	60.0	47.0	69.0	38.0
donate	316.0	129.0	102.0	45.0	32.0	154.0	101.0	28.0
prefer	622.0	15.0	28.0	100.0	54.0	75.0	83.0	63.0

2000 rows × 2000 columns

In [71]:

```
'''i have already saved this co-occrance matrix in file name coo_mat.csv'''
coo_mat=pd.read_csv('coo_matrix.csv')
```

In [72]:

```
coo_mat.reset_index(drop=True, inplace=True)
```

In [73]:

```
mat=coo_mat.values
```

In [74]:

```
type(mat)
```

Out[74]:

```
numpy.ndarray
```

In [75]:

```
# droping index or zero row
m=mat[:,1:]
```

In [76]:

```
m
```

Out[76]:

```
array([[0.0, 17575.0, 126472.0, ..., 343.0, 316.0, 622.0],
       [17575.0, 0.0, 4210.0, ..., 30.0, 129.0, 15.0],
       [126472.0, 4210.0, 0.0, ..., 238.0, 102.0, 28.0],
       ...,
       [343.0, 30.0, 238.0, ..., 0.0, 0.0, 0.0],
       [316.0, 129.0, 102.0, ..., 0.0, 0.0, 0.0],
       [622.0, 15.0, 28.0, ..., 0.0, 0.0, 0.0]], dtype=object)
```

2.3 Applying TruncatedSVD and Calculating Vectors for `essay` and `project_title`

In [77]:

```
from sklearn.decomposition import TruncatedSVD
```

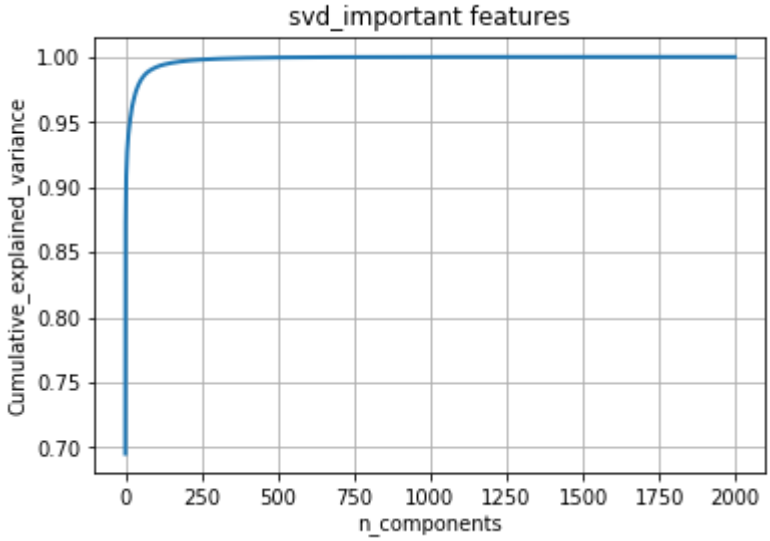
In [78]:

```
svd=TruncatedSVD(n_components=1999)
svd_tf=svd.fit_transform(m)
percentage_var_explained_tf = svd.explained_variance_ / np.sum(svd.explained_variance_);

cum_var_explained_tf = np.cumsum(percentage_var_explained_tf)

# Plot the TSVD spectrum
plt.figure(1, figsize=(6, 4))

plt.clf()
plt.title("svd_important features")
plt.plot(cum_var_explained_tf, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()
```



In [ ]:

- Only 250 component are enough to describe all information
- taking only 250 component out of 2k

In [79]:

```
X_train.columns
```

Out[79]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_titl
e',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approve
d',
       'clean_categories', 'clean_subcategories', 'essay', 'clean_essay',
       'clean_titles', 'price', 'quantity', 'compound', 'neg', 'pos', 'neu',
       'text'],
      dtype='object')
```

In [80]:

```
svd=TruncatedSVD(n_components=250)
svd_2k=svd.fit_transform(m)
```

In [81]:

```
svd_2k
```

Out[81]:

```
array([[ 3.16821307e+05,  2.19778277e+05,  1.77080628e+03, ...,
        -5.29371082e+00, -6.22017511e+00,  8.53089307e+00],
       [ 2.70709169e+04, -3.75944251e+03, -8.10641442e+03, ...,
        -3.97779721e+01, -6.33280618e+01,  1.02023561e+01],
       [ 1.51020558e+05, -5.35957275e+04,  3.65858092e+04, ...,
         5.88102238e+00, -1.00441011e+01, -3.90859097e-01],
       ...,
       [ 4.76100748e+02, -9.93605961e+01,  1.30591624e+02, ...,
         2.15229931e+00, -4.37894138e+00,  8.72450682e+00],
       [ 4.75419789e+02, -7.56055071e+01, -9.28985564e+01, ...,
        -2.33104066e+00, -1.09185283e+01, -4.78735702e+00],
       [ 6.26235816e+02, -3.24559983e+02, -9.66846727e+01, ...,
        -4.96120656e+00, -2.47107008e+01,  9.66284257e+00]])
```

In [82]:

```
svd_dataframe=pd.DataFrame(svd_2k,index=top_words)
```

In [83]:

```
svd_dataframe
```

Out[83]:

	0	1	2	3	4	
students	316821.307076	219778.277398	1770.806283	1310.576492	-2286.129335	1025
nannan	27070.916891	-3759.442506	-8106.414417	-2779.923638	-1841.662456	652(
school	151020.557743	-53595.727498	36585.809192	59509.906838	1290.990421	9147
learning	114760.520266	-40413.643559	-13557.226678	2348.309181	1506.739045	-16547
my	129804.054143	-80170.720796	4011.300078	-16902.866007	6055.334889	18656
classroom	103098.951269	-30713.596946	-10055.418423	8537.285310	-2470.003634	-1776
not	80102.579488	-21993.297729	295.378408	-3685.235583	-3675.118755	-4508
learn	92607.650758	-28988.444087	-1470.357972	-752.644130	18385.529592	-1717(
the	77622.796541	-30640.016953	-6049.140378	-12468.747475	-7910.737223	1021(
help	78301.253465	-26607.595602	-16509.133594	191.329895	-7159.722851	1992
they	70404.692900	-512.174701	-1890.063072	-8343.395255	18431.970693	2315'
many	76642.160397	-31985.622581	14080.263401	-7298.599870	-3409.274270	307(
need	66008.051063	-21852.840761	-7250.946730	3426.897102	-1646.808456	-820'
we	53606.203349	-4348.533985	3745.678991	-12990.933159	-1675.765711	10055
work	58408.154194	-19803.533522	-4576.139700	-796.079427	1575.133591	-4506
come	62687.559144	-20228.600741	19455.758334	-10105.661899	9436.126486	-8117
use	53710.282823	-17406.914918	-11655.859231	2428.301833	-4671.945498	-404'
able	50604.572060	-19707.516022	-10440.183752	282.058722	-2534.079363	-189
love	59491.533029	-12675.006875	-5611.153070	674.672922	10783.595230	-7546
also	38828.890214	-9482.903241	-6400.753943	569.105566	-3878.611363	-1949
day	50915.777006	-8065.269411	4912.861441	-8301.777038	11257.448108	-251'
class	43664.077732	-11669.149635	-1570.919016	3210.369697	108.636699	-189'
this	36452.231408	-11150.944770	-5391.566024	-3572.152960	-4211.083618	4932
our	45295.478753	-9729.086454	16909.902729	-24127.260428	-681.953228	1296
make	36653.146905	-7243.241972	-5834.729361	-585.630089	285.668341	-144
new	39842.285391	-6121.856033	-2891.956055	-1706.799362	5544.740989	233
year	38193.413458	-7354.587815	4503.856993	-10143.383292	926.968253	-4022
one	32972.588465	-6947.757558	2911.400247	-3361.859584	204.905543	-927
these	35145.677953	-15031.442521	-6254.916160	-2575.863296	-3899.421875	5058
would	41957.186702	-11386.462445	-10389.568113	177.974704	-4264.691357	527
...	...	...	...	...	...	
emphasis	477.665531	-12.485642	13.085281	-101.436987	-68.753948	-25
match	475.504073	-101.969603	-128.328720	23.086727	-61.795362	-8
creation	475.900611	-159.794440	-103.703640	6.858457	-64.295296	-16

	0	1	2	3	4	
showcase	624.425331	-178.872521	-109.222421	-13.338251	-34.284690	12
green	585.933705	-144.386705	2.711820	-88.603430	-56.044523	-8
discovering	555.019487	-123.439294	-91.251491	30.249541	152.606853	-1
bigger	483.380932	-89.455631	-48.846204	3.735752	11.238003	-2
tactile	603.809315	-178.558493	-157.325536	42.404932	-2.703594	5
delays	617.251590	-268.340750	18.199864	37.954403	-15.554355	-4
efficiently	561.624422	-146.362206	-195.789818	17.615621	-95.155310	7
out	455.660223	-200.455787	41.266925	-14.875296	-25.358880	-1
wall	525.290082	-149.213332	-105.347052	13.757922	-94.675251	-5
demands	518.850487	-136.414098	2.144350	-24.170581	-38.389310	-2
normal	552.442813	-127.680297	-9.107659	-55.153663	-5.778635	-4
tremendously	656.513082	-188.197478	-168.002577	-5.646213	-121.768875	-7
will	457.518871	-111.193930	-35.812728	45.024244	-13.558222	-7
yes	452.415609	-83.213234	-8.089630	-18.954302	34.911094	-1
consistent	617.510150	-127.067929	-11.022914	-79.911171	-28.932639	-3
coordination	423.425210	-102.820357	-183.350449	32.081410	-107.722588	-3
consideration	345.401562	-61.258795	-126.624199	4.350548	-67.878604	-4
protect	512.161580	-83.930943	-151.389203	64.962813	-106.493571	-6
noise	618.074138	-216.418954	-168.042432	12.861782	-124.368676	-1
functioning	617.227394	-224.207713	2.003574	42.439443	-49.823334	-3
publish	603.865415	-179.506176	-190.018642	16.716040	-93.562396	-4
reports	469.974088	-127.955885	-139.384355	0.835378	-83.155482	-2
meal	625.710617	-164.228585	277.708889	-84.748801	-82.601128	-3
involves	480.160508	-42.575656	-84.302916	16.029371	23.847793	2
nation	476.100748	-99.360596	130.591624	-49.239621	0.654955	-6
donate	475.419789	-75.605507	-92.898556	-26.295822	-92.781250	-1
prefer	626.235816	-324.559983	-96.684673	22.624111	-13.911229	

2000 rows × 250 columns

In [84]:

```
svd=svd_2k
```

In [85]:

```
'''average word to vector test titles'''
avg_w2v_vectors_traintitle = []; # the avg-w2v for each sentence/review is stored in this l
for sentence in tqdm(X_train['clean_titles']): # for each review/sentence
    vector = np.zeros(250) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in top_words:
            vector += svd_dataframe.loc[word ,:]# using our own word vector
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_traintitle.append(vector)

print(len(avg_w2v_vectors_traintitle))
print(len(avg_w2v_vectors_traintitle[0]))
```

```
100%|██████████| 109248/109248 [03:04<00:00, 591.43it/s]

109248
250
```

In [86]:

```
'''average word to vector  essay'''
avg_w2v_vectors_trainessay = []; # the avg-w2v for each sentence/review is stored in this l
for sentence in tqdm(X_train['clean_essay']): # for each review/sentence
    vector = np.zeros(250) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in top_words:
            vector += svd_dataframe.loc[word ,:]# using our own word vector
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_trainessay.append(vector)

print(len(avg_w2v_vectors_trainessay))
print(len(avg_w2v_vectors_trainessay[0]))
```

```
100%|██████████| 109248/109248 [1:40:20<00:00, 18.15it/s]

109248
250
```

In [ ]:

```

```

2.4 Merge the features from step 3 and step 4

In [87]:

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

#merging to get final data matrix
from scipy.sparse import hstack
X_tr = hstack(( X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe, X_train_price_no

print("After vectorizations")
print(X_tr.shape, y_train.shape)

#print(X_test_titles_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(109248, 608) (109248,)
====================================================================================
========================
```

2.5 Apply XGBoost on the Final Features from the above section

https://xgboost.readthedocs.io/en/latest/python/python_intro.html (https://xgboost.readthedocs.io/en/latest/python/python_intro.html)

In [88]:

```
# No need to split the data into train and test(cv)
# use the Dmatrix and apply xgboost on the whole data
# please check the Quora case study notebook as reference

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

from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import uniform
import xgboost as xgb
```

In [89]:

```
# randomize search parameter tunning
param_dist = {"n_estimators":sp_randint(105,125),
"max_depth": sp_randint(10,15),
"min_samples_split": sp_randint(110,190),
"min_samples_leaf": sp_randint(25,65)}
xgb_model = xgb.XGBClassifier(random_state=25)
rf_random = RandomizedSearchCV(xgb_model, param_distributions=param_dist,
n_iter=5,cv=10,scoring='roc_auc',random_state=25,n_jobs=-1,verbose=1)
rf_random.fit(X_tr,y_train)
#print("train_auc is",rf_random.cv_results_["mean_train_score"])
```

```
Fitting 10 folds for each of 5 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 338.3min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 482.3min finished
```

Out[89]:

```
RandomizedSearchCV(cv=10, error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step
=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=25, reg_alpha...
                                          'min_samples_leaf': <scipy.stats._di
stn_infrastructure.rv_frozen object at 0x7f2f22c70550>,
                                          'min_samples_split': <scipy.stats._d
istn_infrastructure.rv_frozen object at 0x7f2f22c70400>,
                                          'n_estimators': <scipy.stats._distn_
infrastructure.rv_frozen object at 0x7f2f22c700f0>},
                   pre_dispatch='2*n_jobs', random_state=25, refit=True,
                   return_train_score=False, scoring='roc_auc', verbose=1)
```

In [91]:

```
print("beat_parameter=",rf_random.best_params_)
print("AUC_SCORE=", rf_random.best_score_)
```

```
beat_parameter= {'min_samples_leaf': 28, 'max_depth': 14, 'min_samples_spli
t': 111, 'n_estimators': 121}
AUC_SCORE= 0.6569038773685419
```

3. Conclusion

In [92]:

```
# Please write down few lines about what you observed from this assignment.
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytab
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = [ "Model","train_AUC"]
```

In [93]:

```
x.add_row(['XGBOOST',0.6569038773685419])
```

In [94]:

```
print(x)
```

```
+---------+--------------------+
|  Model  |     train_AUC      |
+---------+--------------------+
| XGBOOST | 0.6569038773685419 |
+---------+--------------------+
```

In [ ]:

```

```