DonorsChoose

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

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

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

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

About the DonorsChoose Data Set

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

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `My students need hands on literacy materials to manage sensory needs!`
`project_essay_1`	First application essay*
`project_essay_2`	Second application essay*
`project_essay_3`	Third application essay*

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

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

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

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

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

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

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

Notes on the Essay Data

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

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

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

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

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

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

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

1.1 Reading Data

In [2]:

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

In [3]:

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

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

In [4]:

```
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/47301924/4084039

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

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

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

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

1.3 preprocessing of `project_subject_subcategories`

In [6]:

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

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

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

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

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

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

1.3 Text preprocessing

In [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_submitted_datetime	project_grade_cate
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades P
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grade

1.4.2.3 Using Pretrained Models: TFIDF weighted W2V

In [9]:

```
# 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. W
e are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of langua
ge to our school. \r\n\r\n We have over 24 languages represented in our English Learner program wi
th 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 cultures, beliefs, and respect.\"The limits of your language are the limits o
f your world.\"-Ludwig Wittgenstein  Our English learner's have a strong support system at home th
at begs for more resources.  Many times our parents are learning to read and speak English along s
ide 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 hom
e 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 En
glish 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 player will have the
opportunity to check out a dvd player to use for the year.  The plan is to use these videos and ed
ucational dvd's for the years to come for other EL students.\r\nnannan
```

ucational dvd's for the years to come for other EL students.\r\nnannan

==

The 51 fifth grade students that will cycle through my classroom this year all love learning, at least most of the time. At our school, 97.3% of the students receive free or reduced price lunch. Of the 560 students, 97.3% are minority students. \r\nThe school has a vibrant community that loves to get together and celebrate. Around Halloween there is a whole school parade to show 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 end 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 regular, stationary, 4-legged chairs. As I will only have a total of ten in the classroom and not enough for each student to have an individual one, they will 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 them in place of chairs at my small group tables during math and reading times. The rest of the day they will be used by the students who need the highest 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. When the students are sitting in group with me on the Hokki Stools, they are always 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. There are always students who head over to the kidney table to get one of the stools 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 compromise 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 allowing them to activate their core muscles for balance while they sit. For many of my students, these chairs will take away the barrier that exists in schools for a child who can't sit still.nannan

==

How do you remember your days of school? Was it in a sterile environment with 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 class is made up of 28 wonderfully unique boys and girls of mixed races in Arkansas.\r\nThey attend a Title I school, which means there is a high enough percentage of free and reduced-price lunch to qualify. Our school is an \"open 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; they 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 nautical environment. Creating a classroom environment is very important in the success 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, have them developed, and then hung in our classroom ready for their first day of 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\nYour generous donations will help me to help make our classroom a fun, inviting, learning environment from day one.\r\n\r\nIt costs lost of money out of my own pocket on resources to get our classroom ready. Please consider helping 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 language delays, cognitive delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work their hardest working past their limitations. \r\n\r\nThe 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 ants 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 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 don't 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

==

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 largest 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 children and we focus not only on academics but one smart, effective, efficient, and disciplined students with good character.In our classroom we can utilize 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 of my speaker my students can't hear videos or books clearly and it isn't making the lessons as meaningful. But with the bluetooth speaker my students will be able to hear and I can stop, pause and replay it at any time.\r\nThe cart will allow me to have more room for storage of things that are needed for the day and has an extra part to it I can use. The table top chart has all of the letter, words and pictures for students to learn about different letters and it is more accessible.nannan

==

In [10]:

```
# 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 [11]:

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

```
My kindergarten students have varied disabilities ranging from speech and language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir hardest working past their limitations. \r\n\r\nThe 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 pr
ice 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 gr
oove 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 learn or so they say.Wobble chairs are the answer and I love then because they dev
elop their core, which enhances gross motor and in Turn fine motor skills. \r\nThey also want to l
earn through games, my kids do not want to sit and do worksheets. They want to learn to count by j
umping and playing. Physical engagement is the key to our success. The number toss and color and s
hape 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 [12]:

```
# \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 language delays, cogniti
ve delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work th
eir hardest working past their limitations.     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 eag
er to learn and explore.Have you ever felt like you had ants in your pants and you needed to groov
e 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 learn or so they say.Wobble chairs are the answer and I love then because they develo
p their core, which enhances gross motor and in Turn fine motor skills.   They also want to learn t
hrough games, my kids do 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 ma
ts can make that happen. My students will forget they are doing work and just have the fun a 6 yea
r old deserves.nannan
```

In [13]:

```
#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 language delays cognitiv
My kindergarten students have varied disabilities ranging from speech and language delays cognitiv
e delays gross fine motor delays to autism They are eager beavers and always strive to work their
hardest working past their limitations The materials we have are the ones I seek out for my studen
ts 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 lea
rn and explore Have you ever felt like you had ants in your pants and you needed to groove and mov
e as you were in a meeting This is how my kids feel all the time The want to be able to move as th
ey learn or so they say Wobble chairs are the answer and I love then because they develop their co
re 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 worksheets They want to learn to count by jumping and playing Ph
ysical 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 nan
nan
```

In [14]:

```python
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'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', "do
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"]
```

In [15]:

```python
# 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
[00:57<00:00, 1885.47it/s]
```

In [16]:

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

Out[16]:

```
'my kindergarten students varied disabilities ranging speech language delays cognitive delays gros
```

```
'my kindergarten students varied disabilities ranging speech language delays cognitive delays gros
s fine motor delays autism they eager beavers always strive work hardest working past limitations
the materials ones i seek students i teach title i school students receive free reduced price lunc
h despite disabilities limitations students love coming school come eager learn explore have ever
felt like ants pants needed groove move meeting this kids feel time the want able move learn say w
obble chairs answer i love develop core enhances gross motor turn fine motor skills they also want
learn games kids not want sit worksheets they want learn count jumping playing physical engagement
key success the number toss color shape mats make happen my students forget work fun 6 year old de
serves nannan'
```

1.4 Preprocessing of `project_title`

In [17]:

```
# similarly you can preprocess the titles also
# Using above lines of code for preprocessing Title text

from tqdm import tqdm
preprocessed_title = []

for sentance in tqdm(project_data['project_title'].values):
    sentTitle = decontracted(sentance)
    sentTitle = sentTitle.replace('\\r', ' ')
    sentTitle = sentTitle.replace('\\"', ' ')
    sentTitle = sentTitle.replace('\\n', ' ')
    sentTitle = re.sub('[^A-Za-z0-9]+', ' ', sentTitle)
    # https://gist.github.com/sebleier/554280
    sentTitle = ' '.join(e for e in sentTitle.split() if e not in stopwords)
    preprocessed_title.append(sentTitle.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████| 109248/109248
[00:02<00:00, 39446.00it/s]
```

1.5 Preparing data for models

In [18]:

```
project_data.columns
```

Out[18]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', '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',
       'clean_categories', 'clean_subcategories', 'essay'],
      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

1.5.1 Vectorizing Categorical data

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

In [19]:

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

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

In [20]:

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

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', '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 matrix after one hot encodig  (109248, 30)
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

In [21]:

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

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

In [22]:

```
# Similarly you can vectorize for title also
# Using above lines of code

vectorizerTitle = CountVectorizer(min_df=10)
text_bow_title = vectorizerTitle.fit_transform(preprocessed_title)
print("Shape of matrix after one hot encodig ",text_bow_title.shape)
```

```
Shape of matrix after one hot encodig  (109248, 3329)
```

1.5.2.2 TFIDF vectorizer

In [23]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_essay = TfidfVectorizer(min_df=10 , use_idf = True)
vectorizer_essay = vectorizer_essay.fit(preprocessed_essays)
```

```
print("some sample features(unique words in the corpus)",vectorizer_essay.get_feature_names()[0:10
])
print('='*50)
final_tfidf_essay = vectorizer_essay.transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",final_tfidf_essay.shape)
```

```
some sample features(unique words in the corpus) ['00', '000', '00am', '00pm', '03', '10', '100',
'1000', '100s', '100th']
==================================================
Shape of matrix after one hot encodig  (109248, 16623)
```

In [24]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer_title = TfidfVectorizer(min_df=10 , use_idf = True)
vectorizer_title = vectorizer_title.fit(preprocessed_title)

print("some sample features(unique words in the corpus)",vectorizer_title.get_feature_names()[0:10
])
print('='*50)
final_tfidf_title = vectorizer_title.transform(preprocessed_title)
print("Shape of matrix after one hot encodig ",final_tfidf_title.shape)
```

```
some sample features(unique words in the corpus) ['000', '04', '05', '10', '100', '101', '11',
'12', '123', '12th']
==================================================
Shape of matrix after one hot encodig  (109248, 3329)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [25]:

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

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

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

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

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

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

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

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


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/

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


'''
```

Out[25]:

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

In [26]:

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

In [27]:

```
# 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
[00:34<00:00, 3153.79it/s]
```

```
109248
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [28]:

```
# 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 [29]:

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

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

```
100%|████████████████████████████████████████████████████████████████████████████| 109248/109248
[03:53<00:00, 468.19it/s]
```

```
109248
300
```

In [30]:

```
# Similarly you can vectorize for title also

tfidf_model_title = TfidfVectorizer()
tfidf_model_title.fit(preprocessed_title)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_title = dict(zip(tfidf_model_title.get_feature_names(), list(tfidf_model_title.idf_)))
tfidf_words_title = set(tfidf_model_title.get_feature_names())
```

In [31]:

```
# Using above lines of code
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_title = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_title): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_title):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary_title[word]*(sentence.count(word)/len(sentence.split())) # getting
the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_title.append(vector)
```

```
print(len(tfidf_w2v_vectors_title))
print(len(tfidf_w2v_vectors_title[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 109248/109248
[00:03<00:00, 30845.33it/s]
```

```
109248
300
```

1.5.3 Vectorizing Numerical features

In [32]:

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

In [33]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

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

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

```
Mean : 298.1193425966608, Standard deviation : 367.49634838483496
```

In [34]:

```
price_standardized
```

Out[34]:

```
array([[-0.3905327 ],
       [ 0.00239637],
       [ 0.59519138],
       ...,
       [-0.15825829],
       [-0.61243967],
       [-0.51216657]])
```

1.5.4 Merging all the above features

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

In [35]:

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

```
(109248, 9)
```

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

In [36]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
X.shape
```

Out[36]:

```
(109248, 16663)
```

In [37]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# 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
```

Computing Sentiment Scores

In [38]:

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

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

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students with the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelligences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of different backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a caring community of successful \
learners which can be seen through collaborative student project based learning in and out of the classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities to practice a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with real food i will take their idea \
and create common core cooking lessons where we learn important math and writing concepts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade apple sauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cookbooks to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoyment for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')
```

```
# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975
```

```
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

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_` values
- step 2 Compute the co-occurance matrix with these 2k words, with window size=5 (ref)

- step 3 Use TruncatedSVD on calculated co-occurance matrix and reduce its dimensions, choose the number of components (`n_components`) using elbow method

  > - 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**
- **step 6:Hyper parameter tuning (Consider any two hyper parameters)**
  - **Find the best hyper parameter which will give the maximum AUC value**
  - **Find the best hyper paramter using k-fold cross validation or simple cross validation data**
  - **Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning**

In [39]:

```
import sys
import math

import numpy as np
from sklearn.model_selection import learning_curve, 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_round, ve
rbose_eval=1)

    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
```

Computing Sentimental Scores

In [172]:

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

s = SentimentIntensityAnalyzer()

neg = []
pos = []
neu = []
comp = []

for sentence in tqdm(project_data['essay'].values) :

    Sscore = s.polarity_scores( sentence )

    neg.append( Sscore['neg'] )
    pos.append( Sscore['pos'] )
    neu.append( Sscore['neu'] )
    comp.append( Sscore['compound'] )
```

```
100%|██████████████████████████████████████████████████████████████████████████| 109248/109248
[04:46<00:00, 380.75it/s]
```

In [173]:

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

Computing no. of words in Essay and Title

In [175]:

```
''' To count number of words in a column'''

def totalWords(column):

    words = []
    for sent in tqdm( project_data[column].values ) :
        w = 0

        for word in sent.split():
            w += 1

        words.append(w)

    return words
```

In [176]:

```
project_data['totalWordsEssay'] = totalWords('essay')
```

```
100%|████████████████████████████████████████████████████████████████████████| 109248/109248
[00:02<00:00, 39219.11it/s]
```

In [177]:

```
project_data['totalWordsTitle'] = totalWords('project_title')
```

```
100%|███████████████████████████████████████████████████████████████████████| 109248/109248
[00:00<00:00, 995979.57it/s]
```

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

In [320]:

```
from sklearn.model_selection import train_test_split
Donor_train, Donor_test, Approved_train, Approved_test = train_test_split(project_data, project_dat
a['project_is_approved'], test_size=0.33, stratify=project_data['project_is_approved'])
```

In [321]:

```
print(Donor_train.shape,Approved_train.shape)
print(Donor_test.shape,Approved_test.shape)
project_data.columns
```

```
(73196, 27) (73196,)
(36052, 27) (36052,)
```

Out[321]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', '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',
       'clean_categories', 'clean_subcategories', 'essay', 'price', 'quantity',
       'neg', 'pos', 'neu', 'comp', 'totalWordsEssay', 'totalWordsTitle',
       'preprocessed_grade_category'],
      dtype='object')
```

2. TruncatedSVD

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

In [0]:

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

Top 2k words from Project Essay + Titles

In [92]:

```
idf_score_essay = vectorizer_essay.idf_
feature_names_essay  = vectorizer_essay.get_feature_names()
```

In [93]:

```
idfscore_essayFeat=[]
for i in range(len(idf_score_essay)):
    idfscore_essayFeat.append([idf_score_essay[i],feature_names_essay[i]])
```

In [95]:

```
idf_score_title = vectorizer_title.idf_
feature_names_title  = vectorizer_title.get_feature_names()
```

In [96]:

```
idfscore_titleFeat=[]
for i in range(len(idf_score_title)):
    idfscore_titleFeat.append([idf_score_title[i],feature_names_title[i]])
```

In [103]:

```
idfscore_text = idfscore_essayFeat + idfscore_titleFeat
```

In [105]:

```
idfscore_text.sort(reverse=True)
idfscore_text = idfscore_text[:2000]
for i in idfscore_text[:10]:
    print(i)
```

```
[10.203489686799497, 'zoo']
[10.203489686799497, 'zao']
[10.203489686799497, 'yummy']
[10.203489686799497, 'yrs']
[10.203489686799497, 'york']
[10.203489686799497, 'yeah']
[10.203489686799497, 'yahoo']
[10.203489686799497, 'xerox']
[10.203489686799497, 'wth']
[10.203489686799497, 'woodwinds']
```

In [439]:

```
project_data['combined _text'] = project_data['essay'] + ' ' + project_data['project_title']
```

In [ ]:

```
%%time
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizertpe = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizertpe.fit(Donor_train['essay'].values) # fit has to happen only on train data

Donor_train_essay_tfidf = vectorizertpe.transform(Donor_train['essay'].values)
Donor_test_essay_tfidf = vectorizertpe.transform(Donor_test['essay'].values)

print("After vectorizing Project Essays TFIDF")
print(Donor_train_essay_tfidf.shape, Approved_train.shape)
print(Donor_test_essay_tfidf.shape, Approved_test.shape)
print("="*100)
```

In [440]:

```
%%time
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_textcombined= TfidfVectorizer(min_df = 15)
vector_textcombined = tfidf_textcombined.fit_transform( project_data['combined _text'].values )

print( vector_textcombined.shape )
```

```
(109248, 14896)
Wall time: 27.4 s
```

In [443]:

```
ind = np.argsort( tfidf_textcombined.idf_ )[ ::-1 ]
textfeatures = tfidf_textcombined.get_feature_names()
```

In [444]:

```
n = 2000

topFeatures = [ textfeatures[i] for i in ind[:n] ]
topIDF = [ tfidf_textcombined.idf_[i] for i in ind[:n] ]

featIDF = list( zip(topFeatures, topIDF) )

print( featIDF[:20] )
```

```
[('nlittlebits', 9.828796237358086), ('dan', 9.828796237358086), ('transgender',
9.828796237358086), ('ann', 9.828796237358086), ('debris', 9.828796237358086), ('tranquility',
9.828796237358086), ('invoke', 9.828796237358086), ('powder', 9.828796237358086), ('rims',
9.828796237358086), ('daniel', 9.828796237358086), ('apprehension', 9.828796237358086), ('damper',
9.828796237358086), ('jingle', 9.828796237358086), ('apraxia', 9.828796237358086), ('noftentimes',
9.828796237358086), ('rim', 9.828796237358086), ('tolkien', 9.828796237358086), ('prefect',
9.828796237358086), ('prefers', 9.828796237358086), ('pregnant', 9.828796237358086)]
```

In [446]:

```
print( len(topFeatures) )
print( topFeatures[:20] )
```

```
2000
['nlittlebits', 'dan', 'transgender', 'ann', 'debris', 'tranquility', 'invoke', 'powder', 'rims',
'daniel', 'apprehension', 'damper', 'jingle', 'apraxia', 'noftentimes', 'rim', 'tolkien',
'prefect', 'prefers', 'pregnant']
```

2.2 Computing Co-occurance matrix

Co-occurance matrix for Project Essay + Title (Text Data)

Source: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

In [418]:

```
def COmatrix( data, words, cw=5 ):

    cm = pd.DataFrame( np.zeros((len(words), len(words))), index=words, columns=words )

    for sent in data:

        word = sent.split()

        for ind in range( len(word) ):

            if cm.get( word[ind] ) is None:
                continue

            for i in range(1, cw + 1 ):

                if ind - i >= 0:
                    if cm.get( word[ind - i] ) is not None:

                        cm[word[ind-i]].loc[word[ind]] =  (cm.get( word[ind-i] ).loc[ word[ind] ] +
1)
                        cm[word[ind]].loc[ word[ind-i] ] = (cm.get( word[ind] ).loc[ word[ind-i] ] +
1)

                if ind + i < len(word):
                    if cm.get( word[ind+i] ) is not None:

                        cm[ word[ind+i]].loc[word[ind]] =  (cm.get( word[ind+i] ).loc[ word[ind] ] +
1)
                        cm[word[ind]].loc[ word[ind+i] ] = (cm.get( word[ind] ).loc[ word[ind+i] ] +
1)


    np.fill_diagonal( cm.values, 0 )
    cm = cm.div(2)

    return cm
```

Sample Text for co occurence matrix

In [266]:

```
import pandas as pd
import numpy as np

sample_corpus = ['ABC DEF IJK PQR' ,'PQR KLM OPQ','LMN PQR XYZ ABC DEF PQR ABC']

df_sample = pd.DataFrame()
df_sample['text'] = sample_corpus

df_sample.head()
```

Out[266]:

	text
0	ABC DEF IJK PQR
1	PQR KLM OPQ
2	LMN PQR XYZ ABC DEF PQR ABC

In [449]:

```
top_words = ['ABC','PQR', 'DEF']
a = COmatrix(df_sample['text'],top_words,2)
a
```

Out[449]:

	ABC	PQR	DEF
ABC	0.0	3.0	3.0
PQR	3.0	0.0	2.0
DEF	3.0	2.0	0.0

Following code works to compute co occurence matrix

performing on Donor choose Text

In [108]:

```
preprocessed_text = preprocessed_essays + preprocessed_title
```

In [420]:

```
df_text = pd.DataFrame()
df_text['text'] = preprocessed_text
df_text.head()
```

Out[420]:

	text
0	my students english learners working english s...
1	our students arrive school eager learn they po...
2	true champions not always ones win guts by mia...
3	i work unique school filled esl english second...
4	our second grade classroom next year made arou...

In [447]:

```
cooccur_matrix_text = COmatrix(df_text['text'], topFeatures, 5)
cooccur_matrix_text
```

Out[447]:

	nlittlebits	dan	transgender	ann	debris	tranquility	invoke	powder	rims	daniel	...	bmi	introductions	nitems	catast
nlittlebits	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
dan	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
transgender	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
ann	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
debris	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
outlining	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
tweens	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
boss	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
harris	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
mend	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

2000 rows × 2000 columns

In [448]:

```
cooccur_matrix_text.sum()
```

Out[448]:

```
nlittlebits      0.0
dan              0.0
transgender     14.0
ann              1.0
debris           4.0
                ... 
outlining        0.0
tweens           1.0
boss             0.0
harris           0.0
mend             1.0
Length: 2000, dtype: float64
```

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

In [65]:

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

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

In [276]:

```
## Source :
https://chrisalbon.com/machine_learning/feature_engineering/select_best_number_of_components_in_ts


def select_n_components(var_ratio, goal_var: float) -> int:
    # Set initial variance explained so far
    total_variance = 0.0

    # Set initial number of features
    n_components = 0

    # For the explained variance of each feature:
    for explained_variance in var_ratio:

        # Add the explained variance to the total
        total_variance += explained_variance

        # Add one to the number of components
        n_components += 1

        # If we reach our goal level of explained variance
        if total_variance >= goal_var:
            # End the loop
            break

    # Return the number of components
    return n_components
```

In [450]:

```
from sklearn.decomposition import TruncatedSVD

# Dim-reduction using Truncated SVD

svd = TruncatedSVD( n_components = 1999, random_state=42 )
trsvd_text = svd.fit_transform(cooccur_matrix_text)
cumVarianceExplained = np.cumsum( svd.explained_variance_ratio_ )
```

In [451]:

```
select_n_components(cumVarianceExplained,95.0)
```

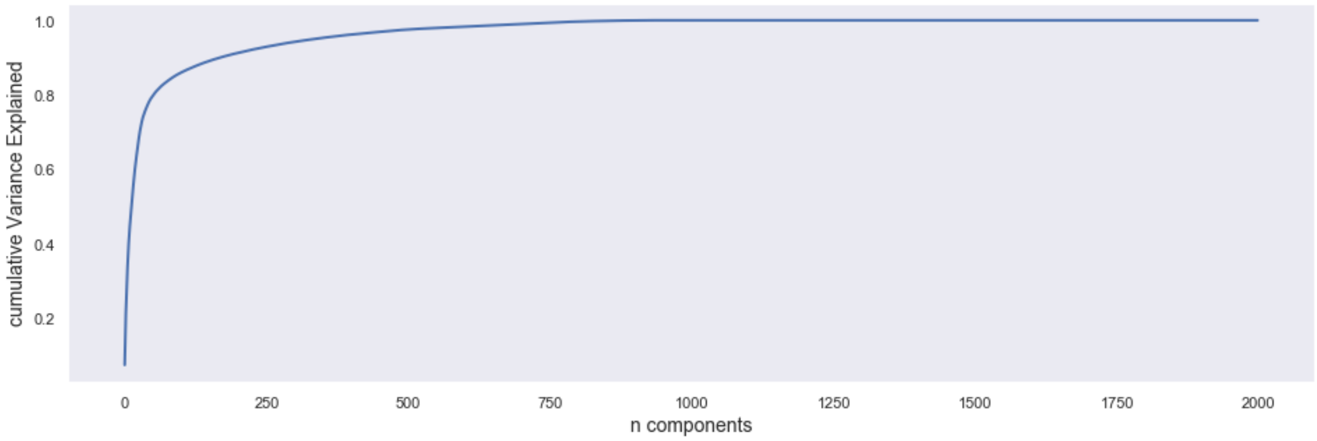
Out[451]:

```
127
```

In [452]:

```
import matplotlib.pyplot as plt1

plt1.figure( figsize=(16, 5))
```

```
plt1.plot( cumVarianceExplained, linewidth = 2 )
plt1.grid()
plt1.xlabel('n components',size=14)
plt1.ylabel('cumulative Variance Explained',size=14)
plt1.show()
```



Chosing n_component as 127 as 95% is covered

In [453]:

```
from sklearn.decomposition import TruncatedSVD

# Dim-reduction using Truncated SVD

svd = TruncatedSVD( n_components = 127, random_state = 42 )
textsvd = svd.fit_transform(coOccurenceMatrix_text)
cumVarianceExplained = np.cumsum( svd.explained_variance_ratio_ )
```

In [454]:

```
print(textsvd.shape)
```

```
(2000, 127)
```

In [457]:

```
top2ktext_df = pd.DataFrame( textsvd, index = cooccur_matrix_text.index)
top2ktext_df.head(4)
```

Out[457]:

	0	1	2	3	4	5	6	7	8	9	...	117	118	119	120	121	122	123	124	125	126
nlittlebits	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
dan	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
transgender	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ann	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

4 rows × 127 columns

Creating a dictionary with word as key and vector as value

In [458]:

```
index = list(topFeatures)
print(len(index))
print(index[:20])
```

```
2000
['nlittlebits', 'dan', 'transgender', 'ann', 'debris', 'tranquility', 'invoke', 'powder', 'rims',
'daniel', 'apprehension', 'damper', 'jingle', 'apraxia', 'noftentimes', 'rim', 'tolkien',
'prefect', 'prefers', 'pregnant']
```

In [459]:

```
vectorDict = dict()
count = 0

for i in textsvd:

    vectorDict[index[count] ] = i
    count += 1
```

In [466]:

```
vectorDict['transgender'][:10]
```

Out[466]:

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

In [467]:

```
vectorDict['transgender'].shape
```

Out[467]:

```
(127,)
```

In [468]:

```
svdFeatures = topFeatures[:127]
```

Vectorizing Textual Data

In [123]:

```
''' Creating a list that contains all the words in the data column '''

def senttowords( data ):
    sentence = []

    for sen in tqdm( data.values ):
        fsentence = []

        for w in sen.split():
            for cw in w.split():

                if cw.isalpha():
                    fsentence.append( cw.lower() )
                else:
                    continue
        sentence.append( fsentence )
    return sentence
```

In [471]:

```
''' AvgW2V defining '''

def avgw2v( data, words ):
    sentV = [] # average word 2 vec for each essay is stored in this

    for sent in tqdm(data):

        svec = np.zeros(127)
        cnw = 0
        for w in sent:
            if w in words:
                vec = vectorDict[ w ]  # Computing it's vector
                svec += vec      # Add it to the svec
                cnw += 1

        if cnw != 0:
            svec /= cnw  # Averaging with the count of number of words with valid vector in the Es
ay
        sentV.append( svec )

    return sentV
```

Train Text Data vectorize

In [469]:

```
## Vectorizing Train Data using essay text

essayTrainfinal = senttowords(Donor_train['essay'])
print(len(essayTrainfinal))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 73196/73196
[00:19<00:00, 3844.98it/s]
```

```
73196
```

In [472]:

```
essayTrainAW2V = np.asarray( avgw2v( essayTrainfinal, svdFeatures ) )
essayTrainAW2V.shape
```

```
100%|████████████████████████████████████████████████████████████████████████████| 73196/73196
[00:57<00:00, 1279.48it/s]
```

Out[472]:

```
(73196, 127)
```

In [474]:

```
titleTrainfinal = senttowords(Donor_train['project_title'])
print(len(titleTrainfinal))
```

```
100%|███████████████████████████████████████████████████████████████████████████| 73196/73196
[00:01<00:00, 72892.82it/s]
```

```
73196
```

In [475]:

```
titleTrainAW2V = np.asarray( avgw2v( titleTrainfinal, svdFeatures ) )
titleTrainAW2V.shape
```

```
100%|███████████████████████████████████████████████████████████████████████████| 73196/73196
[00:01<00:00, 44185.35it/s]
```

Out[475]:

```
(73196, 127)
```

Test Text Data Vectorize

In [476]:

```
essayTestfinal = senttowords(Donor_test['essay'])
print(len(essayTestfinal))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 36052/36052
[00:12<00:00, 2828.44it/s]
```

```
36052
```

In [477]:

```
essayTestAW2V = np.asarray( avgw2v( essayTestfinal, svdFeatures ) )
essayTestAW2V.shape
```

```
100%|██████████████████████████████████████████████████████████████████████████| 36052/36052
[00:27<00:00, 1325.88it/s]
```

Out[477]:

```
(36052, 127)
```

In [478]:

```
titleTestfinal = senttowords(Donor_test['project_title'])
print(len(titleCVfinal))
```

```
100%|████████████████████████████████████████████████████████████████████████| 36052/36052
[00:00<00:00, 198839.74it/s]
```

```
24155
```

In [479]:

```
titleTestAW2V = np.asarray( avgw2v( titleTestfinal, svdFeatures ) )
titleTestAW2V.shape
```

```
100%|████████████████████████████████████████████████████████████████████████| 36052/36052
[00:00<00:00, 112200.16it/s]
```

Out[479]:

```
(36052, 127)
```

Make Data Model Ready: encoding numerical, categorical features

In [330]:

```
# One hot Encoding for School State

vectorizerss = CountVectorizer()
vectorizerss.fit(Donor_train['school_state'].values) # fit has to happen only on train data
Donor_train_state_ohe = vectorizerss.transform(Donor_train['school_state'].values)
Donor_test_state_ohe = vectorizerss.transform(Donor_test['school_state'].values)

# Print One Hot Encoding - School State output
print("After vectorizations School state")
print(Donor_train_state_ohe.shape, Approved_train.shape)
print(Donor_test_state_ohe.shape, Approved_test.shape)
print(vectorizerss.get_feature_names())
print("="*100)
```

```
After vectorizations School state
(73196, 51) (73196,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv
', 'wy']
===================================================================================================
```

In [317]:

```python
## Preprocessing Grade category

grade_catogories = list(project_data['project_grade_category'].values)

grade_cat_list = []
for i in grade_catogories:
    i = i.replace('-',' ')
    i = i.replace('.','')
    grade_cat_list.append(i.strip())

project_data['preprocessed_grade_category'] = grade_cat_list
```

In [331]:

```python
# Preprocessing Project grade
from collections import Counter
my_counter = Counter()
for word in project_data['preprocessed_grade_category'].values:
    my_counter.update(word.split("'"))

project_grade_category_dict = dict(my_counter)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lambda
kv: kv[1]))
vectorizergc = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()),
lowercase=False, binary=True)

# One Hot Encoding - Project grade category

vectorizergc.fit(Donor_train['project_grade_category'].values) # fit has to happen only on train
data
Donor_train_grade_ohe = vectorizergc.transform(Donor_train['project_grade_category'].values)
Donor_test_grade_ohe = vectorizergc.transform(Donor_test['project_grade_category'].values)

# Print One Hot Encoding - Project grade output
print("After vectorizations Project grade category")
print(Donor_train_grade_ohe.shape, Approved_train.shape)
print(Donor_test_grade_ohe.shape, Approved_test.shape)
print(vectorizergc.get_feature_names())
print("="*100)
```

```
After vectorizations Project grade category
(73196, 4) (73196,)
(36052, 4) (36052,)
['Grades 9 12', 'Grades 6 8', 'Grades 3 5', 'Grades PreK 2']
===================================================================================================
```

In [333]:

```python
# One hot Encoding for project subject categories
vectorizercc = CountVectorizer()
vectorizercc.fit(Donor_train['clean_categories'].values) # fit has to happen only on train data
Donor_train_clean_cat_ohe = vectorizercc.transform(Donor_train['clean_categories'].values)
Donor_test_clean_cat_ohe = vectorizercc.transform(Donor_test['clean_categories'].values)

# Print One Hot Encoding - Project subject output
print("After vectorizations project subject categories")
print(Donor_train_clean_cat_ohe.shape, Approved_train.shape)
print(Donor_test_clean_cat_ohe.shape, Approved_test.shape)
print(vectorizercc.get_feature_names())
print("="*100)
```

```
After vectorizations project subject categories
(73196, 9) (73196,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language'
```

```
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================================
```

In [334]:

```
# One hot Encoding for project subject subcategories
vectorizercs = CountVectorizer()
vectorizercs.fit(Donor_train['clean_subcategories'].values) # fit has to happen only on train data
Donor_train_clean_subcat_ohe = vectorizercs.transform(Donor_train['clean_subcategories'].values)
Donor_test_clean_subcat_ohe = vectorizercs.transform(Donor_test['clean_subcategories'].values)

# Print One Hot Encoding - project subject subcategories output
print("After vectorizations project subject subcategories")
print(Donor_train_clean_subcat_ohe.shape, Approved_train.shape)
print(Donor_test_clean_subcat_ohe.shape, Approved_test.shape)
print(vectorizercs.get_feature_names())
print("="*100)
```

```
After vectorizations project subject subcategories
(73196, 30) (73196,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
====================================================================================================
```

In [335]:

```
# One hot Encoding for Teacher Prefix

Donor_train['teacher_prefix'] = Donor_train['teacher_prefix'].fillna(0)
Donor_cv['teacher_prefix'] = Donor_cv['teacher_prefix'].fillna(0)
Donor_test['teacher_prefix'] = Donor_test['teacher_prefix'].fillna(0)

vectorizertp = CountVectorizer()
vectorizertp.fit(Donor_train['teacher_prefix'].values.astype('U')) # fit has to happen only on
train data
Donor_train_teacher_ohe = vectorizertp.transform(Donor_train['teacher_prefix'].values.astype('U'))
Donor_test_teacher_ohe = vectorizertp.transform(Donor_test['teacher_prefix'].values.astype('U'))

# Print One Hot Encoding - Teacher Prefix output
print("After vectorizations Teacher Prefix")
print(Donor_train_teacher_ohe.shape, Approved_train.shape)
print(Donor_test_teacher_ohe.shape, Approved_test.shape)
print(vectorizertp.get_feature_names())
print("="*100)
```

```
After vectorizations Teacher Prefix
(73196, 5) (73196,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
====================================================================================================
```

In [336]:

```
from sklearn.preprocessing import Normalizer
normalizerp = Normalizer()
normalizerp.fit(Donor_train['price'].values.reshape(-1,1))
Donor_train_price_norm = normalizerp.transform(Donor_train['price'].values.reshape(-1,1))
Donor_test_price_norm = normalizerp.transform(Donor_test['price'].values.reshape(-1,1))

print("After vectorizations Numerical Data: Price")
print(Donor_train_price_norm.shape, Approved_train.shape)
print(Donor_test_price_norm.shape, Approved_test.shape)
print("="*100)
```

```
After vectorizations Numerical Data: Price
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

In [337]:

```
from sklearn.preprocessing import Normalizer
normalizert = Normalizer()
normalizert.fit(Donor_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
Donor_train_postedCount_norm =
normalizert.transform(Donor_train['teacher_number_of_previously_posted_projects'].values.reshape(-
1,1))
Donor_test_postedCount_norm =
normalizert.transform(Donor_test['teacher_number_of_previously_posted_projects'].values.reshape(-1
,1))

print("After vectorizations Numerical Data: Previously Posted Projects")
print(Donor_train_postedCount_norm.shape, Approved_train.shape)
print(Donor_test_postedCount_norm.shape, Approved_test.shape)
print("="*100)
```

```
After vectorizations Numerical Data: Previously Posted Projects
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

In [338]:

```
from sklearn.preprocessing import Normalizer
normalizerq = Normalizer()
normalizerq.fit(Donor_train['quantity'].values.reshape(-1,1))
Donor_train_quantity_norm = normalizerq.transform(Donor_train['quantity'].values.reshape(-1,1))
Donor_test_quantity_norm = normalizerq.transform(Donor_test['quantity'].values.reshape(-1,1))

print("After vectorizations Numerical Data: Quantity")
print(Donor_train_quantity_norm.shape, Approved_train.shape)
print(Donor_test_quantity_norm.shape, Approved_test.shape)
print("="*100)
```

```
After vectorizations Numerical Data: Quantity
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

In [339]:

```
## previously posted projects

normalizerppp = Normalizer()
normalizerppp.fit(Donor_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

Donor_train_ppp_norm =
normalizerppp.transform(Donor_train['teacher_number_of_previously_posted_projects'].values.reshape
(-1,1))
Donor_test_ppp_norm =
normalizerppp.transform(Donor_test['teacher_number_of_previously_posted_projects'].values.reshape(
-1,1))

print("After vectorizations Numerical Data: previously posted projects")
print(Donor_train_ppp_norm.shape, Approved_train.shape)
print(Donor_test_ppp_norm.shape, Approved_test.shape)
print("="*100)
```

```
After vectorizations Numerical Data: previously posted projects
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

In [340]:

```
## Sentiment Score

Donor_train_neg = Donor_train['neg'].values.reshape(-1,1)
Donor_test_neg = Donor_test['neg'].values.reshape(-1,1)

Donor_train_pos = Donor_train['pos'].values.reshape(-1,1)
Donor_test_pos = Donor_test['pos'].values.reshape(-1,1)

Donor_train_neu = Donor_train['neu'].values.reshape(-1,1)
Donor_test_neu = Donor_test['neu'].values.reshape(-1,1)

Donor_train_comp = Donor_train['comp'].values.reshape(-1,1)
Donor_test_comp = Donor_test['comp'].values.reshape(-1,1)
```

In [341]:

```
## Total Words Essay

normalizertwe = Normalizer()
normalizertwe.fit(Donor_train['totalWordsEssay'].values.reshape(-1,1))

Donor_train_twe_norm = normalizertwe.transform(Donor_train['totalWordsEssay'].values.reshape(-1,1))
Donor_test_twe_norm = normalizertwe.transform(Donor_test['totalWordsEssay'].values.reshape(-1,1))

print("After vectorizations Numerical Data: Total Words Essay")
print(Donor_train_twe_norm.shape, Approved_train.shape)
print(Donor_test_twe_norm.shape, Approved_test.shape)
print("="*100)
```

```
After vectorizations Numerical Data: Total Words Essay
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

In [342]:

```
## Total Words Title

normalizertwt = Normalizer()
normalizertwt.fit(Donor_train['totalWordsTitle'].values.reshape(-1,1))

Donor_train_twt_norm = normalizertwt.transform(Donor_train['totalWordsTitle'].values.reshape(-1,1))
Donor_test_twt_norm = normalizertwt.transform(Donor_test['totalWordsTitle'].values.reshape(-1,1))

print("After vectorizations Numerical Data: Total Words Title")
print(Donor_train_twt_norm.shape, Approved_train.shape)
print(Donor_test_twt_norm.shape, Approved_test.shape)
print("="*100)
```

```
After vectorizations Numerical Data: Total Words Title
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

2.4 Merge the features from step 3 and step 4

In [0]:

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

In [480]:

```
from scipy.sparse import hstack

Donor_final_tr =
hstack((essayTrainAW2V,titleTrainAW2V,Donor_train_ppp_norm,Donor_train_neg,Donor_train_pos,Donor_tr
ain_neu,Donor_train_comp,Donor_train_twt_norm,Donor_train_twe_norm,Donor_train_state_ohe,
Donor_train_teacher_ohe, Donor_train_grade_ohe,
Donor_train_clean_cat_ohe,Donor_train_clean_subcat_ohe,Donor_train_price_norm,Donor_train_postedCou
nt_norm,Donor_train_quantity_norm)).tocsr()
Donor_final_te = hstack((essayTestAW2V,titleTestAW2V,Donor_test_ppp_norm,Donor_test_neg,Donor_test_
pos,Donor_test_neu,Donor_test_comp,Donor_test_twt_norm,Donor_test_twe_norm,Donor_test_state_ohe,
Donor_test_teacher_ohe, Donor_test_grade_ohe, Donor_test_clean_cat_ohe,Donor_test_clean_subcat_ohe
,Donor_test_price_norm,Donor_test_postedCount_norm,Donor_test_quantity_norm)).tocsr()

print("Final Donor Data Set")
print(Donor_final_tr.shape,Approved_train.shape)
print(Donor_final_te.shape,Approved_test.shape)
print("="*100)
```

```
Final Donor Data Set
(73196, 363) (73196,)
(36052, 363) (36052,)
====================================================================================================
```

2.5 Apply XGBoost on the Final Features from the above section

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

In [0]:

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

In [481]:

```
%%time

# source: https://qiita.com/bmj0114/items/8009f282c99b77780563

from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

learning_rate = [0.01, 0.02, 0.05, 0.06, 0.08, 0.1]
n_estimators = [20, 50, 100, 150, 200, 300, 400, 500]

parameters = {'learning_rate': learning_rate, 'n_estimators' : n_estimators}

clf_XGBT_finalData = GridSearchCV(XGBClassifier(booster='gbtree', class_weight = 'balanced' ),
parameters, cv = 5, return_train_score = True)
clf_XGBT_finalData.fit(Donor_final_tr, Approved_train)
```

```
Wall time: 1h 13min 24s
```

Out[481]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     class_weight='balanced',
                                     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=0, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='warn', n_jobs=None,
             param_grid={'learning_rate': [0.01, 0.02, 0.05, 0.06, 0.08, 0.1],
                         'n_estimators': [20, 50, 100, 150, 200, 300, 400,
                                          500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [482]:

```python
print(clf_XGBT_finalData.best_estimator_)
print(clf_XGBT_finalData.best_score_)
```

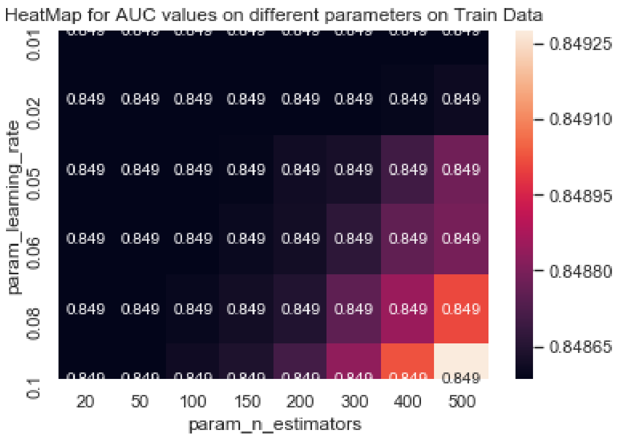
```
XGBClassifier(base_score=0.5, booster='gbtree', class_weight='balanced',
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              gamma=0, learning_rate=0.01, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=20, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
0.8485846221104978
```

In [483]:

```python
df_gridsearch = pd.DataFrame(clf_XGBT_finalData.cv_results_)
max_scores = df_gridsearch.groupby(['param_learning_rate','param_n_estimators']).max()
max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
sns.heatmap(max_scores.mean_train_score,annot=True, annot_kws={"size": 10}, fmt='.3g');
plt.title('HeatMap for AUC values on different parameters on Train Data')
```

Out[483]:

```
Text(0.5, 1, 'HeatMap for AUC values on different parameters on Train Data')
```

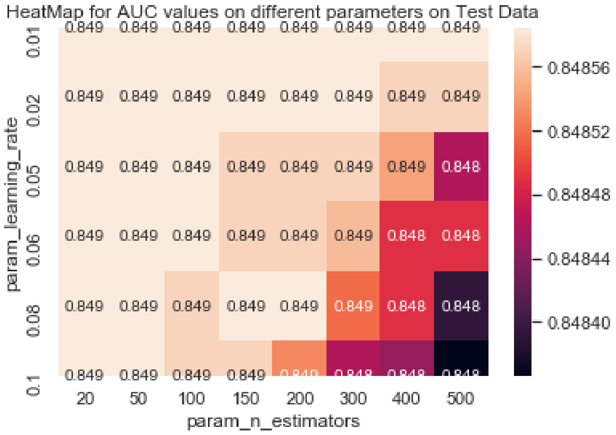


In [484]:

```python
df_gridsearch = pd.DataFrame(clf_XGBT_finalData.cv_results_)
max_scores = df_gridsearch.groupby(['param_learning_rate','param_n_estimators']).max()
max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
sns.heatmap(max_scores.mean_test_score,annot=True, annot_kws={"size": 10}, fmt='.3g');
plt.title('HeatMap for AUC values on different parameters on Test Data')
```

Out[484]:

```
Text(0.5, 1, 'HeatMap for AUC values on different parameters on Test Data')
```



In [489]:

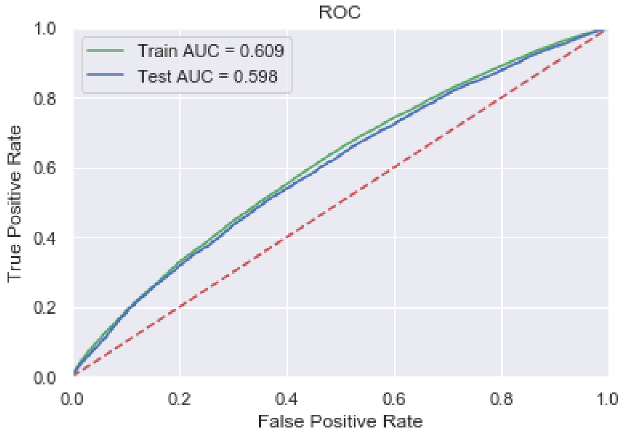
```
clf_XGBT_finalData_best = XGBClassifier(booster='gbtree', class_weight = 'balanced', gamma=0, learn
ing_rate=0.08,max_depth=2, n_estimators=200, n_jobs=4 )
clf_XGBT_finalData_best.fit(Donor_final_tr, Approved_train)

pred_test = clf_XGBT_finalData_best.predict_proba(Donor_final_te)[:,1]
pred_train = clf_XGBT_finalData_best.predict_proba(Donor_final_tr)[:,1]
```

In [490]:

```
fpr, tpr, thresholds = roc_curve(Approved_test, pred_test)
fpr2, tpr2, thresholds = roc_curve(Approved_train, pred_train)

score_test = roc_auc_score(Approved_test, pred_test)
score_train = roc_auc_score(Approved_train, pred_train)
```

In [491]:

```
roc_auc_test = metrics.auc(fpr, tpr)
roc_auc_train = metrics.auc(fpr2, tpr2)
plt.title('ROC')
plt.plot(fpr2, tpr2, 'g', label = 'Train AUC = %0.3f' % score_train)
plt.plot(fpr, tpr, 'b', label='Test AUC = %0.3f' % score_test)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```



In [358]:

```
def predictcm(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

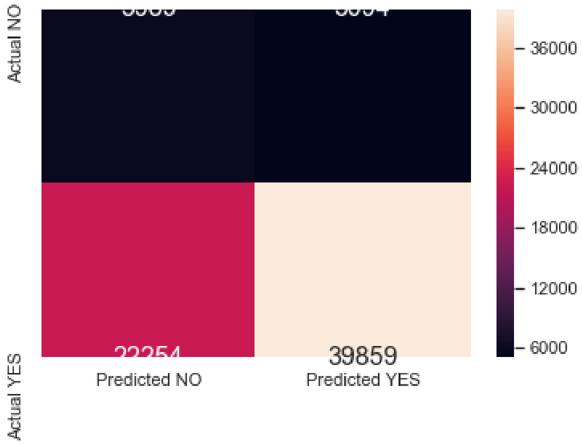
In [395]:

```
# Confustion Matrix for Train vs Test data
import matplotlib.pyplot as plt

print("Train confusion matrix ")
donor_cm_te = pd.DataFrame(confusion_matrix(Approved_train, predictcm(Approved_train_pred[:, 1],te_
thresholds, test_fpr, test_fpr)))
donor_cm_te.columns = ['Predicted NO','Predicted YES']
donor_cm_te = donor_cm_te.rename({0: 'Actual NO', 1: 'Actual YES'})


sns.set(font_scale=1.0)#for label size
sns.heatmap(donor_cm_te, annot=True,annot_kws={"size": 16}, fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999161092998 for threshold 0.842
```

Out[395]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x214eb86efc8>
```



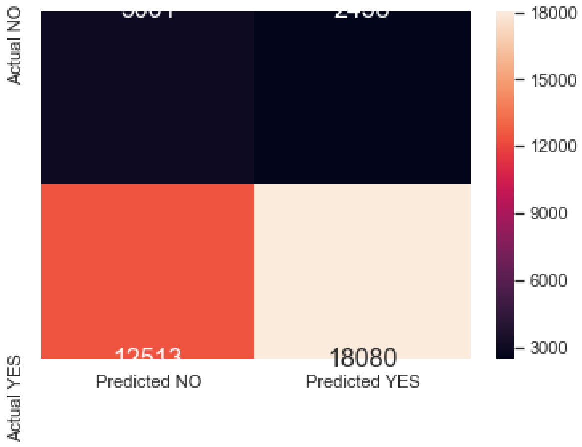
In [407]:

```
print("Test confusion matrix ")
donor_cm_tr = pd.DataFrame(confusion_matrix(Approved_test, predictcm(Approved_test_pred[:, 1],
tr_thresholds, train_fpr, train_tpr)))
donor_cm_tr.columns = ['Predicted NO','Predicted YES']
donor_cm_tr = donor_cm_tr.rename({0: 'Actual NO', 1: 'Actual YES'})
sns.set(font_scale=1.0)#for label size
sns.heatmap(donor_cm_tr, annot=True, annot_kws={"size": 16}, fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.3509743709027531 for threshold 0.847
```

Out[407]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x214ec9e7348>
```



3. Conclusion

In [492]:

```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Model", " Learning Rate","n_estimators", "Train AUC","Test AUC"]

x.add_row(["XGBoost", 0.08, 200, 60.19, 59.80])

print(x)
```

```
+---------+----------------+--------------+-----------+----------+
|  Model  |  Learning Rate | n_estimators | Train AUC | Test AUC |
+---------+----------------+--------------+-----------+----------+
| XGBoost |      0.08      |     200      |   60.19   |   59.8   |
+---------+----------------+--------------+-----------+----------+
```